MULTI-SOURCE ONTOLOGY-BASED MAIZE PHENOTYPE SEARCH ENGINE

A Thesis

presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

JASON GREEN

Dr. Chi-Ren Shyu, Thesis Supervisor

MAY 2009

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

MULTI-SOURCE ONTOLOGY-BASED

MAIZE PHENOTYPE SEARCH ENGINE

presented by Jason Green,

a candidate for the degree of Master of Science,

and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Chi-Ren Shyu

Dr. Dong Xu

Dr. Mary Schaeffer

To my mom and dad, for teaching me responsibility and hard work and for their unwavering support and love.

ACKNOWLEDGEMENTS

There are several people who have been instrumental in the development of this thesis. First and foremost, I would like to offer my sincere thanks to my advisor, Dr Chi-Ren Shyu, for teaching and pushing me to be a better researcher. Since I entered the MedBio lab, he has consistently provided guidance and motivation, which were vital to the completion of this thesis. I would also like to express appreciation to Dr Dong Xu and Dr. Mary Schaeffer for graciously offering to serve on my committee. A special thanks goes to Dr Schaeffer for her invaluable maize expertise and for suggestions and ideas that are sure to make this phenotype search engine useful to the entire plant community. These acknowledgements would not be complete without also thanking Jaturon Harnsomburana for hours of fruitful discussion.

I would also like to thank MaizeGDB, specifically Dr. Carolyn Lawrence and Dr Schaeffer, for providing data for this project. This project is supported by the National Science Foundation, grant #DBI-044794.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

MULTI-SOURCE ONTOLOGY-BASED MAIZE PHENOTYPE SEARCH ENGINE

Jason Green
Dr. Chi-Ren Shyu, Thesis Supervisor

ABSTRACT

In the midst of this genomics era, major plant genome databases are collecting massive amounts of heterogeneous information, including sequence data, gene product information, as well as images and descriptions of mutant phenotypes. While basic browsing and search capabilities are available to allow researchers to query and peruse the names and attributes of stored data, advanced search mechanisms that can take advantage of textual descriptions of various types of stored data are nonexistent. Furthermore, though much time and effort have been afforded to the development of plant-related ontologies, the knowledge embedded in these ontologies remains largely unused in available plant search mechanisms. Addressing both of these issues, we have developed a unique search engine for the phenotypes in MaizeGDB. This advanced search mechanism exploits the content and structure of available domain ontologies for the purposes of query enrichment, with currently both the Plant Ontology and Gene Ontology being utilized. The search engine also has the flexibility to integrate various text description sources to aid the user in retrieving desired phenotype information. This framework can be generalized to any domain with a domain-specific ontology or to sets of text sources that are heterogeneous and interconnected.

CHAPTER 1

INTRODUCTION

1.1. Motivation

Major plant genome databases like MaizeGDB [15], Gramene [12], TAIR [28], SGN [18], Soybase [27], and Oryzabase [14] have compiled and organized large quantities of data for their respective research communities. Though enormous amounts of new data are being generated and submitted, completion of the respective genomes is expected to increase the rate of data collection even more. Currently, each group provides browsing capabilities to allow individuals to sift through the available data as well as basic search mechanisms to aid users in locating specific information. As the amount and rate of submitted data increases, efficient and useful search mechanisms will become even more important to researchers interested in locating and accessing specific information stored in the database.

Plant researchers need flexible and accurate search mechanisms to retrieve documents related to their complicated queries. To illustrate some of the shortcomings of many search mechanisms available in the plant community, consider the following examples:

Suppose a maize researcher is looking for "all catalogued maize phenotypes linked to genes associated with the flower" in MaizeGDB. A typical available search mechanism would allow the researcher to search, for example, for phenotypes related to the body part (attribute) "flower". While this may return many phenotypes, it would fail to find phenotypes related to "pollen" or any other part of the flower, which are also relevant to the user's query. Also, current search mechanisms typically only reference entity names or attributes, and thus do not take advantage of the information available in free text fields. In addition, because other associated sources, e.g. description of genes linked to phenotypes which are stored separate from the phenotype information, may mention "flower" or a part of the flower, the basic search would also fail to find phenotypes that are linked to genes related to the flower in some way.

Consider a second example in which a tomato researcher is interested in finding "maize phenotypes that affect fruit color" for comparative purposes. Using available search mechanisms, the researcher is unlikely to find many relevant documents, as the vocabulary used by the two plant communities differs, e.g. "kernel" is the maize term that corresponds to "fruit" in tomato.

Domain ontologies may help to alleviate some of these issues. Large amounts of time, money, and energy have been put forth towards the development of several plant-related ontologies. These ontologies hierarchically arrange the standard terms used in the domain via a variety of relationships. These ontologies could be exploited, for example,

to identify plant anatomy that is part of the flower as well as to translate vocabulary between different plant organisms.

1.2. Goals of the Maize Phenotype Search Engine

Cross-species and complicated queries, like the examples discussed earlier, are becoming increasingly more important in the plant sciences as research in comparative and systems biology increases. Search mechanisms are needed that can more fully utilize stored information to better accommodate these kinds of queries and better meet the needs of plant researchers.

We have developed a flexible, advanced search mechanism in response to these needs. This maize phenotype search engine utilizes ontological structure and embedded knowledge for query expansion. This inclusion would, for example, improve the results of the first example query by allowing the search engine to additionally retrieve documents related to parts of a flower. It could also be used to relate “fruit” and “kernel” and thus retrieve more relevant documents for the tomato researcher. While the current search engine uses the Plant Ontology (PO) [11] and Gene Ontology (GO) [9], the system is designed to accommodate the addition and integration of other domain ontologies. This search mechanism is also designed to integrate related text sources to aid the user in retrieving desired information; for example, the first example query could be configured to search phenotype descriptions as well as associated locus descriptions so that a more complete list of flower-related phenotypes is retrieved. While only a few related text sources are currently integrated, the system is flexible in that additional sources can be easily added and incorporated into the search mechanism.

1.3. Thesis Outline

The remainder of this thesis is organized as follows: In CHAPTER 2, the specific browsing and search capabilities offered by each major plant genome group are discussed. A survey of existing maize-related ontologies is provided as well as a literature review of ontology utilization in retrieval systems. CHAPTER 3 describes the various data sources underlying the search mechanism as well as the database schema used as the backend of the retrieval engine. In CHAPTER 4, retrieval models are discussed, starting with the retrieval model that forms the basis for this search engine. The theoretical formulation of the proposed similarity measures is also developed. CHAPTER 5 discusses the preprocessing of documents from each text source, with query processing handled in CHAPTER 6. CHAPTER 7 describes the process of document case retrieval and ranking. In CHAPTER 8, the user interface as well as available options and adjustable parameters are discussed, with CHAPTER 9 describing issues related to the handling of data updates and the adding of new ontologies or text sources to the search. The thesis is concluded in CHAPTER 10 with areas for future work identified.

CHAPTER 2

LITERATURE REVIEW

This chapter focuses on three topics. First, the browsing and search capabilities currently available by the various plant genome groups are discussed. Second, the ontologies that are utilized in the developed system as well as other ontologies that may be integrated in the future are introduced. Finally, a survey of how ontologies have been used in information retrieval is performed.

2.1. Survey of Plant Genome Search Capabilities

All the major plant genome databases currently provide basic browsing and search capabilities for the data they maintain. The search mechanisms for each database are discussed below, with special attention paid to phenotype querying and search methods unique to a specific plant group. Though the search engine developed for this thesis is focused on maize, search mechanisms available for the plant community as a whole are surveyed, as the framework for this search engine is easily portable to other plant domains. As described below, the majority of current search mechanisms can be

accomplished via single Structured Query Language (SQL) queries. These search methods, referred to as SQL-based searches, include exact-match, prefix, suffix, and substring searches.

2.1.1. Maize Genetics and Genomics Database (MaizeGDB)

The maize plant genome database is equipped with a variety of browsing and search capabilities [15]. Specifically, this database offers simple search capabilities for the entirety of data types collected, including gene products, loci, maps, metabolic pathways, phenotypes/mutants, probes and molecular markers, QTL experiments, sequences, traits, stocks, and variations/polymorphisms/alleles. The search mechanisms are SQL-based and only retrieve items that exactly match the query, though the text-based query fields allow wildcard characters for substring searches. These Boolean-style searches, however, do not provide ranking of results by relevance, only a list of matching items. To help reduce the number of search results, filters oftentimes accompany these search mechanisms. Figure 1 shows the current interface for maize phenotype browsing and searching. For each filter, only a single option can be selected from each dropdown list.

2.1.2. Gramene

Gramene is a database resource for agriculturally important grasses including organisms like maize, soybean, rye, and wheat [12]. It is specifically designed to facilitate comparative genomics of grasses. Like the other genome databases, it contains searches

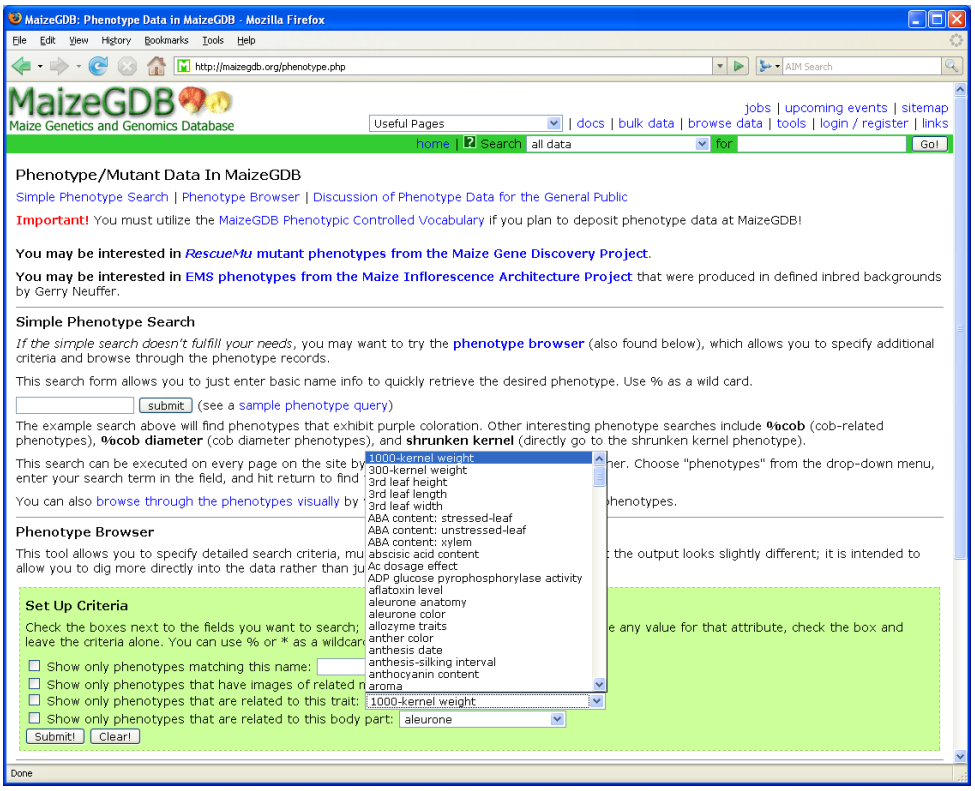

Figure 1: Screenshot of current MaizeGDB phenotype browsing and search capabilities.

for entities including markers, comparative maps, germplasms, phenotypes, polymorphisms, QTLs, proteins, sequences, and pathways. Some search mechanisms, like the phenotype search shown in Figure 2, allow a user to select query options from dropdown boxes. Other mechanisms allow text queries, which are parsed into words and treated as Boolean [2] queries with the terms combined with the OR operation. Other notable search mechanisms in Gramene include an ontology browser that searches the terms and definitions of up to seven ontologies, and a SQL-based gene search that examines descriptions of genes and alleles.

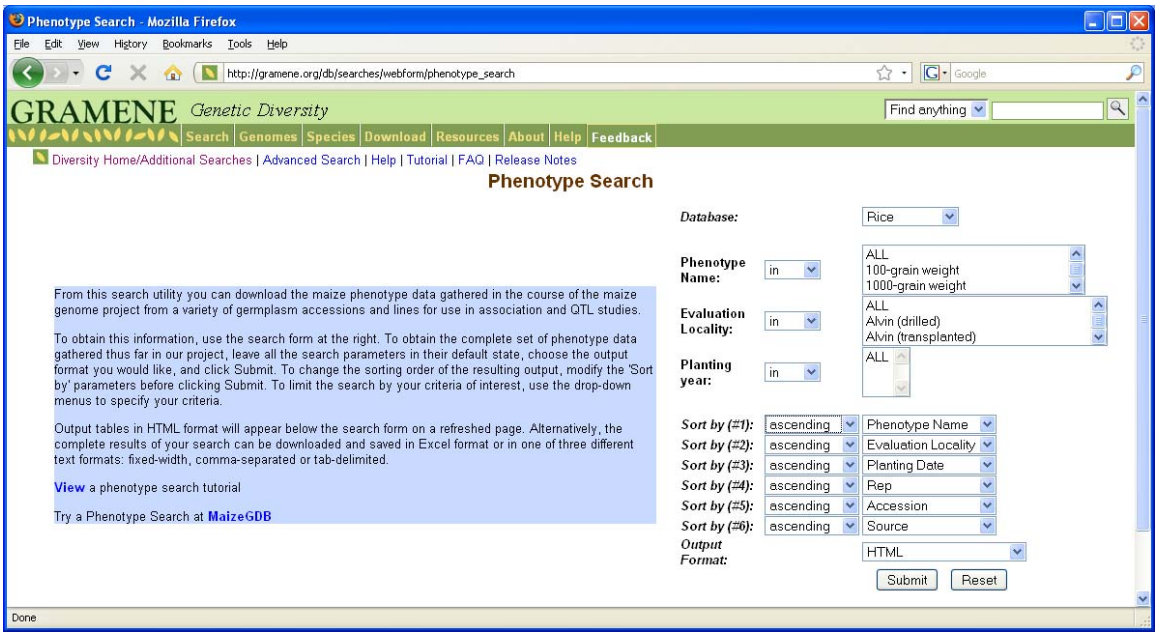

Figure 2: Screenshot of Gramene's phenotype search module.

2.1.3. The Arabidopsis Information Resource (TAIR)

The TAIR database collects genetic and molecular biology data for the model plant *Arabidopsis thaliana* [28]. This genome database also is outfitted with a number of search capabilities related to DNA/clones, ecotypes, genes, GO annotations, locus history, markers, microarray experiments, polymorphisms and alleles, proteins, seeds and germplasms, and sequences. These search mechanisms are generally SQL-based or Boolean in nature (see the example polymorphism search in Figure 3) and offer several filters. TAIR also allows substring searches of text-entered queries, but the implementation is different from MaizeGDB. Instead of using wildcard characters, dropdowns allow the user to specify whether retrieved items should "start with," "contain," "end with," or "exactly match" the query. Of note, TAIR does contain an

ontology keyword search that, while also being a Boolean search, utilizes an ontology to expand the query with synonyms.

