An Implementation of Prioritized Queries using the Google SOAP Search API

by

David Kerins

B.Sc., University of Alberta, 1989

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

An Implementation of Prioritized Queries using the Google SOAP Search API

by

David Kerins

B.Sc., University of Alberta, 1989

Supervisory Committee

Dr. William W. Wadge, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)

Dr. Ulrike Stege, Departmental Member
(Department of Computer Science)

Dr. Peter Driessen, External Examiner
(Department of Electrical and Computer Engineering)

Supervisory Committee
Dr. William W. Wadge, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)

Dr. Ulrike Stege, Departmental Member
(Department of Computer Science)

Dr. Peter Driessen, External Examiner
(Department of Electrical and Computer Engineering)

Abstract

In this thesis, we provide an implementation of prioritized queries using the Google SOAP Search API. These prioritized queries are generated from the parsing of valid prioritized constraint expressions (PCE). Prioritized constraint expressions use two unary operators (μ and ω) to enhance an infinitesimal logic framework to allow preferential constraints and backup constraints respectively. The Perl CGI algorithm parses valid prioritized constraint expressions using a grammar and then generates a prioritized list of standard Google search queries. These standard Google queries are sent one at a time to Google via the Google SOAP Search API. The results of the search are returned as structured data to the user with their associated Truth value.

Table of Contents

List of Tables

List of Tables

List of Figures

List of Figures

Acknowledgments

Thanks to Dr. William Wadge for his support andnearly unfailing belief that I could get this done.

Thanks to my committee members for taking the time to read this thesis and attending the defense.

Finally, thanks to my family. There were missed weekends in my pursuit of academic credentials. That won't happen again! In particular, my wife Kristine who picked up the slack I created finishing this degree.

Dedication

This work is dedicated to my kids, Athena and Triumph, who missed their father on some sunny days under the apple tree.

1 Introduction

Information retrieval is a daily occurrence. Retrieving correct information is important. Equally important is finding information quickly and easily. It is for this reason that traditional information sources such as reference books and encyclopedias contain tables of contents and indexes. Easily finding the required information adds additional value or usefulness to the information source. If information is impossible to find or requires a great deal of effort to find then the usefulness of the source of information is diminished.

Early in the development of the World Wide Web websites that indexed evena subset of all possible web pages were required and highly valued. Yahoo! began as a pet project of two Stanford students to keep track of interesting web pages. In testament to the importance of this information retrieval function, Yahoo! has today become one of the largest Internet companies. As a further indicator of the importance of information retrieval on the World Wide Web, the rise of Google as market leader in web search emphasizes that quick and relevant information retrieval is extremely important for users of the web.

1.1 Problem and Motivation

Despite the usefulness of tools like Yahoo! and Google in helping web users, both have one glaring omission. Search engines such as Google assume that their users *know* what they are looking for. A Google user is required to enter some search words and all of these words are treated as required words of equal importance. If a user is certain of the information they wish to retrieve then the use of several search words should create a very focused search which will hopefully lead to the specific information they need. However, if the user is uncertain of the importance or applicability of a search word including that search word – which is given equal priority by Google – will unreasonably constrain the search results. Unfortunately, with this standard search model there is no notion of user preference or prioritization of the search words, all words are of equal priority so a user cannot specify different priorities between search words nor can they infer some preference in the combination of the search words.

The desire to apply preference and priority to search terms provides the motivation for this thesis. By preprocessing well formed Prioritized Constraint Expressions (PCEs), generating a series of standard Google queries and then submitting these in prioritized order, the software will create a result set which reflects the preferences and priorities that the user requires. By moving beyond the standard boolean notion of AND and OR in Google queries, this thesis and its supporting software allows a Google user to prioritize combinations of search words effectively by giving higher and lower priority to different combinations of these words. This priority ranking is represented in the search results. Furthermore, the implementation also offers a method for allowing some

compromise in acceptable search results when there are no results for certain combinations of search words. This enhancement to Google search allows for situations where there are too many search results and it would be nice to prioritize them as well as for situations where there are no search results and the user is willing to compromise on the rigidity of their search word choices.