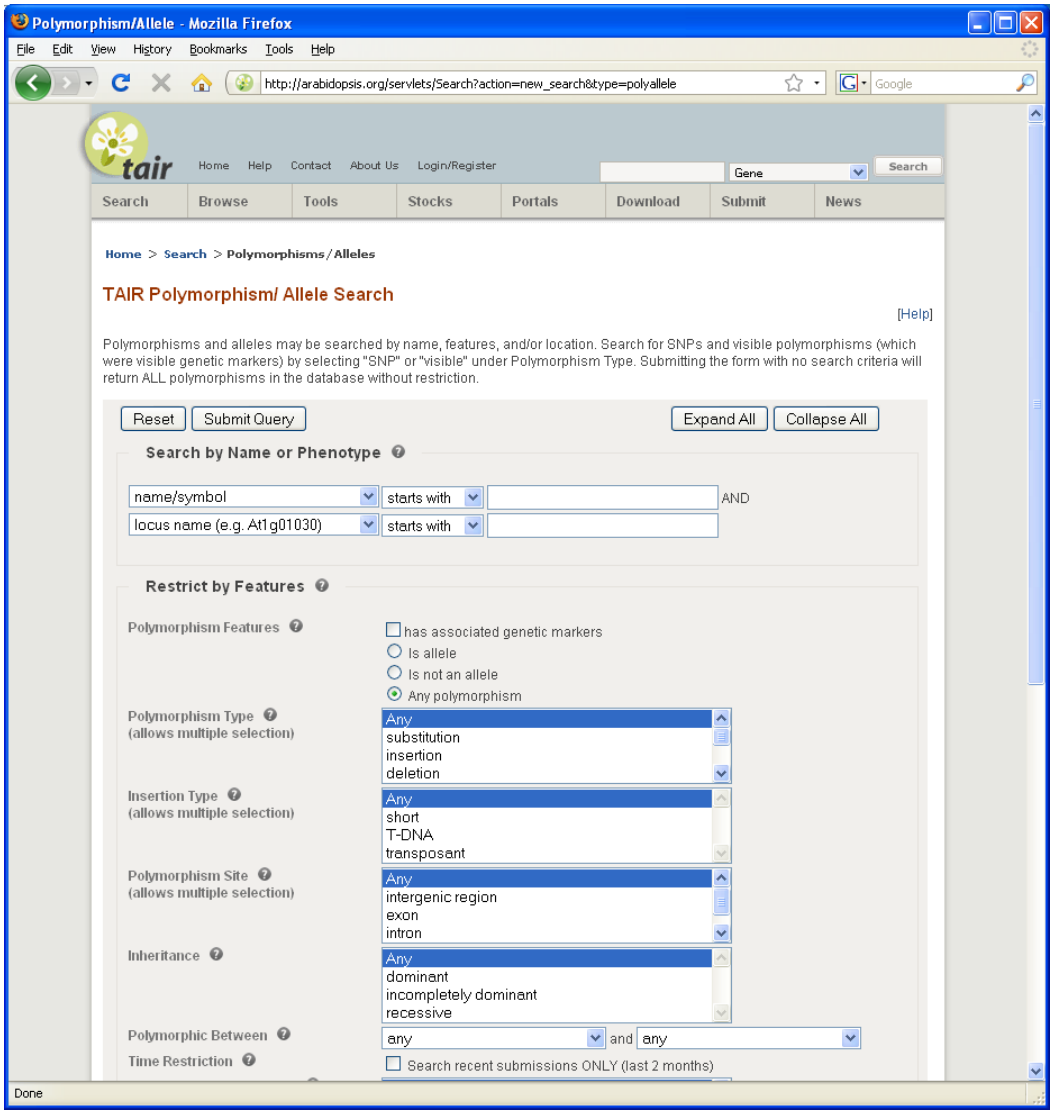

Figure 3: Example search interface for TAIR.

2.1.4. Sol Genomics Network (SGN)

The SGN database maintains genetic data for members of *Solanaceae*, including tomato, potato, pepper, and eggplant [18]. It, too, provides an array of SQL-based query mechanisms for searching its data, including searches of genes, phenotypes, QTLs and

traits, families, markers, genomic clones, annotations, and images. Most of SGN's search mechanisms provide filtering options as well. The phenotype search mechanism provides a means to include ontology terms or accessions in the query. These are entered in a separate text field, partitioning them from the keyword-based query. Wildcards may be included for prefix, suffix, and substring searches. As the phenotype descriptions are linked to images and associated to loci, the results page, shown in Figure 4, includes thumbnails of phenotype images as well as links to associated loci. The descriptions are not ranked, but rather are sorted by accession name.

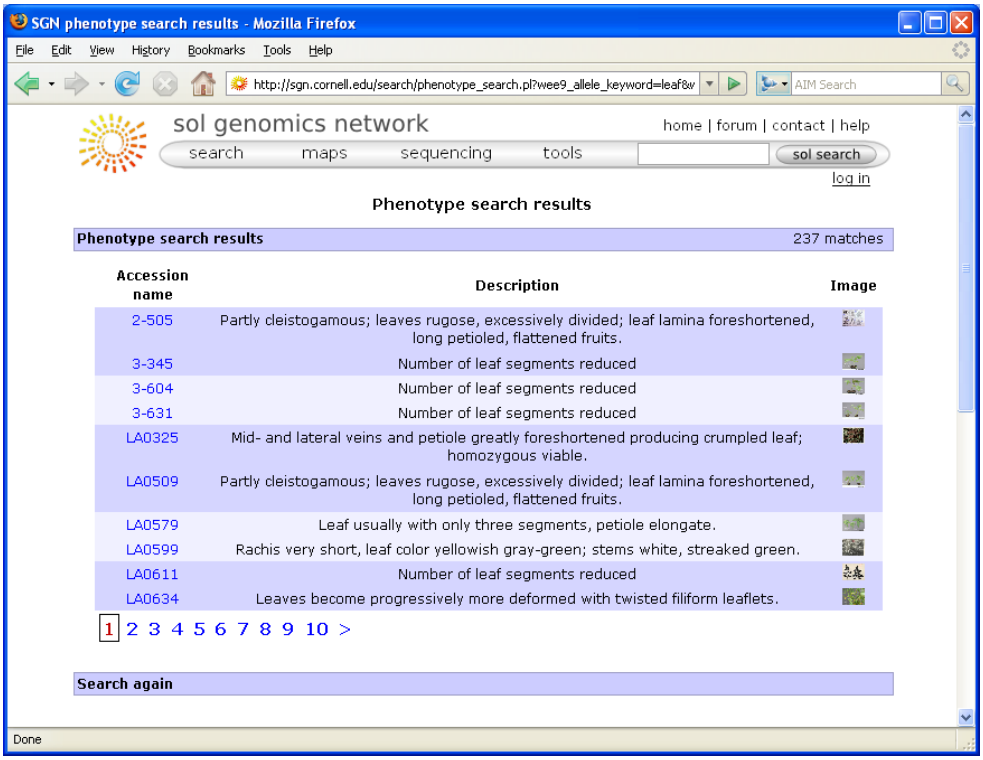

Figure 4: Phenotype search results for the *Solanaceae* genome database including image thumbnails.

2.1.5. Soybase

Soybase is the genomics database for soybean and contains similar browsing and search capabilities on loci, QTL, pathologies, diseases, genes, traits, and maps [27]. All searching and browsing are done using the interface in Figure 5. Queries search name fields only; no searches of entity descriptions are provided.

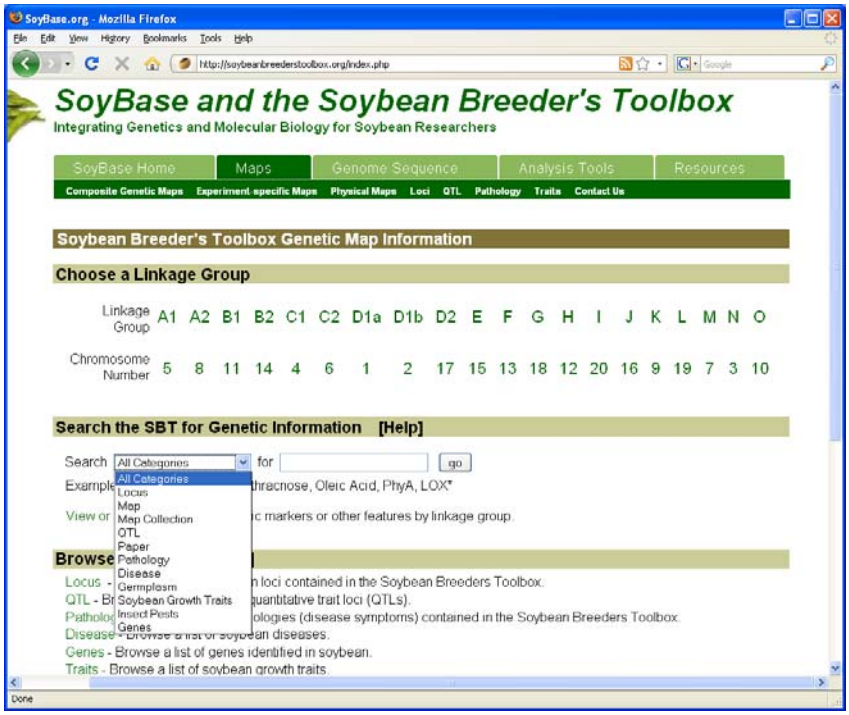

Figure 5: Searching and browsing at Soybase.

2.1.6. Oryzabase

Oryzabase is the integrated genomics database for the rice crop [14]. This genome database allows search of entities like strains, trait genes, mutant phenotypes, and markers. Linkage, physical, and comparative maps are also available for browsing. Searches are performed in a manner similar to the other plant genome databases. As an

example of the search interfaces available for this genomic database, the mutant phenotype search is shown in Figure 6.

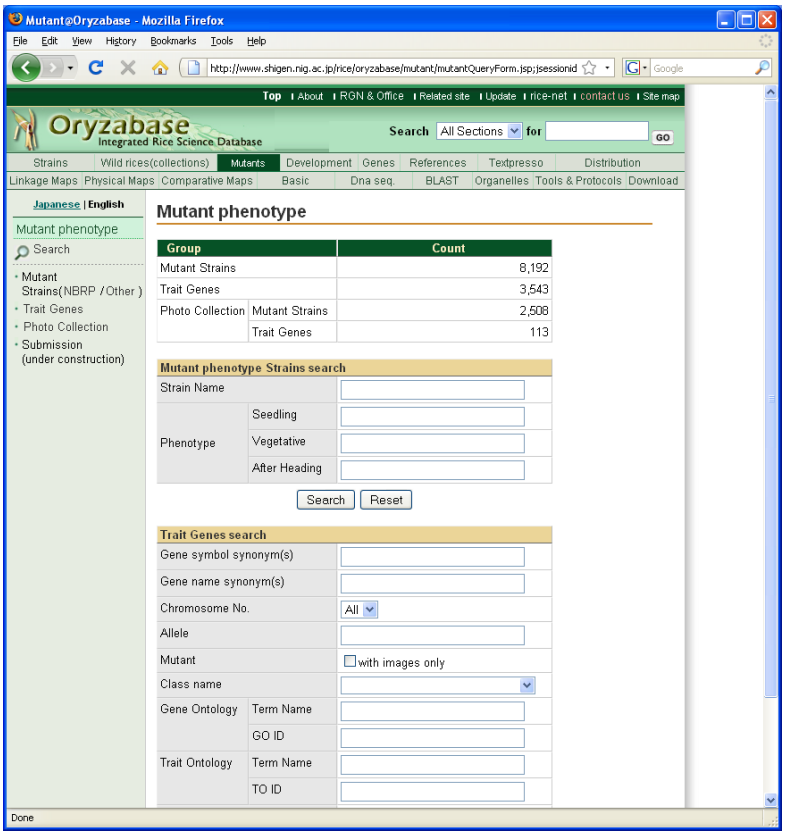

Figure 6: Phenotype search interface at Oryzabase.

A summary of the types of search mechanisms offered by the major plant genome databases is seen in Table I. It is observed that there is a large amount of overlap regarding the types of search capabilities offered by each group, though each group typically has a few unique fields to search. While a few groups do offer ontology search mechanisms, those currently available search for terms within an ontology. The ontologies are generally not used in conjunction with other available data for improving search results. Finally, note that very few groups perform search on entity descriptions,

and that no group attempts to rank results from any search for the user in terms of relevance. This is a direct result of the "exact-match" nature of these mechanisms. As descriptions of various entities are stored by these groups, a more advanced search mechanism that retrieves and ranks these descriptions could be developed. In addition, because of the variety of data collected and its interconnectedness, there is a need for more advanced retrieval tools that better integrate these different sources.

Table I: Summary of available search capabilities of the major plant genomic groups.

	MaizeGDB	Gramene	TAIR	SGN	Soybase	Oryzabase
Gene Products	√	√	√			
Genes/Loci	√	√	√	√	√	√
Locus History			√			
Maps	√	√		√	√	√
Pathways	√	√				
Phenotypes	√	√	√	√		√
Polymorphisms	√	√	√			√
Markers	√	√	√	√		
QTLs	√	√		√	√	
Sequences	√	√	√	√		√
Traits	√			√	√	
Germplasms/Stocks	√	√	√		√	
Ontology	√	√	√			
Species			√			
Microarrays			√			
DNA			√	√		
Image Metadata				√		
Diseases/Pathologies					√	
Description Search		√		√		
Result Ranking						

2.2. Survey of Existing Maize-Related Ontologies

Domain ontologies contain the standard vocabulary used within a specific domain and relate that vocabulary through relationships and hierarchies. Significant time and effort have been spent on the development of several ontologies that are relevant to phenotypes and the maize community. The terminology contained within each ontology can help to determine which ontologies are potentially useful in phenotype searches for maize.

2.2.1. Gene Ontology (GO)

The GO provides controlled vocabularies for describing "gene and gene product attributes in any organism" [9]. GO comprises three ontologies: (1) the biological process ontology, (2) the cellular component ontology, and (3) the molecular function ontology. Terms are organized into directed acyclic graphs (DAGs), which are similar to hierarchies except a term can have more than parent.

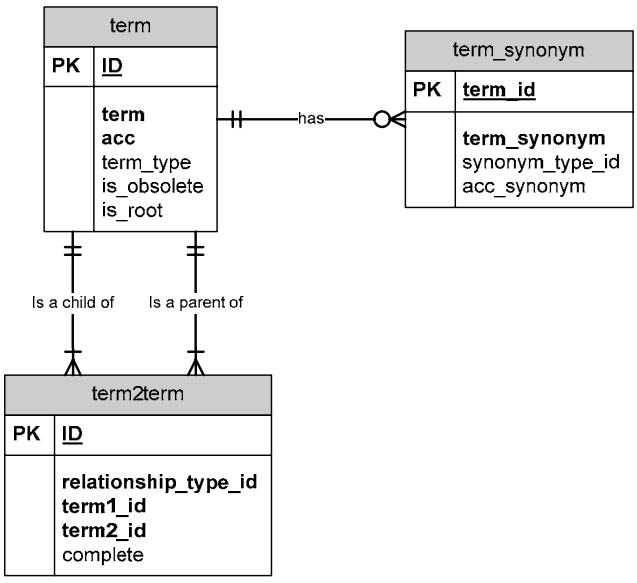

Figure 7: Database schema of utilized subset of GO.

This ontology is used in the current version of the search engine, though only a few GO tables (shown in Figure 7) are utilized. The *term* table lists the concepts embedded in the ontology, with varying degrees of term synonyms available in the *term_synonym* table. Relationships between terms are defined in the *term2term* table.

2.2.2. Plant Ontology (PO)

The PO is an umbrella ontology developed by the Plant Ontology Consortium. The goal of the project is to "develop, curate, and share controlled vocabularies…that facilitate and accommodate functional annotation efforts in plant databases and by plant science researchers" [11]. Currently the PO comprises two sub-ontologies: the plant structure ontology, containing terms related to anatomical and morphological structures, and the growth and developmental stages ontology. The terms are organized and related hierarchically using the same database structure as GO, with the same set of tables currently utilized (see Figure 7).

2.2.3. Trait Ontology (TO)

The Trait Ontology is a controlled vocabulary to "describe each trait as a distinguishable feature, characteristic, quality, or phenotypic feature of a developing or mature individual" [13]. Example terms include "leaf color", "cold tolerance", and "starch yield". While initially developed for the rice community by Gramene, this ontology is being extended to include more general plant traits and phenotypes. As this ontology contains terms related to phenotypic features, it may be added to the search engine's utilized ontologies in the future.

2.2.4. Phenotype and Trait Ontology (PATO)

The Phenotype and Trait Ontology is an ontology of phenotypic qualities [20], consisting of values that various phenotypes and traits may assume. PATO is designed to be used in conjunction with other ontologies by pairing a "quality-bearing entity" with a phenotypic quality. As PATO was originally designed primarily for animal model organisms, its use in the plant realm has been limited. Its focus on phenotypic qualities also makes it an excellent candidate for future inclusion into the developed search engine.

2.3. Utilization of Ontologies in Information Retrieval

From the inception of information retrieval, various techniques to improve retrieval performance have been proposed. The emergence of ontologies, both domain dependent and independent, provided a new resource for improving search, and these ontologies have been utilized in various ways by the information retrieval community.

One such utilization of ontologies is for word sense disambiguation (WSD), which is the complex task of determining the correct meaning of a polysemous word from its context. Several groups have worked to develop algorithmic solutions to WSD that make use of the large lexical English database, WordNet [6]. Li et al developed an approach for WSD that utilized a set of heuristics based on the semantic network in WordNet. In the reported dataset, the correct sense of nouns was identified 72% of the time [17]. Banerjee and Pedersen improved the traditional Lesk algorithm to use a smaller context window and also make use of the semantic relationships in WordNet, instead of using the term glosses in standard dictionaries, for WSD [3]. Domain-specific ontologies have also

been utilized for WSD. Widdows et al [30] used the UMLS ontology for unsupervised WSD in English and German medical documents.

Ontologies have also been used for thematic summarization and concept mapping. The thematic summarization task tries to identify broad concepts or themes present in documents, while concept mapping involves matching documents to concepts in an ontology. These may be seen as two variations of the same problem, as the ontology concepts mapped to a document may also be used for thematic summarization. Harabagiu and Lacatusu used a large hand-crafted ontology of English words to summarize the themes present in multiple documents on the same topic [10]. Lee et al utilize a fuzzified domain ontology in order to choose the best sentences for summarizing Chinese news articles [16]. Aronson introduced the MetaMap algorithm, which can be used for discovering UMLS Metathesaursus concepts within biomedical texts [1].

Query expansion, a technique used to enhance a query by adding additional terms, can also be accomplished through ontologies. This technique has been applied since the early 1990s with mixed success [4]. Though many types of expansions are possible, the most basic being synonyms, parents, children, and siblings, little systemic research has been conducted regarding the identification of good expansion strategies [31]. Despite this, they have been utilized extensively in the literature. Navigli and Velardi utilize WordNet for WSD then query expansion using various strategies including synonyms, children, gloss words, and common nodes in the semantic network. Expansion using gloss words produced the most improvement in the results [19]. Fu et al. utilize two kinds of ontologies, a domain ontology and a geographical ontology, for improved retrieval of documents that are spatially relevant to the user's queries [7]. Wollersheim and Rahayu

use the UMLS ontology for query expansion using a variety of expansion strategies and techniques [31]. Of special interest, they quantify the effects of adding synonyms, parents, and children to the query as proportions of the precision attributed to each type of expansion (see Table II).

Table II: Improvement in precision by expanding the query using different kinds of relationships [31].

Relationship Type	Precision
Exact Synonym	12.20%
Narrower Synonym	6.50%
Broader Synonym	15.00%
Parent	10.50%
Child	7.30%
Siblings	9.50%

As illustrated, much research has been undertaken regarding the best ways to utilize the knowledge embedded inside ontologies, specifically with regards to query expansion, within information retrieval systems. State-of-the-art query expansion generally uses hierarchical relationships, typically parents and children [31], and this is the technique that will be employed in this thesis.

CHAPTER 3

DATA AND STORAGE

3.1. Data

The maize phenotype search mechanism will be responsible for searching text data from MaizeGDB, specifically phenotype image captions and related text sources. The document repository comes from the latest release of MaizeGDB, as of March 31, 2009. Though any text source in MaizeGDB that is related to the phenotype captions can, in theory, be included in the search engine, currently only three sources are being utilized. Figure 8 illustrates the linkage between these sources via the Entity-Relationship Diagram (ERD) for this subset of MaizeGDB. The main body of the repository is the set of 4103 mutant phenotype image captions (in the *WEB_IMAGE* table). These captions are also currently tied to two additional text sources in the database. As each mutant is linked, via the *VARIATION* table, to the genetic loci that have been shown to play some part in controlling the manifestation of the phenotype, a text source describing the genetic loci themselves (from the *MEMO* table) is included. Also, a text source describing the gene products (from the *GENE_PRODUCT* table) associated with the loci controlling the mutant phenotype can also be included in the search as well.

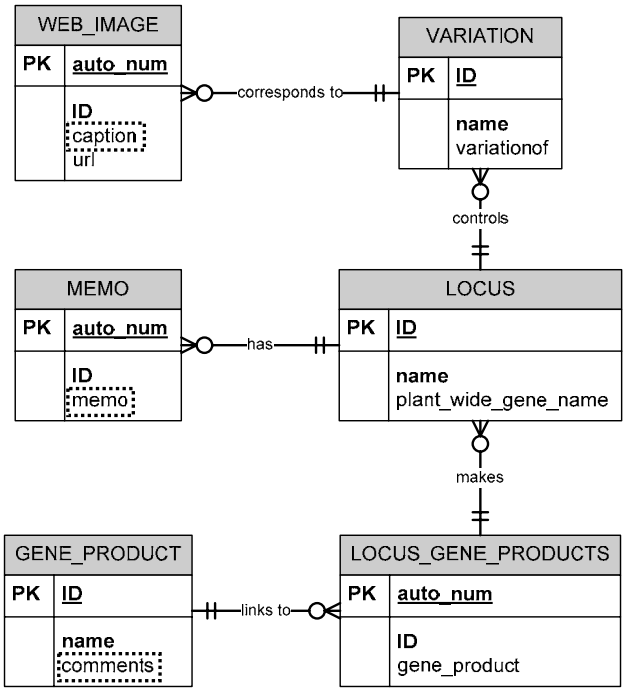

Figure 8: The subset of the MaizeGDB data model that is being utilized for the maize phenotype search engine. The text sources currently being searched are surrounded by dotted boxes.

The phenotype search mechanism will also make use of ontologies for query expansion. Any ontology whose domain may be covered by at least one of the searchable text sources can be considered for inclusion. Currently, the latest versions (as of March 31, 2009) of the PO and GO are incorporated.

3.2. Data Model

To facilitate faster retrieval, a custom database schema has been developed, which is used in conjunction with the MaizeGDB and ontology databases. Briefly, the database is responsible for storing information regarding the text sources the search engine supports, the ontologies that can be utilized by the retrieval engine, as well as intermediate

calculations associated with document representation, which will be discussed in Section 4.1. The database structure is shown below in Figure 9.

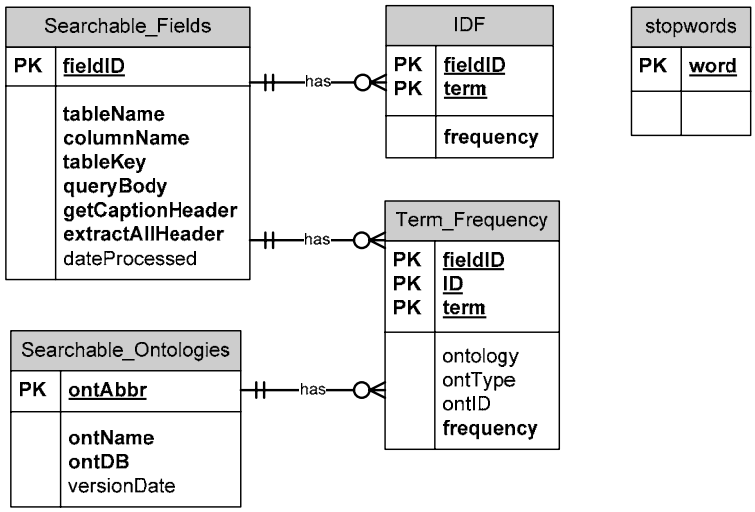

Figure 9: The database schema underlying the phenotype search engine.

The *Searchable Fields* table contains a row for every text source that can be searched. For this search engine, text sources correspond to text columns located in the database. All the information required to incorporate the source into the search engine is stored in this table including the primary key (PK) as well as some partial queries that tie this search engine to the other text sources. The two queries that are needed for each text source are:

1. *Retrieve all documents from a text source that are related to a phenotype image caption.*

   Before a search engine can began accepting queries and retrieving documents, the underlying corpus is preprocessed. This query pulls out all the relevant

documents from a text source so that they can be processed, as discussed in CHAPTER 5. This query is a concatenation of the *extractAllHeader* and *queryBody* columns. The date a text source was last processed is stored in the *dateProcessed* field.

2. *For a specific document from a text source, retrieve all associated phenotype image captions*

   During the retrieval process (discussed in CHAPTER 7), scored documents from each text source are linked to phenotype captions. This query provides that mapping and is formed by combining the *getCaptionHeader* and *queryBody* columns.

Similarly, the *Searchable Ontologies* table contains information regarding the ontologies that may be utilized during a search. As both the PO and GO have the same database structure, this table currently holds only the name of the database where the ontologies are located as well as some identifying information.

The *Term Frequency* and *IDF* tables store information related to document representation and term statistics. These values are calculated for each term in each text source during offline preprocessing of documents and facilitate speedy retrieval of documents during the online aspect of the search engine.

CHAPTER 4

RETRIEVAL MODEL

When building a search engine, there are many retrieval models that may be utilized. This chapter introduces the information retrieval model selected as well as the rationale for that choice. It also discusses why the multi-source retrieval problem deviates from the standard IR problem. Transformation of the multi-source problem to fit the vector space model as well as formulation of an appropriate similarity measure concludes the section.

4.1. Vector Space Model

In the vector space model [24], documents and queries are represented as vectors of index terms. An index term is a word "whose semantics help in remembering the document's main themes" [2]. Let C be a corpus, t be the number of index terms in the corpus, d_j be the j^{th} document from the corpus, q be a query, and $w_{j,i}$ be the term weight associated with index term i in document j. A document can then be represented as

$\overrightarrow{d_j} = (w_{j,1}, w_{j,2}, \ldots, w_{j,t})$, and a query as $\vec{q} = (w_{q,1}, w_{q,2}, \ldots, w_{q,t})$. Though many term weight definitions have been implemented, most retrieval engines nowadays rely on some variation of the standard weighting scheme below:

$$w_{j,i} = TF_{j,i} * IDF_j \quad (1)$$

where $TF_{j,i}$ is the term frequency component and IDF_j is the inverse document frequency component.

Term frequency (TF) is a measure of the importance of a term *within* a document. The normalized term frequency (see Equation 2) is traditionally used, as it guarantees $0 \leq TF_{j,i} \leq 1$. The rationale for including TF in the weight of a term is that the more times a term appears within a document, the more representative it is of the document's semantics and thus the higher weight it should receive.

$$TF_{j,i} = \frac{freq_{j,i}}{max_x\{freq_{j,x}\}} \quad (2)$$

where $freq_{j,i}$ is the number of times term i appears in document j.

The second component to the term weight is the inverse document frequency (IDF). This component determines the rarity of a term within the entire document corpus and is calculated using Equation 3. From the formula, it is observed that terms that appear in larger percentages of the document will be given a lower weight than those that appear in fewer documents. This is logical as the more documents a term appears in, the less discriminating is the term.

$$IDF_i = log_{10}\left(\frac{|C|}{|\{d_j \mid w_{j,i} > 0\}|}\right) \quad (3)$$

where the numerator is the number of documents in the corpus and the denominator represents the number of documents containing term i.

With these vector representations, the similarity of a document and query can be expressed in terms of the vectors. The standard way to define the similarity between the two is to determine the angle between the vectors. The smaller the angle, the higher the similarity is between the document and query. This rationale mathematically corresponds to the cosine of theta. Using this logic, the cosine similarity measure can be defined as:

$$sim(d_j, q) = \frac{d_j \cdot q}{|d_j| \times |q|} \quad (4)$$

which in terms of index term weights can be written as:

$$sim(d_j, q) = \frac{\sum_{i=1}^{t} w_{j,i} * w_{q,i}}{\sqrt{\left(\sum_{i=1}^{t} w_{j,i}^2\right)} * \sqrt{\left(\sum_{i=1}^{t} w_{q,i}^2\right)}} \quad (5)$$

Document retrieval using the vector space model simplifies to calculating the similarity between the query and each document, with ranking of documents accomplished by sorting the documents according to similarity score.

4.2. Selection of the Vector Space Model

Though many other retrieval methods have been developed in the IR community, the vector space model was chosen for this application for several reasons.

A number of models have been developed based on set theory. These include the Boolean model, the extended Boolean model [23], and several fuzzy set models. The major reason to select the vector space model over all set theory models is the simple desire to have free text queries, not Boolean expression queries. The fact that the vector

space model also provides non-binary term weights and partial query matching gives it an added benefit over the classic Boolean model.

Probabilistic models have also been proposed including the classic probabilistic model [22], Bayesian networks [21], and inference networks [29]. In these models, documents are ranked not in terms of their similarity to the query, but rather in terms of their probability of being relevant to the query, a subtle but important distinction. As there is controversy over whether the probabilistic model outperforms the vector space model [2], the vector space model was chosen for its speed and simplicity.

Several algebraic alternatives to the vector space model are available as well including the generalized vector space model (GVS) [32] and latent semantic indexing (LSI) [8]. Implicit in the vector space model is the assumption that terms are mutually independent; as this assumption may not always be true, the GVS model tries to incorporate term dependencies into the retrieval. However, because term-term correlations have not been shown to have a significant impact on retrieval performance and because of the increased complexity and computational expense of GSV, the standard vector space model is preferred.

LSI is a model that represents queries and documents in terms of prominent corpus concepts, which correspond to the largest eigenvectors from singular value decomposition. LSI concepts present considerable difficulties for employing query expansion techniques because the concepts from eigenvectors typically do not correspond to single terms, but rather are linear combinations of index terms. Thus, since we want to make use of domain-specific ontology knowledge in our search engine, the vector space model is the better choice.

As a final point, the vector space model is justified for this search engine as it is quick and simple and there is near consensus that “in general, the vector model is either superior to or almost as good as the known alternatives” [2].

4.3. The Rationale for Document Case Retrieval

Unfortunately, the vector space model cannot, in its traditional formulation, be directly applied to this retrieval engine. This is because while most search engines retrieve and rank single documents, this multi-source search mechanism has to be able to retrieve and rank *groups* of documents. What are these groups of documents? Consider the set of sources depicted in the top half of Figure 10. Each phenotype image caption is linked to zero or more documents from each text source. Thus, when a phenotype search is submitted, the retrieval engine will need to consider sets of related documents (the bottom half of Figure 10) consisting of a phenotype image caption as well as related loci and gene product descriptions. Each of these sets of documents, referred to as a *document case*, must be retrieved and ranked. (Note that the formal definition of a document case is given in Section 4.4.) It should be noted that document cases may vary in the number of component documents and also may not be disjoint in terms of their

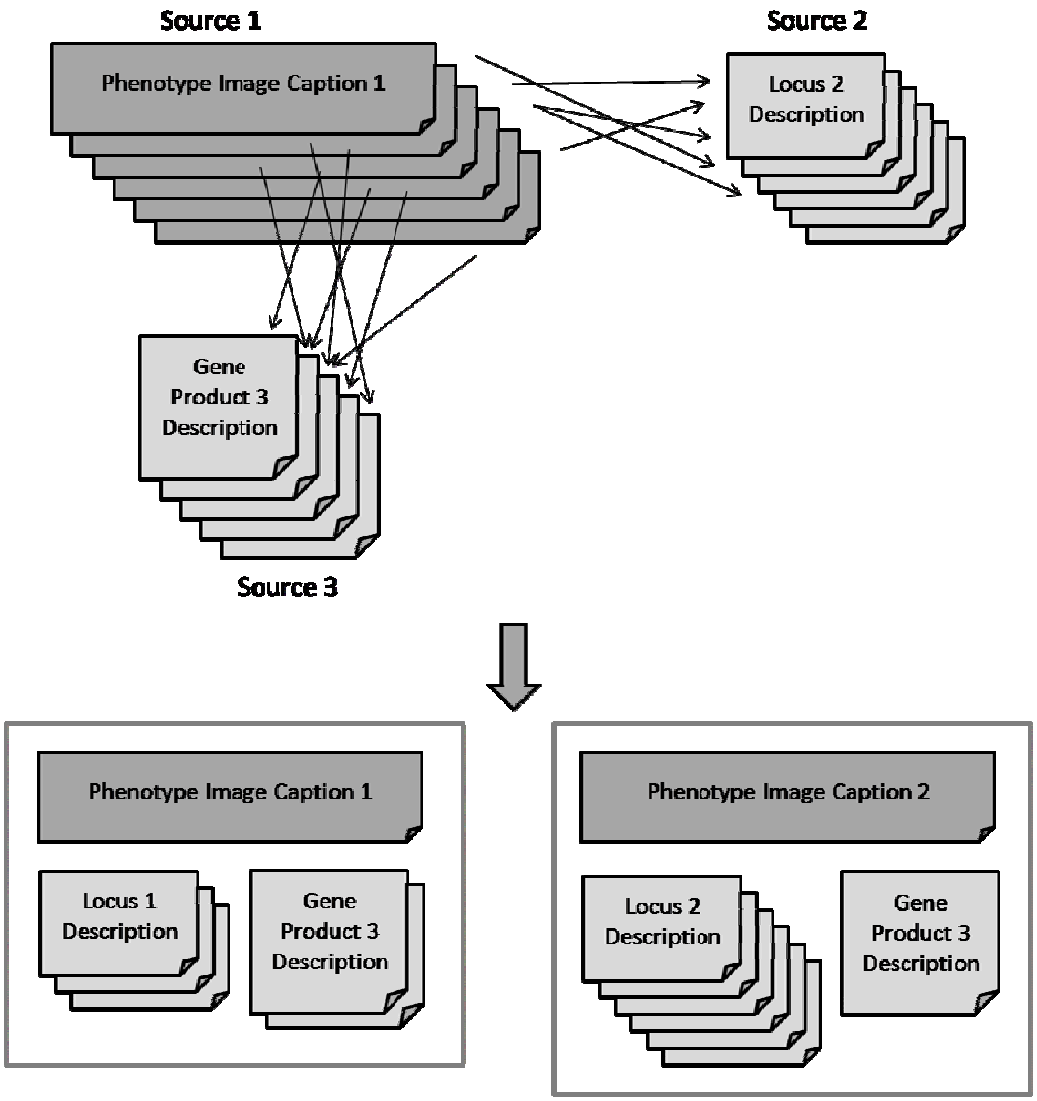

Figure 10: Depiction of various text sources along with the relationships between sources (top) as well as the document case view (bottom) where related documents are grouped.

component documents. (e.g. Phenotype Image Captions #1 and #2 in Figure 10 both link to Gene Product #3 Description).

The simplest solution to this problem is obviously just to merge the text from all the documents making up the document case into one large document. While this solution does have the advantage of allowing the use of the standard vector space retrieval model,

it does not, however, reflect the data making up these document cases very well. This is because the phenotype captions, loci descriptions, and gene product descriptions are describing very different aspects of the maize domain. Thus, there is variability in the distribution of specific terms, both in terms of usage and rarity, between text sources. This can be illustrated concretely by examining the variability of ontology terms across these sources. Because the GO contains terms regarding genes and gene products, we would expect these terms to appear more often in the loci and gene product descriptions than in the phenotype captions. Likewise, since the PO contains plant anatomy and morphology terms, we expect these to appear most often in the phenotype descriptions and, to a lesser extent, in the loci descriptions. By examining Table III, this is precisely the situation we find. The trend shown is that PO terms appear less frequently as we move from phenotype captions to locus descriptions to gene product descriptions, with precisely the opposite pattern present with GO terms.

Table III: Ontology term breakdown per text source.

Average # of Terms per Document	*GO*	*Text Source*	*PO*	*Average # of Terms per Document*
0.30	↓	Phenotype Image Caption	↑	1.32
0.58		Locus Description		0.94
1.03		Gene Product Description		0.31

As the terms are distributed differently across text sources, it follows that some terms may be more discriminate in some sources versus others. Consider the term “starch”, which appears in only 2 of 4103 phenotype captions, yielding a term IDF of 3.312 in captions, versus in 37 of 1539 locus descriptions, giving a term IDF of 1.619 in this field.

Though “starch” is fairly discriminate in both fields, its IDF values make it exceedingly more discriminate in the phenotype caption field.

For these reasons, merging all the components of a document case into a single document does not make the most logical sense. The proposed alternative, and the main contribution of this thesis, is the transformation of this multi-source retrieval problem so that the vector space model can be applied with the formulation of a novel similarity measure between document cases and the query.

4.4. Vector Space Formulation for Document Cases

Let K be the set of text sources searched for a particular query. Let D^k be the set of all documents within source k with D^1 denoting the base document set, that is, the seed documents that are used to build document cases and also the set that will be fully displayed in the ranked results. The other document sets, $D^k, k > 1$, are secondary documents that are related to one or more base documents. In this search engine, D^1 corresponds to the phenotype image captions, with D^2 and D^3 being the genetic locus and gene product descriptions, respectively. The secondary text sources may partially or completely determine the ordering of the D^1 documents in the final ranked results. The secondary text sources will only contribute to the ranking when D^1 is included in the searched sources, i.e. $(1 \in K)$, and they will completely control the ordering when D^1 is not included in the searched sources, i.e. $(1 \notin K)$. Let d_j^k denote document j within document set D^k and q be a query represented by the term vector $\vec{v}$.

In order to formally define a document case, the relationships of documents between text sources must be defined. Let f be the mapping (see Equation 6) that, given a

document j from D^1 and a text source k, returns the list of documents in source k that are related to d_j^1. In this search engine, these relationships are accomplished through a series of database table joins.

$$f(j, k) = \{d_l^k \mid d_l^k \sim d_j^1, 0 \leq l < |D^k|\} \quad (6)$$

In the current dataset, f may return up to nine genetic locus documents or up to four gene product documents for a single phenotype description. Note, again, that a document from a secondary document set may link to more than one base document, as one gene product, for example, may be associated with several phenotype descriptions.

Let F be defined as the set of all documents mapped to d_j^1, a base text document, for a particular K. $F(j, K)$ defines the document case for d_j^1 for given set of text sources K.

$$F(j, K) = \{\bigcup_{k \in K} f(j, k)\} \quad (7)$$

It is noted that two document cases may consist of different numbers of documents and also that the number of documents comprising a document case is not constant and depends on *K*.

With the notion of document cases formalized, considerations for determining the similarity between a document case and the query can be addressed. Clearly, to use any variant of the vector space model, each document case must be represented as a vector of constituent term weights.

One may consider forming a document case vector by concatenating document vectors from each document in $F(j, K)$. This approach suffers because each base document can be linked to an arbitrary number of documents from each secondary source, which makes each document case vector a different length.

More viable approaches treat the document case as a collection of $|K|$ documents, one for each of the text sources being searched. These approaches apply some function $g(.)$ to the documents in $f(j,k)$ in order to obtain a single vector for each document case from each source. Two classes of $g(.)$ functions (merge and selection) are described below:

- *Merge*: With this approach, all the documents in $f(j,k)$ are merged together. Conventional weighting schemes can be applied to the conglomerate document vector. This option seems most appropriate when the documents in $f(j,k)$ contain text on the same theme, as the amalgamation of the set is likely to better represent the set than any one document in the set. This is also useful when the user wants to match more globally to all the documents in $f(j,k)$.

- *Selection*: Instead of merging the documents in $f(j,k)$ together, one may select a particular document from the set to represent $f(j,k)$. The selection operator can be used when the documents in $f(j,k)$ are more disjoint in nature. From the user perspective, if the user was interested in finding only the best match from $f(j,k)$, the document with the maximum similarity to the query could be selected. One could consider other schemes like the most average document or just selecting one at random.

The preferred function class is highly dependent on the text source as well as users' information needs. In the case of this maize phenotype search engine, both classes of functions are applicable. The locus source is a good candidate for the *merge* option. This

is because with the current dataset, each phenotype is linked to only a single locus. The several locus descriptions linked to a phenotype caption thus all describe the same locus. On the other hand, the gene product source is better suited for the *selection* option, as each phenotype caption may be linked to several gene product descriptions, each describing a different gene product. We do not expect the descriptions of two different gene products to be similar because they are describing very different entities. The selection function used in this search engine is defined in Equation 8. Briefly, it selects the document from $f(j,k)$ that has the highest similarity to the query.

$$sel\big(f(j,k)\big) = \left\{d_i^k | i = max_l\{sim(d_l^k, q)\}\right\} \quad (8)$$

From a user's perspective, if an image caption is linked to several gene product descriptions, only one of which is highly similar to the query, the user would expect that document case to be ranked highly because of the one good match, not ranked lowly because several other associated gene products were not very similar to the query, and this is precisely the expected behavior using this selection function.

In addition to choosing the function $g(.)$ for each text source, a decision must be made on how to handle the $|K|$ document vectors comprising the document case. Again, two options are provided.

- *Weighted average*: The first method to combine the $|K|$ document vectors is to perform a weighted average, per Equation 9.

$$sim(F(j,K),q) = \sum_{k\in K} w_k sim(g(f(j,k)),q) \quad (9)$$

With this approach, each text source can be weighted to reflect its importance relative to the other sources. Also, the traditional cosine similarity measure can be applied between the query and the formed vector (by merge or selection) for each text source.

- *Document case vector:* Alternatively, one could form a document case vector by concatenating the $|K|$ component document vectors (see Equation 10).

$$\vec{F}(j,K) = (\vec{d}^{k_1}_{g(f(j,k_1))}, \ldots, \vec{d}^{k_n}_{g(f(j,k_n))}) \qquad (10)$$

The similarity between the query and a document case can then be expressed as a reformulation of the standard cosine similarity measure (Equation 5) in terms of text sources and component documents. The similarity formula is provided for both the merge and selection approaches

For the merge case, the similarity measure is given by Equation 11. In keeping with the idea behind merge, each component document from each source is involved in the calculations of the cross product (numerator) and the document case normalization factor (left term in the denominator).

$$sim_{merge}(F(j,K),q) = \frac{\sum_{k\in K}\left(\sum_{i=1}^{t}\left(\left(\sum_{d=f(j,k)} w^k_{d,i}\right)*w^k_{q,i}\right)\right)}{\sqrt{\sum_{k\in K}\left(\sum_{i=1}^{t}\left(\sum_{d=f(j,k)} w^k_{d,i}\right)^2\right)}*\sqrt{\sum_{k\in K}\left(\sum_{i=1}^{t} {w^k_{q,i}}^2\right)}} \qquad (11)$$

where the weight of term i in document d of text source k, $w^k_{d,i}$, is expressed by

$$w^k_{d,i} = tf^k_{d,i} * idf^k_i \qquad (12)$$

and the analogous query term weights are given by

$$w^k_{q,i} = tf^k_{q,i} * idf^k_i \qquad (13)$$

Using the properties of summation and the distributive law, the numerator can be rewritten for more efficient calculation by Equation 14. This version allows the calculation to proceed by looping over all the terms in each document in $f(j,k)$, rather than having to loop through each document in $f(j,k)$ for each term.

$$sim_{merge}(F(j,K),q) = \frac{\sum_{k\in K}\left(\sum_{d\in f(j,k)}\left(\sum_{i=1}^{t} w_{d,i}^{k} * w_{q,i}^{k}\right)\right)}{\sqrt{\sum_{k\in K}\left(\sum_{i=1}^{t}\left(\sum_{d\in f(j,k)} w_{d,i}^{k}\right)^{2}\right)} * \sqrt{\sum_{k\in K}\left(\sum_{i=1}^{t} {w_{q,i}^{k}}^{2}\right)}} \quad (14)$$

The similarity measure when using the selection function may be computed as in Equation 15.

$$sim_{selection}(F(j,K),q) = \frac{\sum_{k\in K}\left(\sum_{i=1}^{t}\left(w_{g(f(j,k)),i}^{k} * w_{q,i}^{k}\right)\right)}{\sqrt{\sum_{k\in K}\left(\sum_{i=1}^{t} {w_{g(f(j,k)),i}^{k}}^{2}\right)} * \sqrt{\sum_{k\in K}\left(\sum_{i=1}^{t} {w_{q,i}^{k}}^{2}\right)}} \quad (15)$$

Recall that with selection, only one document from each text source, chosen using the function $g(.)$, is used in the similarity calculation of the document case to the query. Document and query term weights continue to be defined as in Equations 12 and 13.

The document case vector option was selected over the weighted average approach because it better represents the cohesiveness of the documents in the document case. One positive side effect of this choice is that there is more weight given when a query term matching to documents from more than once source in a document case than with the weighted average option.

CHAPTER 5

DOCUMENT SET PROCESSING

With document case representation and the similarity measure defined, focus now shifts to the construction of the search engine, which can be partitioned into two main steps: (1) offline preprocessing of the entire document corpus and (2) online query processing and retrieval. Offline preprocessing is discussed in this chapter, with the online aspect covered in the following chapter. The process flow for each document in the corpus is shown in Figure 11. Each component of the process is discussed in more detail in its respective section.

To illustrate the preprocessing procedure, the following example of a simple phenotype caption is used: “a maize leaf with a purple leaf blade.”

5.1. Parsing

The first step in the process is the parser. In this step, all non-word characters are first replaced with spaces. This rids the documents of punctuation and other special characters that should not be relevant to the search. Then, the document is then split into terms.

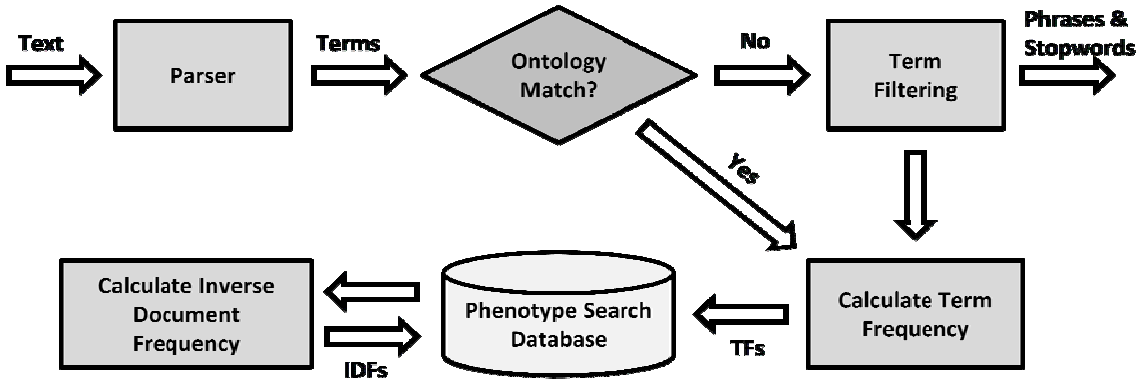

Figure 11: Preprocessing flow chart, from parsing through insertion into the database.

Traditionally, parsing for the vector space model splits a document into single words and uses those "terms" as vector elements; however, because this particular search engine needs to connect to ontological terms, which primarily consist of more than a single word (the maximum length of a term is 12 words in PO and 27 in GO), we must be more sophisticated with our parsing. To facilitate attachment to ontologies, documents are parsed into phrases, which are defined as a set of consecutive words in a document, using a sliding window approach.

A maximum phrase length is imposed on the parsing. This is an adjustable parameter that defaults to six, meaning phrases consisting of one to six consecutive words in length are generated from each document during parsing. The rationale for such a maximum is multi-factorial. First, longer (than six-word) phrases are not common in the ontologies; they account for only 7.1% and 12.9% of terms in the PO and GO, respectively. Further, neither documents nor queries are expected to contain terms that match long ontology terms. In fact in the current dataset, the longest matches are to ontology terms comprised of only four words. Finally, because of the future expected rarity of these long phrases

Table IV: Statistics on term sizes in PO and GO. The terms are partitioned by number of words. The coverage columns show the percent of terms that are covered by the search engine if terms up to that length are formed by parsing.

# of words	# of PO Terms	% PO Coverage	# of GO Terms	% GO Coverage
1	208	17.8%	487	1.7%
2	360	48.7%	3873	15.4%
3	269	71.8%	9742	49.9%
4	155	85.1%	4877	67.1%
5	60	90.2%	3530	79.6%
6	**31**	**92.9%**	**2104**	**87.1%**
7	28	95.3%	1235	91.4%
8	6	95.8%	769	94.1%
9	36	98.9%	555	96.1%
10	12	99.9%	349	97.3%
11	0	99.9%	250	98.2%
12	0	99.9%	173	98.8%
13	1	100.0%	110	99.2%
14	0	100.0%	74	99.5%
15	0	100.0%	55	99.7%
>15	0	100.0%	88	100.0%
TOTAL	**1166**		**28271**	

matching to ontology terms, it would be wasteful computationally to spend the time generating all possible phrases. To determine the default value, a balance between computational effort and ontology coverage was sought. Table IV shows the statistics of terms in the GO and PO with respect to the number of words per term. The default value, which parses phrases up to a length of six words, was chosen as this covers approximately 90% of terms in both ontologies (92.9% in PO and 87.1% in GO) and is not that computationally expensive, though admittedly is overkill for the current set whose longest match is only four words.

Using these parsing parameters, our example caption would produce the list of possible document terms shown in Table V

Table V: Generated phrases from the sample phenotype description.

Phrase Length (in words)			
1	**2**	**3**	**4**
a	a maize	a maize leaf	a maize leaf with
maize	maize leaf	maize leaf with	maize leaf with a
leaf	leaf with	leaf with a	leaf with a purple
with	with a	with a purple	with a purple leaf
purple	a purple	a purple leaf	a purple leaf blade
blade	purple leaf	purple leaf blade	
	leaf blade		
5			**6**
a maize leaf with a			a maize leaf with a purple
maize leaf with a purple			maize leaf with a purple leaf
leaf with a purple leaf			leaf with a purple leaf blade
with a purple leaf blade			

5.2. Matching to Ontologies

After the list of terms is generated by the parser, each is tested to determine if it corresponds to an ontology concept. A correspondence is recorded if and only if the parsed term exactly matches an ontology concept (using Query 1) or a concept synonym (using Query 2).

```
SELECT id
  FROM term
 WHERE is_obsolete = 0
   AND (name = '$term'
    OR name = '$term (sensu Zea)'
```

Query 1: SQL statement for matching to ontology terms

Both PO and GO maintain all concept references within the *term* table. Inspection of the WHERE clause shows exact matching of terms, including those that contain the maize-specific suffix "(sensu Zea)". This textual addition only occurs in the PO and can be used to confirm the contextualization of the matched term within the search engine's domain (maize). Note that terms that have been labeled obsolete are not retrieved. Obsolete terms, while still technically existing in the ontology, are no longer connected to other terms, and so they cannot be utilized to enhance the query by accessing embedded ontology knowledge.

Parsed terms may also correspond to ontology concept synonyms. These are found by querying the *term_synonym* table.

```
SELECT term_id
  FROM term_synonym
 WHERE term_synonym = '$term'
    OR term_synonym = '$term (sensu Zea)'
```

Query 2: SQL statement for linking terms to ontology synonyms.

One may wonder why correspondences are defined as exact matches and not partial matches. While it is possible to define a correspondence based on a partial match of an ontology concept or synonym, this greatly increases the risk of decreased contextualization of the query, as a parsed term may partially match many, many items in an ontology. For example, the term "leaf" partially matches 69 different PO concepts and 113 different PO synonyms. Matching to all these different ontology concepts will hinder the query, not improve it. However, if a longer phrase containing "leaf" matches to an ontology concept, e.g. "leaf blade", that match will also be recorded, as it too is an exact match.

For each identified correspondence, the term along with the ontology and ID of the concept to which it matched as well as the type of match (term or synonym) are recorded. For the example phenotype description "a purple leaf blade," the matched ontology concepts generated can be seen in Table VI.

Table VI: Matched terms from "a maize leaf with a purple leaf blade."

Term	*Ontology*	*ID*	*Type*
leaf blade	Plant	806	Synonym
leaf	Plant	686	Term

It should be noted that the search engine was designed to include embedded ontology knowledge without having to process the ontology information itself. This plug-and-play behavior of included ontologies is the reason why stopwords are not removed from the query until after ontology matching is conducted, as some ontology words contain stopwords.

5.3. Filtering Unmatched Terms

Terms that match an ontology concept are retained and sent to the TF module (see Section 5.4). Those that do not match, however, proceed through to filtering. If the unmatched term is a single word, it is retained so long as it does not correspond to a stopword, which is defined as "a word that does not carry meaning in natural language" [2]; otherwise, it is removed. The list of stopwords used for this search mechanism can be found in Table VII.

Table VII: List of stopwords used in search engine.

a	as	is	it	the	when
about	at	from	of	this	where
an	be	how	on	to	who
and	by	i	or	was	will
are	for	in	that	what	with

If the unmatched term is a phrase, i.e. consists of multiple words, then it is discarded. This is because the important words in the phrase are already counted as single words, and the remaining unimportant words (stopwords) do not contribute positively to the meaning of the document.

The final set of terms representing the sample description is shown in Table VIII. During this step, all generated phrases are deleted, except for “leaf blade”, which was shown in the previous section to match to a synonym in the PO. The single words “maize”, “purple” and “blade” are retained, while the stopwords “a” and “with” are removed.

Table VIII: Final set of terms representing the sample phenotype description.

Term	*Ontology*	*ID*	*Type*
leaf blade	Plant	806	Synonym
leaf	Plant	686	Term
maize	-	-	-
blade	-	-	-
purple	-	-	-

5.4. Calculating Term Frequency

In the final steps of preprocessing, TFs are calculated for each matched ontology concept and important single word within each document. Typically, normalized TFs are

calculated during offline processing and stored for later use in the querying process; however, due to the allowance of phrases as part of the document vector, normalized TFs cannot be computed in the preprocessing stage, and another deviation from the standard vector space model is required.

Why is this? Let $freq_{j,x}$ be the raw frequency of term x in document j. Consider the raw frequencies of terms in the sample description, shown in Table IX. If the normalized TFs were calculated for each term, as in Equation 2, "leaf" would receive a normalized TF of 1 while all other term TFs would be 0.5. This calculation, however, double counts the "leaf" in "leaf blade." The count of 2 for "leaf" is accurate *only if* the query does not contain "leaf blade." If it does contain the phrase, both the raw frequency of "leaf" and all the pre-calculated normalized TFs become incorrect. Because of the issue of double counting when there are matched phrases, normalized TFs can only be computed on-the-fly in the context of a specific query. As a result, the terms as well as their raw frequency counts in the document are written to the database (again, see Table IX for an example) and will be used later in the querying process. The specifics of how normalized TFs are calculated dynamically during the querying process are discussed later in the thesis, specifically in Section 7.1.

Table IX: TF information written to the *Term Frequency* table for "a maize leaf with a purple leaf blade."

FieldID	ID	Term	Ontology	Ont ID	Type	Frequency
1	99	leaf blade	PO	806	Synonym	1
1	99	purple	-	-	-	1
1	99	leaf	PO	686	Term	2
1	99	blade	-	-	-	1
1	99	maize	-	-	-	1

5.5. Calculating Inverse Document Frequency

After all the documents from each source have been processed and their TF values stored in the database, the IDF can be calculated and stored in the database with a single query (Query 3). It should be noted that the IDF is not calculated globally. But rather, because there is variability in the distribution of terms across text fields, as discussed in Section 4.3, and because the flexibility of the system allows the user to choose which individual text sources will be included or excluded from the search, IDFs are calculated within a text source. Thus, for the current arrangement of the search engine, IDFs are calculated separately for terms from the *WEB_IMAGE.caption* source, the *MEMO.memo* source, and the *GENE_PRODUCT.comments* source. This helps to maintain term discrimination within specific text sources.

```
INSERT INTO IDF (fieldID, term, frequency
  SELECT T.fieldID, term, LOG10(C.Num/COUNT(ID))
    FROM Term_Frequency T,
        (SELECT fieldID, COUNT(DISTINCT ID) AS Num
           FROM Term_Frequency GROUP BY 1) C
   WHERE T.fieldID = C.fieldID
GROUP BY 1,2;
```

Query 3: INSERT statement to compute and store IDF values.

The nested query determines the number of documents within each searchable text source. The outer query counts the number of documents within each searchable source that contain each term. The fourth column selected calculates the IDF using the standard formula (Equation 3).

Unlike TFs, IDFs can be calculated entirely during preprocessing. This is because with the IDF calculations, we are concerned with whether a term is present in a document, not how many times it appears.

The calculation of IDFs concludes the preprocessing portion of the retrieval engine. The stored term statistics for each document will be utilized for retrieval with each query submitted to the system. Before retrieval can occur, though, the query must be processed.

CHAPTER 6

QUERY PROCESSING

After the document set has been preprocessed, with all the documents parsed, generated terms matched to the ontologies, and raw TFs and IDFs calculated, one must then consider how a submitted query is to be processed and how retrieval and ranking of relevant documents is to be carried out. The steps from query submission to outputting the ranked results from the search engine are displayed in Figure 12. The initial query processing steps look grossly the same as document processing, though there are differences and these will be discussed in each step's respective subsection.

A similar phenotype description, "a purple leaf blade," will be used again to illustrate the subtle, but important, differences between document and query processing.

6.1. Parsing

Query parsing is identical to document parsing (Section 5.1), in which all non-word characters are replaced by spaces, and phrases consisting of one to six consecutive words in the query are generated.

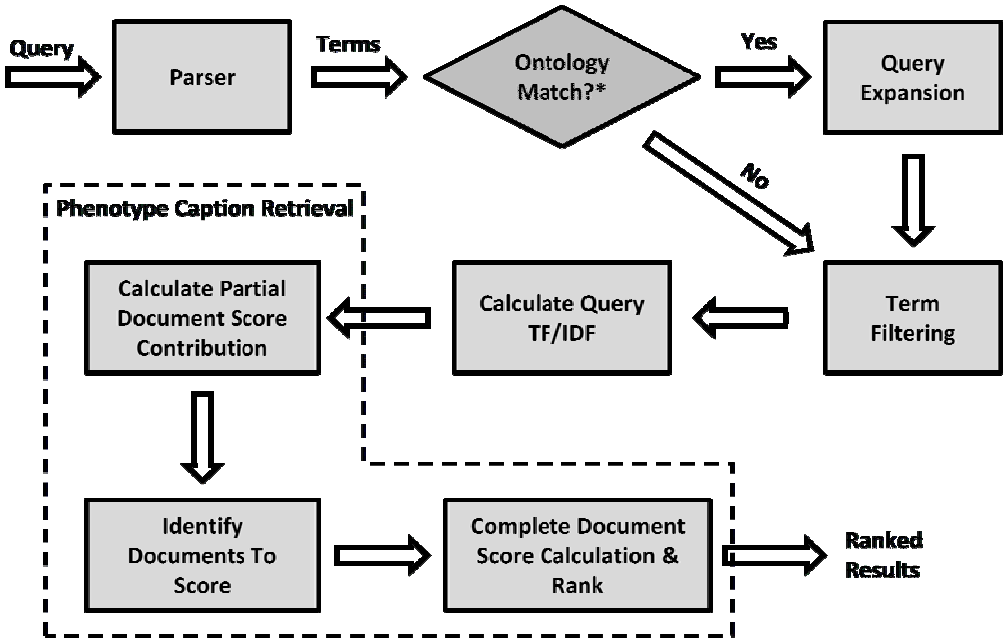

Figure 12: Flow chart for search engine retrieval, from query submission through output of ranked results.

6.2. Matching to Ontologies

The parsed terms are then matched to the available ontologies. This is performed in precisely the same manner as discussed in Section 5.2, with one very important exception. With document processing, all generated phrases that matched to the ontology were recorded, including those concepts that were subphrases of other matched ontology concepts, e.g. recall that both "leaf blade" and "leaf" were matched to the PO. This was required to ensure document matching for both queries containing the full phrase as well as those containing only the subphrase concept. Had the sample document only been matched to "leaf blade" and not "leaf," the document would not have been picked up for queries only containing "leaf." When processing queries, however, we are no longer interested in all possible ontologies matches. If a user submits a query with "leaf blade," it can be reasoned that the user is looking for captions containing this particular part of

the leaf and not just the concept "leaf" itself. While it is true that consecutive terms in a query could match to an ontology term not intended by the user, the occurrences should be rare considering the primary audience is domain experts. With this in mind, once a term i has been matched to ontology j, all subphrases of term i are eliminated from ontology match consideration within ontology j. These subphrases may, however, still match to other ontologies. Using this modified matching scheme, Table X shows the matched terms for the sample query. Note that even though "leaf" is a PO concept, it is not shown as a match as it is a subphrase of the PO concept "leaf blade."

Table X: Matched terms for the query: "a purple leaf blade."

Term	*Ontology*	*ID*	*Type*
leaf blade	Plant	806	Synonym

6.3. Expanding the Query

The terms that were identified as matching to ontology concepts can then be used as ontology attachment points, where the vocabulary and knowledge embedded into the ontology can be used to enhance, and hopefully improve, the query. For each ontology chosen for expansion, the user can determine what kinds of data should be used to enhance the queries. The choices include concept synonyms, which may be exact, narrow, or broad synonyms, as well as concept parents and children, which are more general and specific than the original term, respectively.

Using the ontology IDs, denoted $ID, stored with each matched term, term synonyms can be added by querying the *term_synonym* table (see Query 4), and parent and children terms can be identified by querying the *term2term* table. Query 5 corresponds to the

query that retrieves parent terms; an analogous query, formed by swapping *term1_id* and *term2_id*, can be used for getting children terms.

```
SELECT term_synonym
  FROM term_synonym
 WHERE term_id = $ID;
```

Query 4: SELECT statement to find synonyms using an ontology ID

```
SELECT T.name, T.id
  FROM term T, term2term M
 WHERE T.ID = M.term1_id AND T2.term2_id = $ID;
```

Query 5: SELECT statement to find parents using an ontology ID.

Note that the ontology IDs associated with parents and children are also retrieved in Query 5. This does not happen in Query 4, as term synonyms do not have dedicated ontology IDs;

Supposing the user elected to expand with synonyms and children (not parent terms) from all ontologies, the lone matched concept from the query, "leaf blade," would expand initially to the following:

Table XI: Possible expansion terms for "leaf blade."

Term	*Ontology*	*ID*	*Type*
leaf lamina epidermis	Plant	60	Child
leaf lamina vascular system	Plant	61	Child
leaf lamina base	Plant	631	Child
central zone of the leaf lamina	Plant	632	Child
leaf margin	Plant	865	Child
leaf apex	Plant	873	Child
leaf vein	Plant	874	Child

Because there is the potential for the query to be expanded with several terms, many of which may not be helpful to the search, one additional filtering step is applied before ending the query expansion task. Expanded terms are deemed to be irrelevant to the search and are removed if they fail to appear in the document set. Since ontology IDs are recorded with matched terms, finding expanded terms that exist in the document set requires just a simple query (Query 6) to reduce the expanded terms to the finalized set (see Table XII). In Query 6, *$idList* is a comma separated list of the ontology IDs, and *$ont* specifies the ontology to which the IDs belong.

```
SELECT DISTINCT term
  FROM Term_Frequency
 WHERE ontology = $ont AND ontID IN ($idList);
```

Query 6: SQL statement for reducing potential query expansion set to only those that appear in the corpus.

Table XII: Final expansion terms for "leaf blade."

Term	*Ontology*	*ID*	*Type*
leaf margin	Plant	865	Child
leaf apex	Plant	873	Child
leaf vein	Plant	874	Child

6.4. Filtering Unmatched Terms

Term filtering for queries is also similar to its corresponding step in document processing, again with one important exception. Similar to document processing, nonmatching phrases and stopwords are removed from the list of generated query terms, resulting in a query consisting of matched ontology concepts and *all* single words. Because the set of single words may correspond to subphrases of matched ontology

concepts, in query term filtering, single words are also removed if they match a word in a matched ontology phrase. In the case of the sample phenotype query, though "leaf" is not matched to an ontology concept (because it is a subphrase of "leaf blade"), it should also not be retained as an unmatched term in the query, which would happen without this extra processing step for queries. Table XIII shows the terms that represent the query after both ontology matching and term filtering.

Table XIII: Final set of terms representing the sample phenotype query.

Term	*Ontology*	*ID*	*Type*
leaf blade	Plant	806	Synonym
purple	-	-	-
leaf margin	Plant	865	Child
leaf apex	Plant	873	Child
leaf vein	Plant	874	Child

6.5. Calculating Query TF and IDF

Before a query can be used to retrieve documents, it first must be transformed into a vector. To do this, each term i, both those that matched to ontology concepts and those that did not, becomes an element in the query vector, represented by $w_{q,i}$, with term weighting calculated as follows:

$$w_{j,i} = tf_{q,i} * idf_i = \left(0.5 + \frac{0.5\, freq_{q,i}}{max_l \{ freq_{q,l} \}}\right) * log \frac{N}{n_i} \qquad (16)$$

From Equation 16, it can be observed that the query TF component is defined differently than it is for processing the document set. This variation ensures that the minimal importance of a term within the query is at least 0.5. One can also see that the

query IDF corresponds to the rarity of the term in the document set, which is the same definition as before. Since this has already been calculated for all terms in the document set, finding a query term IDF is simply a matter of querying the *IDF* table. It is possible, however, that a query term does not match any documents in the corpus, in which case the IDF query will not return any rows. Since the term does not match any documents, it will not be useful for retrieving documents; however, the fact that it is a query term and that no document matches it should still be factored into the similarity score. To account for this, terms that do not have any matches with D^k are given an IDF equal to the average IDF of all terms present in D^k.

Using the term weights as described, the query can now be represented as a vector and used to retrieve the most relevant document cases.

CHAPTER 7

DOCUMENT CASE RETRIEVAL

After a query has been parsed, expanded, and converted into a vector, the query vector is then used to retrieve the most relevant document cases from the database (see dotted box in Figure 12). To do this, similarity scores will be determined between the query and every document case by calculating the cosine of the angle between the query vector and document case vector using the modified similarity measure given by Equation 15. After the query has been compared with all document cases, the similarity scores are sorted from highest to lowest similarity and are returned as the output to the user.

The implementation details of retrieving and ranking those document cases that are most relevant to a submitted query are described in detail in this section. The following algorithm (Pseudocode 1) is used for the search retrieval and ranking, by efficiently calculating and sorting the similarity scores between the query and each document case. The code is partitioned into three major phases, separated by blank lines, with each discussed in detail in its respective subsection.

Pseudocode 1: Algorithm for document case retrieval and ranking.

```
01  FOR EACH text source k to be searched
02    FOR EACH query term q_i^k
03        Compute query term weighting
04        Progressively compute query normalization factor
05        FOR EACH document from k containing q_i^k
06            Compute partial scores between the document and query
07        END FOR
08    END FOR
09  END FOR

10  FOR EACH text source k to be searched
11    FOR EACH document from k with partial document score
12        Retrieve associated base text documents
13    END FOR
14  END FOR

15  FOR EACH retrieved base text document p
16    FOR EACH text source k to be searched
17        Retrieve documents linked to p
18        FOR EACH associated document d
19            IF d has not been seen THEN compute partial document score for d
20        END FOR
21        Determine document d̄ with highest similarity to query
22        Include the partial document score of d̄ to document case
23    END FOR
24  END FOR
25  Normalize document case scores
26  Sort document case scores in descending order
```

7.1. Computing Partial Document Scores (Lines 1-9)

The initial retrieval step is very similar to traditional document retrieval. The terms in the query are used to search for relevant documents, though in this case that meaning search in each of the text sources in K. Within each text sources, all the documents matching a query term are retrieved using the query below.

```
SELECT ID
  FROM Term_Frequency
 WHERE fieldID = $k AND term = '$query_term[$i]';
```

Query 7: SQL statement to find documents that match a specific query term.

For each of the documents matching a query term, partial document scores are calculated. What are these scores? By studying Equations 14 and 15, one can observe that there are three major components to either form of the similarity calculation, as shown below in Equation 17: the query normalization factor (φ), the contribution of one text source to the dot product between the document case and query (ξ), and the contribution of one text source to the document case normalization factor (η). The equations for each of these components, given the selection similarity equation, are given in Equations 18, 19, and 20.

$$sim(F(j,K),q) = \frac{\sum_{k\in K}(\xi)}{\sqrt{\sum_{k\in K}(\eta)} * \varphi} \tag{17}$$

where

$$\varphi = \sqrt{\sum_{k\in K}\left(\sum_{i=1}^{t} {w_{q,i}^{k}}^{2}\right)} \tag{18}$$

$$\xi = \sum_{i=1}^{t}\left(w_{g(f(j,k)),i}^{k} * w_{q,i}^{k}\right) \tag{19}$$

$$\eta = \sum_{i=1}^{t} {w_{g(f(j,k)),i}^{k}}^{2} \tag{20}$$

Consider a document from a text source k, d_i^k. If the merge similarity measure is used, this document *will* contribute to the document case of all d_i^1 for which $d_i^k \in F(i,K)$. If the selection similarity measure is used, this document *may* contribute to the same set of document cases. However, because at this early stage in the retrieval process

it is not known which document will be chosen by the selection function for each text source, it is wise to calculate and store the contribution potentially made by each document. Thus, regardless of which of the two similarity measures is used, calculating a document's contribution during this pass is justified as it facilitates the calculation of full document case similarity scores in the last phase of retrieval.

Thus for each document encountered in this pass, η_j^k and ξ_j^k (the η and ξ components for document j from text source k, respectively) can be computed and stored. The computation of both these quantities, however, requires normalized TFs, which as discussed earlier, were not calculated offline because of the issue with double counting terms. During this pass, however, because the specific query is known, normalized TFs can be calculated from the stored raw frequency counts. Depending on the makeup of the query, there are two different cases for computing normalized TFs.

The simplest case occurs when the query contains no phrases, in which case the normalized term frequencies for each document can be calculated from the stored raw frequency counts with Query 8.

```
  SELECT term, frequency
    FROM Term_Frequency
   WHERE fieldID = $k AND ID = $ID
     AND term NOT LIKE '% %'
ORDER BY 2 DESC
```

Query 8: SELECT statement to get all single-word TFs for a specific document.

This is accomplished by ignoring all matched phrases in the document and only considering the single words terms. There are two reasons why all phrases in retrieved

documents can be ignored. First, since the query contains no phrases, all query weights for phrases are zero. Thus, in the dot product portion of the similarity score (numerator), where the query and document term weights are multiplied together, any document weight for a phrase will be nullified by the zero weight from the query vector. Second, though one may argue that η would still need to account for the phrase to properly normalize the document vector, it is noted that the important single words making up the phrase are still present in the document vector and thus will still be used for normalization. Thus, when queries contain no phrases, phrases in retrieved documents can be safely ignored. We are also assured that all the single word frequency counts are correct, since double counting only occurs when phrases are involved.

To complete the calculation of TFs, each raw frequency should be divided by the maximum raw frequency count in each document. Fortunately, the rows are returned in decreasing order of frequency, which means the term with the maximum raw frequency for the document will be the first row returned. This makes normalized TF calculations very simple (Equation 2).

The second case occurs when the query vector does contain phrase terms. The calculation is then slightly more complicated, as the double counting must be removed before calculating the normalized TFs. The process is described in pseudocode below and begins with a query similar to Query 8, except *all* terms from the document are retrieved, not just single words.

Because terms may be counted both as a single word and as part of phrases, each single word that is part of a matching phrase must have its raw frequency count adjusted, by subtracting out the count of the phrase from the count of the word. In this way, the

Pseudocode 2: Eliminating double counting prior to normalized TF calculations.

```
01  Retrieve ALL term and raw frequencies for the document.
02  FOR EACH query phrase
03      IF (phrase appears in document) THEN
04          FOR EACH word comprising the phrase
05              Decrease the word's raw count by the phrase's raw count
06              IF (word's raw count <= 0) THEN remove word from consideration
07          END FOR
08          Resort remaining raw frequencies in descending order
09      ELSE
10          Remove phrase from consideration
11      END IF
12  END FOR
```

appropriate terms are selected for the document and the raw frequency counts are correct. If a document phrase does not match to the query or if a word's frequency drops to zero, then it is not used in the normalized TF calculations. After making these changes to the set of terms and their raw counts, the normalized term frequencies can be computed directly as per Equation 2.

To illustrate the process, consider the query "leaf blade" on our sample phenotype description "a maize leaf with a purple leaf blade." The raw frequency counts are shown below in Table XIV. Since "leaf blade" is a query phrase, its frequency (1) is subtracted from the frequency of each of its subphrases, "leaf" and "blade", yielding final frequencies for those terms of 1 and 0, respectively. Since the frequency of "blade" dropped to zero, that term should not be considered as part of the document's vector for this query, as it will have no effect on either the document normalization factor or the dot product component of the similarity calculation.

With normalized TFs calculated, the η_d^k term for a document d can be calculated by summing up ${w_{d,i}^k}^2$ for all the document terms, and ξ_d^k from this document can be found

Table XIV: Before and after raw frequency adjustment for query "leaf blade" to eliminate term double counting.

BEFORE	
Term	**Frequency**
leaf blade	1
purple	1
leaf	2
blade	1
maize	1

→

AFTER	
Term	**Frequency**
leaf blade	1
purple	1
leaf	1
blade	0
maize	1

by simply summing $w_{d,i}^{k} * w_{q,i}^{k}$ for all document terms.

Though query term weights were discussed in Section 6.5, it is during this phase of retrieval that the query term weights and the query vector normalization factor are calculated. Before using each query term to retrieve matching documents, its term weight is calculated using Equation 16. The query normalization factor can be accumulated by summing the squares of these term weights as they are calculated.

After this first phase of retrieval, the following quantities have been calculated and stored:

- Query normalization factor (φ).
- Normalization factors for each document j from text source k matching a query term (η_j^k).
- Dot products of each matching document j from text source k with the query (ξ_j^k).

7.2. Identifying Document Cases to Score (Lines 10-14)

From the partial document score calculations, all the documents from all searched text sources that match at least one query term have been found. The document cases that will be scored and ranked are precisely those that contain at least one of these documents. In order to identify these document cases, the inverse of the mapping function f (the original f is Equation 6) is invoked.

$$f^{-1}(l\,,k) = \{d_j^1 \mid d_l^k \sim d_j^1\,, 0 \leq j < |D^1|\} \qquad (21)$$

The implementation of this inverse mapping is a query on the database. For each document from each text field, the associated base text documents are retrieved. This set represents the document cases to be scored. Recall the discussion of queries stored in the *Searchable Fields* table for each text source (see Section 3.2). The second query discussed is precisely the query corresponding to this inverse mapping.

For each document scored in the previous section, this inverse mapping function is called. The union of all these function calls represents the set of document cases with non-zero similarity scores.

7.3. Computing of Document Case Similarity Scores (Lines 15-26)

Using the list of document cases generated in the previous section, the similarity between each document case and the query can now be computed. Similar to the partial document scoring in the first retrieval phase, the similarity of document case i to the query will be calculated progressively, with accumulators for ξ_i and η_i. Using the mapping function f for each text source, all the documents included in document case i from that source are then retrieved. Because this retrieval engine is utilizing the selection

similarity measure, the document with the highest similarity to the query is selected from each text source, and its pre-computed partial document scores (ξ and η) are added to ξ_i and η_i, respectively. If the document with the highest similarity has no pre-computed partial document scores, they are calculated on-the-fly, as discussed in Section 7.1, using the no-phrase case for the TF calculation.

With the calculations of ξ_i, η_i, and φ complete, the final similarity score for document case i can be simply computed by combining the three components as in Equation 17.

This process is repeated for each document case. Once all document case scores are done, the scored document cases are sorted in descending order, so that the best matching document case appears first in the list.

CHAPTER 8

SEARCH INTERFACE

The described document case retrieval engine for three text sources in MaizeGDB was implemented as part of the Visual Phenotype Database System (VPhenoDBS) project. The system is available to the public at the following URL:

http://PhenomicsWorld.org/QBTA.php

The interface for the search mechanism is shown in Figure 13. As the screenshot indicates, there is a text box for the user's free-text query. The standard "Search" and "Reset" buttons are provided to submit a query to the system and to clear the text box, respectively.

8.1. Customizing the Search

All the available options for the system can be found in the "Search Options" box underneath these controls. Each column represents a related set of options.

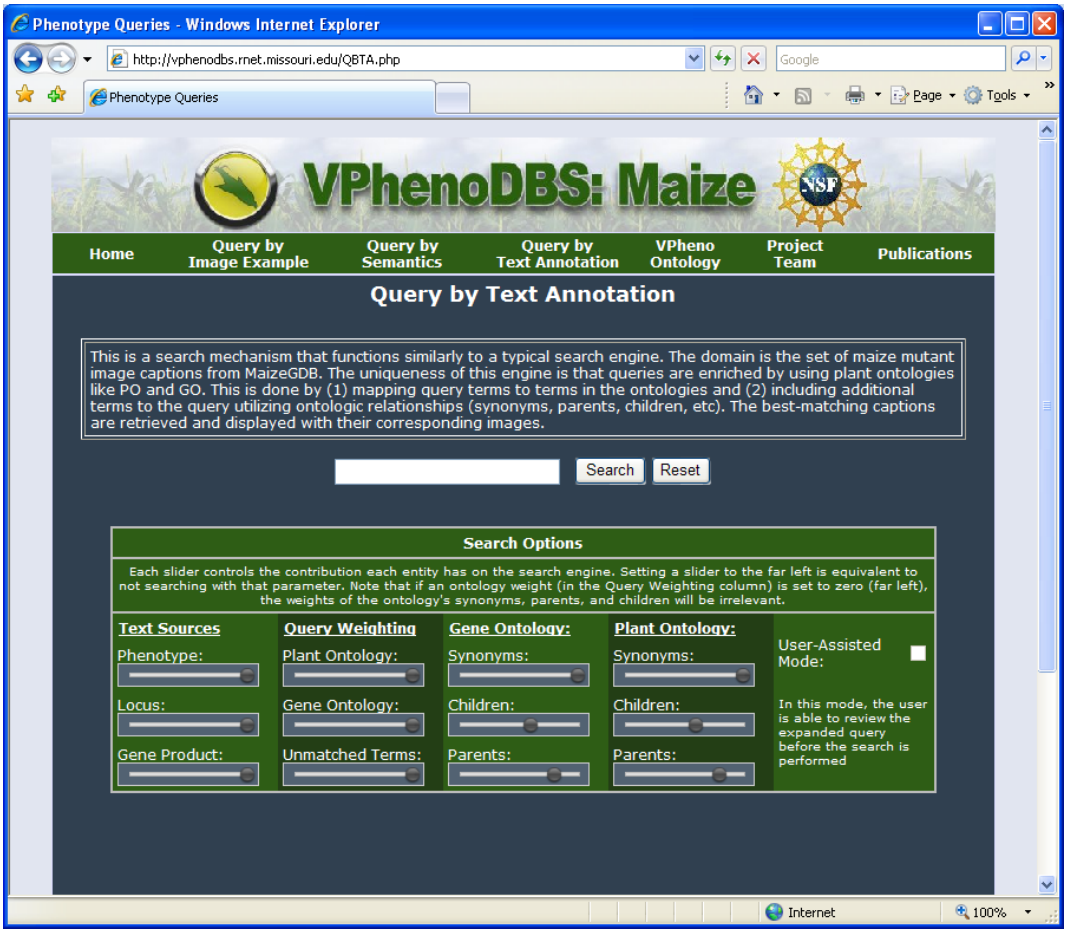

Figure 13: Search interface for the maize phenotype retrieval engine.

The first column regards the available searchable text sources. The slider bars can be used to select which text sources are included in the search and also how much emphasis to place on each source. Moving a slider to the far left removes that text source from the search. For example, if the locus slider bar was at the far left, retrieval would be done based solely on the phenotype captions and gene product descriptions. If a slider bar is not at the far left, then a text source is included in the search and the position of the slider bar determines the amount of emphasis placed on that source (relative to the other sources). The sliders are defaulted to the far right, which corresponds to full weight for a column. By moving a slider to the left, the emphasis of that text source is decreased. For

example, moving the locus slider to the middle and leaving the other two sliders at the far right gives the text in the locus descriptions half the emphasis (weight) of the caption and gene product descriptions.

The second column has sliders that control the emphasis of the query terms. The top two sliders control the weight for each searchable ontology. In addition, if an ontology slider has a non-zero weight, then it will be used for query expansion. The bottom slider in this column controls the weight for query terms that did not match an ontology. This gives the user the flexibility to really stress specific types of terms.

The third and fourth columns control the emphasis of terms added through query expansion from the GO and PO, respectively. These sliders determine what kinds of ontology terms are used to expand the query and how much weight is given to each type of term (again, relative to the other term types). Initial weights give synonyms the most weight, followed by parent terms and then children. This recommendation is based on the results in [31].

Query expansion can be performed in three modes: manual, user-assisted, and automatic. By default, query expansion is done automatically, meaning that terms are added to the query without consulting the user. The option available in the last column gives the user the chance to enter user-assisted mode. In this mode, the terms that are usually automatically added to the query are presented to the user for confirmation or refusal. The user can decide to expand or not expand any term in that list, which helps ensure that the query does not deviate from the user's information needs.

8.2. Viewing the Ranked Results

After the query is submitted, the search engine retrieves and ranks document cases in decreasing order of similarity to the query. For the sample query "tryptophan lysine kernel," the results page is shown in Figure 14.

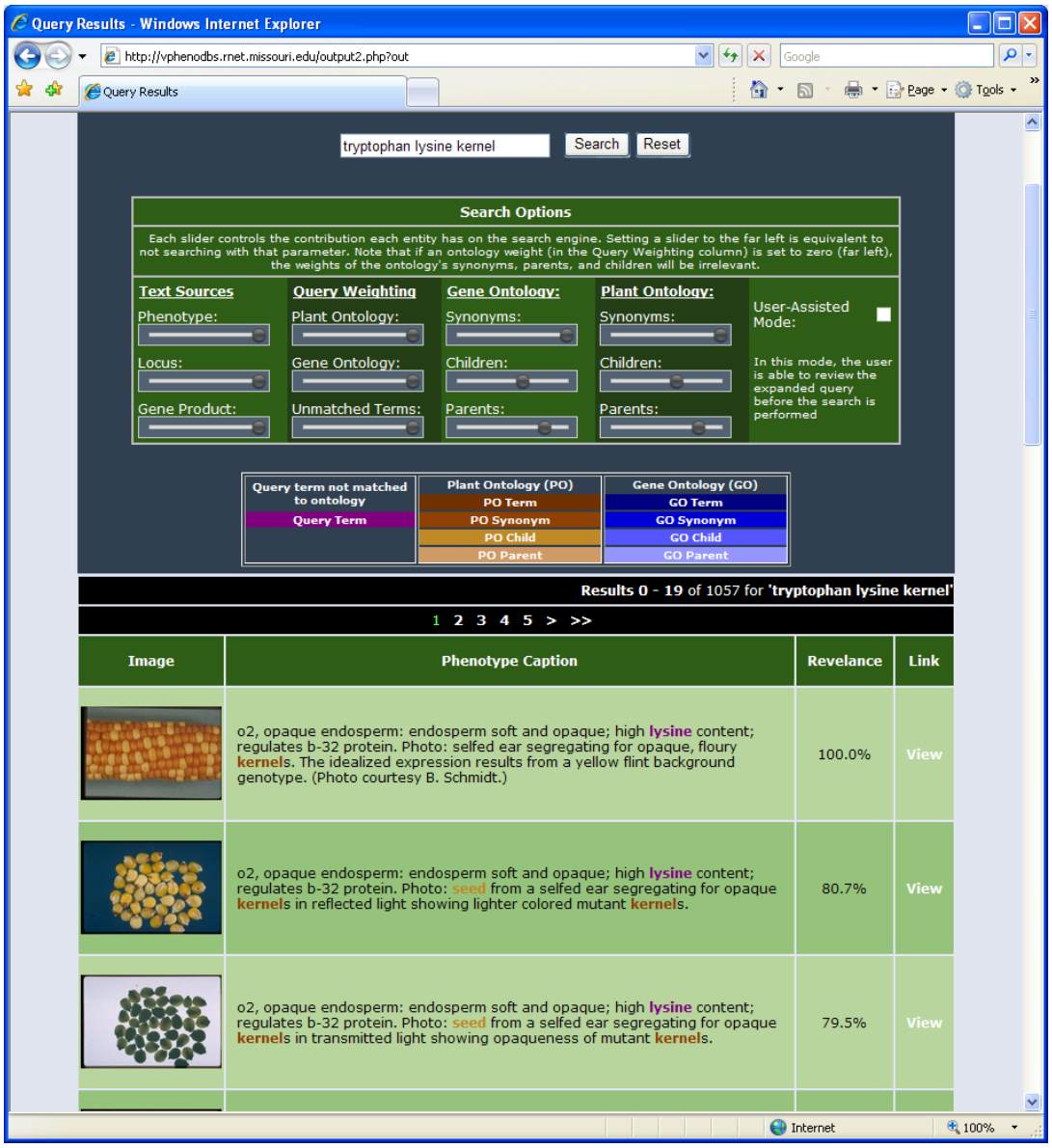

Figure 14: Search results for "tryptophan lysine kernel."

At the top of the page, the user's query and the current system weights are redisplayed in the search interface. This interface is repeated so that the user can make adjustments, to the query and/or other options, and resubmit a query.

Below the search interface is the list of ranked document cases. With each document case, the phenotype image, image caption, and relevance are provided to the user. Relevance is calculated using Equation 22, which considers two factors: (1) the similarity of a document case relative to the similarity of the top-ranked document case and (2) the percentage of query terms that appeared in the corpus. The first factor is a normalization term, and the second penalizes the relevance for each query term that was never matched.

$$rel_i = \frac{sim(F(i,K),q)}{max_j \{ sim(F(j,K),q) \}} * \frac{\left|\{q_i \mid \exists\, k,j \text{ such that } w_{j,i}^k > 0\right|}{|\{q_i\}|} \qquad (22)$$

Why is it that the only text field shown in the initial view of the results is the phenotype caption? This is because the phenotype captions are the base document set for his retrieval engine. Since each document case may have several documents in each secondary field, displaying the entire document case as one row in a list of results did not seem feasible. Thus, it was decided that a summary of each document case, not including the secondary documents, would be displayed originally. To view an entire document case, including *all* associated secondary documents, the user can click on the provided link in the last column (see Figure 15).

8.3. Displaying Matched Query and Ontology Terms

When performing query expansion with synonyms, parents, and/or children, it is possible that a query will be enriched with many terms from the available ontologies. As such, results containing few words from the original query may appear in the top-ranked

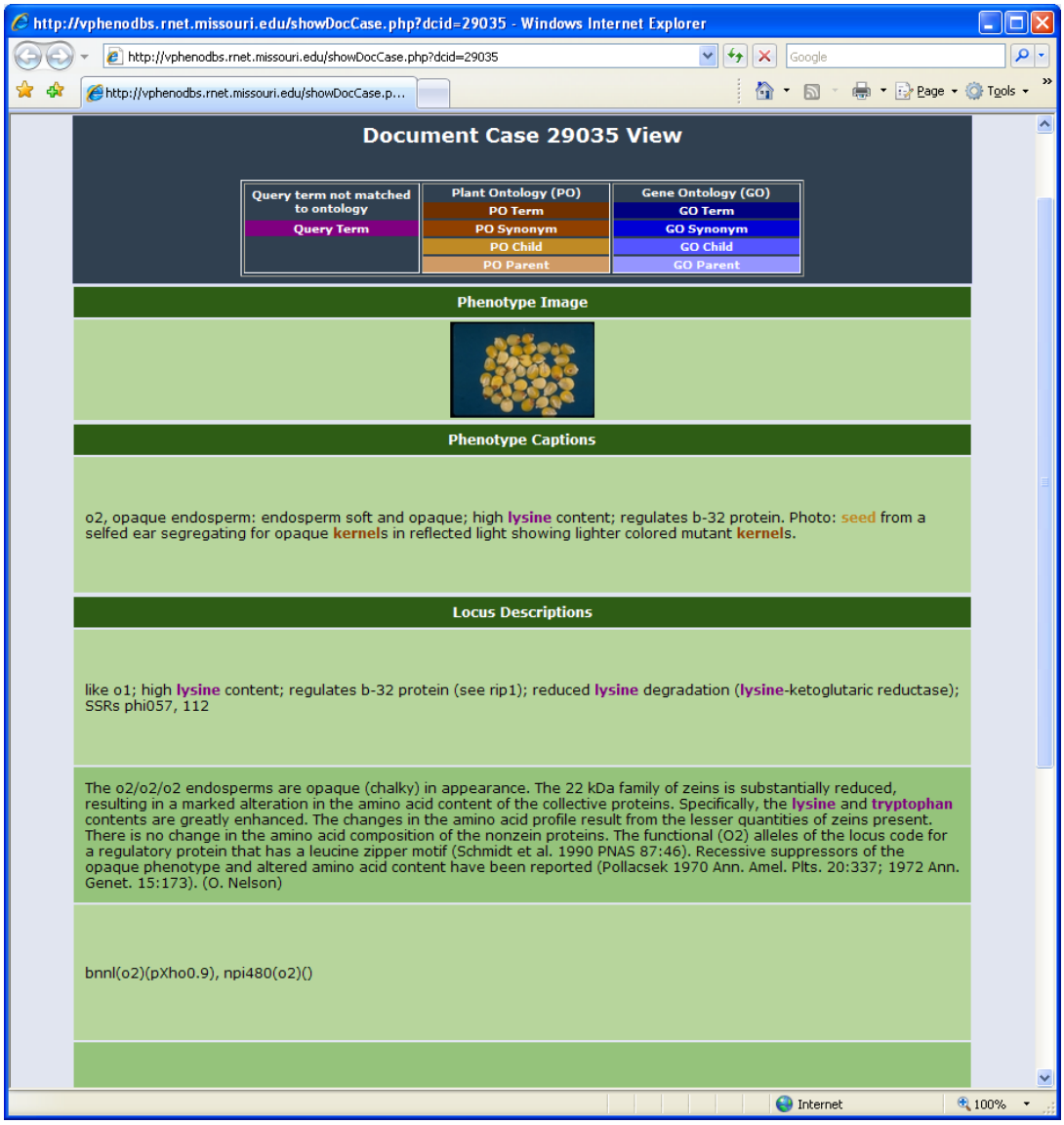

Figure 15: A sample document case, viewable by clicking the link associated with one result in the ranked results.

results. To aid the user in determining which terms were added and how they relate to the query, color highlighting of terms is provided. Terms matched to or added from a specific ontology are given the same general color. Different shades of that color will distinguish the relationship as matched term, synonym, parent, or child and are defined in the legend. By placing the mouse over a highlighted term, additional information about

the term will be shown in a tool tip. This information includes the ontology to which the term matched, the ontology ID of the term, and the relationship of the term to the query.

CHAPTER 9

DYNAMIC KNOWLEDGE MANAGEMENT

This chapter discusses the management of the search engine with respect to the dynamic nature of the knowledge underlying the search engine as well as the flexibility inherent in the system. As the data that underlies the search engine is dynamic, meaning that there are rather frequent updates to the corpus or ontologies, a discussion of how these actions can be handled to ensure up-to-date search results are discussed. Also, the flexibility of the system inasmuch as its ability to allow additional text sources or ontologies to be included in the search engine is also addressed.

9.1. Handling Changes to the Corpus

The document corpus underlying this search engine will evolve. Massive amounts of research are underway in an effort to understand genotype/phenotype relationships, and through this research, a substantial amount of phenotypic and genotypic data will be collected, a portion of which will be submitted to MaizeGDB in the form of phenotypic and genotypic textual descriptions. As such, being able to automatically handle updates to the document corpus is important. Fortunately, for this search engine, changes to the

corpus are fairly easy to handle. It should be noted that the search engine, at least in its present state, is independent of the database storing the document corpus, though it does require read access in order to preprocess documents.

The case when documents are added to a searchable text source can be handled by executing the preprocessing script on only the new documents. The new documents from a text source *k* can be identified using Query 9, which compares the documents in the corpus database to the processed documents in the *Term Frequency* table. The variables prefixed with '$' correspond to information stored in the row corresponding to the text source *k* in the *Searchable Fields* table.

```
SELECT $tableKey
$queryBody AND $tableKey NOT IN
   (SELECT DISTINCT ID
      FROM Term_Frequency
     WHERE fieldID = $fieldID);
```

Query 9: SELECT statement to retrieve newly added documents to the corpus.

The above query is simply a set difference, all the documents in text source *k* minus the processed documents. The *$queryBody* column in the *Searchable Fields* table provides the linkage between text source k and the base document set. The preprocessing script will count and store raw TFs for the added documents. It will then execute the SQL statements in Query 10 to update the IDFs of terms in text source k.

If documents are deleted from a searchable text source, a similar set difference query (Query 11) can be executed to identify processed documents that no longer appear in the corpus database. Observe that Query 9 and Query 11 are set differences between the same two sets, only the direction of the operation is reversed. The entries for these

```
DELETE FROM IDF WHERE fieldID = $k;
INSERT INTO IDF (fieldID, term, frequency
  SELECT fieldID, term, LOG10(C.Num/COUNT(ID))
    FROM Term_Frequency T,
        (SELECT COUNT(DISTINCT ID) AS Num
           FROM Term_Frequency
          WHERE fieldID = $k) C
   WHERE T.fieldID = C.fieldID AND fieldID = $k
GROUP BY 1,2;
```

Query 10: SQL statements to update the IDFs for a specific text source.

documents could then be deleted from the search engine *Term Frequency* table. Because these deletions may affect IDFs, Query 10 should be executed to recalculate IDFs for text source k. After this, the search engine will be free of the effects of the deleted documents.

```
SELECT DISTINCT ID
  FROM Term_Frequency
 WHERE fieldID = $fieldID AND ID NOT IN
   (SELECT $tableKey $queryBody);
```

Query 11: SQL statement to determine processed documents that have been deleted from the corpus.

The final case is handling updates to documents in a searchable text source. Unfortunately, there is no straightforward way to identify updated documents. If the corpus database logged changes, a list of changed documents could be formed through the log. Alternatively, an occasional reprocessing of the entire document corpus could be performed to synchronize the corpus database with the search engine tables.

It should also be noted that in practice, tables mirroring the *TF* and *IDF* tables exist for preprocessing. In this way, partial or complete corpus reprocessing can be conducted

without affecting the search engine. Once the reprocessing is complete, the updated TF and IDF values can be moved to the production tables.

9.2. Handling Changes to an Ontology

It is probably safe to say that no ontology is complete; all are in a constant state of evolution: terms are being added, others are being deprecated, and relationships between terms are being identified. With the current approach, there is no processing of ontologies, so there is no work to be done on that end. However, as documents may be indexed by terms matching ontology concepts, there is the potential for (1) indexed terms to be removed from the ontology and (2) new terms to be added to the ontology, which may match documents in the corpus. Because of these effects, the best course of action is to reprocess the entire document corpus when the newest version of an ontology is loaded into the ontology database.

9.3. Adding a Text Source to the Retrieval Engine

As discussed in Section 3.2, the *Searchable Fields* table contains a row for every text source in the database that is currently linked to the base document set in the search engine. If a new source that would be useful to the search is identified, all it requires is first an INSERT statement followed by processing of all the documents in the newly added text source. As an illustration, consider the values for the row regarding locus descriptions in Table XV.

The *fieldID* column does not need to be specified. As this column is AUTO_INCREMENT, it will automatically be filled with the next available ID. The text source needs to be identified by table and column; in addition, the primary key (PK) of the table should be specified in *tableKey*. These three database columns are used to

Table XV: Values for the *Searchable Fields* table for locus descriptions.

COLUMN	VALUE
fieldID	2
tableName	MEMO
columnName	memo
tableKey	auto_num
queryBody	FROM WEB_IMAGE I, VARIATION V, MEMO M WHERE I.ID=V.ID AND V.variationof=M.ID and M.memo IS NOT NULL
extractAllHeader	SELECT M.memo, M.auto_num
getCaptionHeader	SELECT I.auto_num
dateProcessed	04/20/2009

associate text with their IDs, which are linked via the next column, *queryBody*, to the base document set. The *queryBody* field provides the series of database links by which documents in this secondary text field are linked to base documents. By affixing one of the next two fields, *extractAllHeader* and *getCaptionHeader*, to the front of *queryBody*, a query is formulated that implements the mapping. The direction of the mapping is determined by which field begins the query. If *extractAllHeader* begins the query, then secondary documents are returned that link to base documents, and if *getCaptionHeader* starts the query, then base documents are returned that link to secondary documents. Additional restrictions can be suffixed onto these queries for more specific use, e.g. finding all the base documents associated with a specific secondary document by adding " AND M.auto_num = <fill in the blank>." The *dateProcessed* field is written automatically whenever the entire text source is processed.

Once the row is inserted into the *Searchable Fields* table, the preprocessing script should be run on all the documents in that source followed by Query 3 to calculate IDFs.

9.4. Adding a Ontology to the Retrieval Engine

Analogous to the *Searchable Fields* table, there is a *Searchable Ontologies* table that contains information about the ontologies available for use in the search engine. Adding a new ontology to the search engine requires that a row be added in the *Searchable Ontologies* table. In addition, the database schema of the ontology may need to be adjusted to mirror those of the GO and PO. As an illustration, consider the addition of the Trait Ontology in Table XVI.

Table XVI: Potential values for the *Searchable Ontologies* table for adding the Trait Ontology.

COLUMN	VALUE
ontAbbr	TO
ontName	Trait Ontology
ontDB	trait_ontology
versionDate	03/31/2009

One will observe that a lot less information is stored here than in the *Searchable Fields* table. This is because this table makes a lot more assumptions, specifically about the structure of the ontology database. Quickly, the ontology abbreviation is the prefix given to accessions in this ontology. The ontology database is the name of MySQL database where the TO tables are stored. The search engine assumes each ontology has the same basic structure and that each are located in a dedicated database. Finally, the version date specifies when the ontology was last updated.

The more difficult aspect to adding an ontology to the retrieval engine is potentially the conversion of its native structure to the assumed structure, which again is the

structure of GO and PO. Specifically, only three tables are required: *term*, *term2term*, and *term_synonym*.

Once the ontology is represented in this format, the entire document corpus should be reprocessed so that documents can be matched to this new ontology.

CHAPTER 10

CONCLUSIONS AND FUTURE WORK

10.1. Conclusions

In this thesis, a unique phenotype search engine for the maize community was discussed extensively. By surveying the currently available search mechanisms available to the major plant genome groups, with specific attention to phenotype searches, the need for a more sophisticated phenotype retrieval engine was identified. All current search mechanisms relied on traditional SQL queries, with no ranking of results, which can lead to poor usability if there are lots of results to manually sift through. It was also noticed that a wide variety of phenotypic and genotypic data were being stored in these databases, but no advanced search utilities were available to take advantage of these interconnected data. In addition, although domain ontologies were available for plants as well as genetic information, they had not been incorporated into any current plant search mechanisms.

In response to these needs, a multi-source ontology-based phenotype search engine was built. First, an extension to the vector space retrieval model was developed specifically to handle the document case concept, so that instead of retrieving individual documents, interconnected data from multiple sources could be retrieved and ranked as a group. This involved the formulation of a novel similarity measure to compare queries with document cases. Two flavors of the similarity measure were defined (merge and selection), and the situations in which each were appropriate were discussed. A document case in the developed phenotype retrieval engine consisted of a phenotype image caption (base document) as well as its related locus and gene product descriptions (secondary documents).

The phenotype retrieval engine also sought to utilize the knowledge embedded in domain ontologies to improve search results. This was accomplished through an information retrieval technique called query expansion, in which terms are added to the query in an effort to provide better contextualization for the query and increase retrieval accuracy. If an ontology concept was identified in the query, the synonyms, parents, and children of that concept were included in the query. The PO was utilized for concept matching in the plant realm, with the GO provided the vocabulary for the gene and gene product domains.

Techniques for managing the dynamic knowledge underlying the search engine were discussed, including updates to the document corpus and utilized ontologies and also the addition of additional ontologies or text sources in the search.

The principles used to build the search engine are general enough to be used in other domains. Specifically, the multi-source aspect can be applied to any corpus consisting of

heterogeneous, interconnected text sources, and the ontology-based query expansion can be implemented whenever a domain-specific ontology is available.

10.2. Future Work

The development of this phenotype search engine paves the way for several future directions. These are described below:

- The current search mechanism has only been applied to maize. Porting the framework to other studied organisms could be done. More interesting, though, is the possibility of combining data from multiple organisms to build a cross-species phenotype search. This would enable the retrieval of similar phenotypes from several genomes.
- The current results visualization is strictly text-based; the descriptions that are used in the search are displayed for the user. However, more advanced visualization tools are available that could be utilized by this retrieval engine. For example, integration of the results into the Genome Browser, which would allow the user to visualize the location of loci and gene products in a genome, may be a very useful feature for plant science researchers.
- Evaluation of the proposed framework is necessary; however, there are some obstacles. Because of the novelty of multi-source retrieval, no benchmark datasets are available for measuring the standard search engine performance measures (precision and recall). Our plan is to evaluate the system qualitatively as well as quantitatively. Several domain experts will be recruited for the study. Using a set of predetermined queries, with at least one corresponding to each combination of available text sources, the domain experts will be asked to first use current

phenotype search mechanisms to try and find the desired information. A set of questions regarding usability, time, and quality of results will be administered. The users will then follow the same procedure using our search mechanism, answering the same questionnaire. In addition to the usability questions, the users will also denote which of the top results are relevant and which are irrelevant. A comparison of the usability studies will give us a qualitative assessment of the search engine. The identification of relevant versus irrelevant results from our new search engine will allow quantitative assessment, specifically the calculation of precision.

- The requirement of the normalized TFs to be calculated on-the-fly will cause the search engine to become slower as the corpus size increases. Ways to maintain search speed as the database scales may need to be investigated. To speed up retrieval, it is proposed that more advanced indexing structures be utilized. By indexing with an M-Tree [5] or EBS k-d tree [25], document vectors closest to a query vector, with respect to some defined distance metric, could be retrieved very quickly even with a very large corpus. This would require calculation of normalized TFs during the preprocessing step. It is hypothesized that traditional normalized TFs could be used, even with double counting, in the indexing structures for document vector approximation. These approximations could be used to initially retrieve well matching documents, and then, for only those documents that matched a query phrase, their TFs could be adjusted for the final ranking.
- In this retrieval engine, only text sources were utilized. A very interesting extension would be to consider the inclusion of other types of information sources, such as sequences, images, etc. Sequence data would allow a user to find

phenotypes that were linked to specific types of sequences and contained textual descriptions related to the text query. As for images, we have already explored in a previous work [26] how to represent phenotype images as feature vectors. In both cases, however, an approach to intelligently combine these various types of sources remains unstudied.

BIBLIOGRAPHY

[1] A Aronson. (2001) Effective Mapping of Biomedical Text to the UMLS Metathesaurus: The MetaMap Program. In Proc. of the AMIA Symposium, 2001.

[2] R Baeza-Yates, B Ribeiro-Nero (1999) Modern Information Retrieval, Addison Wesley

[3] S Banerjee and T Pedersen (2002) An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet. Computational Linguistics and Intelligent Text Processing, Springer Berlin/Heidelberg, vol 2276, pp 117-171, 2002.

[4] J Bhogal, A Macfarlane, P Smith. (2007) A review of ontology based query expansion. Information Processing and Management, vol 43, pp 866-886, 2007.

[5] P Ciaccia, M Patella, P Zezula. (1997) M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In Proc. of the 23rd VLDB Conference, 1997.

[6] C Fellbaum (ed.): WordNet: An Electronic Lexical Database, MIT Press, 1999.

[7] G Fu, C Jones, A Abdelmoty. (2005) Ontology-based Spatial Query Expansion in Information Retrieval. Intl Workshop & Tutorial on Adaptive Text Extraction and Mining, 2005.

[8] GW Furnas, S Deerwester, ST Dumais, TK Landauer, RA Harshman, LA Streeter, KE Lochbaum. (1988) Information retrieval using a singular value decomposition model of latent semantic structure. In Proc. of the 11th Annual Intl ACM SIGIR Conference on Research and Development in Information Retrieval, pp 465-480, 1988.

[9] The Gene Ontology Consortium. (2000) Gene Ontology: tool for the unification of biology. Nature Genet. 25: 25-29, 2000.

[10] S Harabagiu and F Lacatusu. (2005). Topic themes for multi-document summarization. In Proc. of the 28th Annual Intl ACM SIGIR Conference on Research and Development in Information Retrieval, pp 202-209, 2005.

[11] P Jaiswal, S Avraham, K Ilic, EA Kellogg, SR McCouch, A Pujar, L Reiser, SY Rhee, MM Sachs, ML Schaeffer, LD Stein, PF Stevens, LP Vincent, DH Ware, F Zapata. (2005) Plant Ontology: A controlled vocabulary of plant structures and growth stages. Comparative and Functional Genomics, 2005, vol 6 (7-8), 388-397

[12] P Jaiswal, J Ni, I Yap, D Ware, W Spooner, K Youens-Clark, L Ren, C Liang, W Zhao, K Ratnapu, B Faga, P Canaran, M Fogleman, C Hebbard, S Avraham, S Schmidt, TM Casstevens, ES Buckler, L Stein, S McCouch (2006) Gramene: a bird's eye view of cereal genomes. Nucleic Acids Research, 34: D717-723.

[13] P Jaiswal, D Ware, J Ni, K Chang, W Zhao, S Schmidt, X Pan, K Clark, L Teytelman, S Cartinhour, L Stein, S McCouch (2002) Gramene: development and integration of trait and gene ontologies for rice. Comparative and Functional Genomics 3: 132-136.

[14] N Kurata and Y Yamazaki. (2006) Oryzabase: An Integrated Biological and Genome Information Database for Rice. Plant Physiology 2006 140:12-17.

[15] CJ Lawrence, LC Harper, ML Schaeffer, TZ Sen, TE Seigfried, DA Campbell. (2008) MaizeGDB: The Maize Model Organism Database for Basic, Translational, and Applied Research. Int J Plant Genomics. 2008:496957.

[16] C-S Lee, Z-W Jian, L-K Huang. (2005) A Fuzzy Ontology and Its Application to News Summarization, IEEE Trans. On Systems, Man, and Cybernetics, vol 35, no 5, pp 859-880, October 2005.

[17] X Li, S Szpakowicz, S.Matwin. (1995) A WordNet-based Algorithm for Word Sense Disambiguation. Intl Joint Conference on Art. Intel. 1995.

[18] LA Mueller, TH Solow, N Taylor, B Skwarecki, R Buels, J Binns, C Lin, MH Wright, R Ahrens, Y Wang, EV Herbst, ER Keyder, N Menda, D Zamir, SD Tanksley. (2005) The SOL Genomics Network. A Comparative Resource for Solanaceae Biology and Beyond. Plant Physiology 2005 138:1310-1317.

[19] R Navigli and P Velardi. (2003) An Analysis of Ontology-based Query Expansion Strategies. In Proc. of the Intl Workshop & Tutorial on Adaptive Text Extraction and Mining, September 2003.

[20] Phenotype and Trait Ontology. http://bioontology.org.

[21] BA Ribeiro-Neto and R Muntz. (1996) A belief network model for IR. In Proc. of the 19th Annual Intl ACM SIGIR Conference on Research and Development in Information Retrieval, pp 253-260, 1996.

[22] SE Robertson, K Sparck Jones. (1976) Relevance weighting of search terms. Journal of the American Society for Information Sciences, vol 27, no 3, pp 129-146, 1976.

[23] G Salton, A Fox, H Wu. (1983) Extended Boolean Information Retrieval. Communications of the ACM. vol 26, no 11, pp 1022-1036, November 1983.

[24] G Salton, A Wong, CS Yang. (1975) A Vector Space Model for Automatic Indexing. Information Retrieval and Language Processing. vol 18, no 11, pp 613-620, November 1975.

[25] G Scott and C-R Shyu. (2003) EBS k-d Tree: An Entropy Balanced Statistical k-d Tree for Image Databases with Ground-Truth Labels. In Proc. of the Intl Conference of Image and Video Retrieval, 2003.

[26] C-R Shyu, J Harnsomburana, J Green, A Barb, T Kazic, M Schaeffer, E Coe. (2007) Searching and Mining Visually-Observed Phenotypes of Maize Mutants. In Journal of Bioinformatics and Computational Biology: Special Issue on Making Sense of Mutations Requires Knowledge Management, vol 5, no 6, pp 1193-1213, December 2007.

[27] SoyBase and the Soybean Breeder's Toolbox. http://soybase.org/index.php

[28] D Swarbreck, C Wilks, P Lamesch, T Berardini, M Garcia-Hernandez, H Foerster, D Li, T Meyer, R Muller, L Ploetz, A Radenbaugh, S Singh, V Swing, C Tissier, P Zhang, E Huala (2008) The Arabidopsis Information Resource (TAIR): gene structure and function annotation. Nucleic Acids Research 2008 36:D1009-D1014.

[29] H Turtle and WB Croft. (1990) Inference networks for information retrieval. In Proc. of the 13th Annual Intl ACM SIGIR Conference on Research and Development in Information Retrieval, pp 1-24, 1990.

[30] D Widdows, S Peters ,S Cederberg, C-K Chan. (2003) Unsupervised Monolingual and Bilingual Word-Sense Disambiguation of Medical Documents using UMLS. Proc. Of ACL 2003 Workshop on NLP, 2003.

[31] D Wollersheim and WJ Rahayu. (2005) Ontology Based Query Expansion Framework for Use in Medical Information Systems. Journal of Web Information Systems, vol 1, no 1, March 2005.

[32] SKM Wong, W Ziarko, PCN Wong. (1985) Generalized vector space model in information retrieval. In Proc. 8th ACM SIGIR Conference on Research and Development in Information Retrieval, pp 18-25, New York, USA, 1985.