1.2 Prioritized queries

This thesis is based on two notions of soft constraints: preferential constraints and backup constraints – represented by the unary operators μ and ω respectively -- developed in Agarwal [1]. Expressions using μ and ω to express infinitesimal truth values are called Prioritized Constraint Expressions (PCEs).

In turn, the concept of soft constraints is based on the work by Rondogiannis and Wadge [2] on infinitesimal logic. Infinitesimal logic presents an uncountable linearly ordered set of truth values where T_0 and F_0 represent the classical notions of true and false. Applying infinitesimal truth values to search words in a standard Google search query requires preprocessing of valid prioritized constraint expressions because Google has no notion of prioritized search words. Nonetheless, allowing the user to use preferential and backup constraints for Google searches addresses the rigidity of Boolean based search technologies with regard to the fuzzy nature of information retrieval employed by a human being who is not quite sure what they are looking for.

1.3 Thesis Outline

This thesis goes on, in Chapter 2, to present a survey of the literature on Classical Logic, Non-Aristotelian Logic, Three-Valued Logic, Multi-Valued Logic, Infinitesimal Logic, Information Retrieval Systems, and the technology behind the Google Search Engine. In Chapter 3 some key topics in Prioritized Constraint Expressions are covered. In Chapter 4, the Prioritized Constraint Expressions are implemented in software. Chapter 5 shows the browser based interface to the Prioritized Constraint Expression Search Engine. In Chapter 6, the Google SOAP Search API and the SOAP protocol are discussed. In Chapter 7, the usability of this implementation of prioritized constraints for Google search is covered. Chapter 8 provides some conclusions of the work presented.

2 Literature Survey

2.1 Classical Logic

Infinitesimal logic has roots that reach deep into antiquity. Classical logic, also called Aristotelian logic [3], is the well-spring for all modern logics including infinitesimal logic. Classical logic has been well studied and well defined. For example, Gabbay [4] outlines a number of properties which define a classical logic:

1. Law of the excluded middle and Double negative elimination;
2. Law of non-contradiction;
3. Monotonicity of entailment and Idempotency of entailment;
4. Commutativity of conjunction;
5. De Morgan duality: every logical operator is dual to another.

Of particular interest is the *Law of the excluded middle* which states that some proposition either is true or false. This law gives classical logic its bivalent character, meaning that the only possible truth values are *true* and *false*. An example of Classical logic is Aristotle's Syllogistic [5]. Classical logic forms the stepping off point for future logics which have more than two truth values.

2.2 Non-Aristotelian Logics

What characterizes modern logics is the rejection of one or more of the properties of classical logic. The rejection of some classical logic property allows for the expansion of the logic system in some interesting way, usually by increasing the number of truth values that are valid for the logic.

2.3 Three-valued Logic

As its name implies, three-valued logic adds one more truth value to the two classical logic truth values of true and false. The Law of the excluded middle is rejected to allow this new term to occupy a position somewhere in between true and false. Prominent versions of three-valued logic include the work by Łukasiewicz and Kleene. Łukasiewicz introduced a third truth value, "possible", in an attempt to handle the modalities "it is necessary that" and "it is possible that". These can be found in Aristotle's modal logic [6] and are further used as an attempt to handle Aristotle's sea battle paradox[7]. Kleene, in his three-valued logic [8], introduced an “undefined” truth value that was used with respect to partial recursive functions. According to Gottwald, Kleene's “connectives were the negation, the weak conjunction, and the weak disjunction of the 3-valued Łukasiewicz system together with a conjunction $\wedge+$ and an implication $\rightarrow+$ determined by truth degree functions” [9]. This can be shown in the following function tables:

$\wedge_+$	0	½	1
0	0	½	0
½	½	½	½
1	0	½	1

Table 1: Kleene Conjunction Function

$\rightarrow_+$	0	½	1
0	1	1	1
½	½	½	½
1	0	½	1

Table 2: Kleene Implication Function

Kleene's "undefined" value in these function tables is represented by ½. The value 1 represents the classical notion of truth (verum) and the value 0 represents the classical notion of false (falsum).

2.4 Multi-valued Logic

It follows that if the bivalent Classical logic was expanded to create a three-valued logics, then it could be expanded further to allow even more truth values. Multi-valued logics are just that, the premise being that there is no limit to the number of truth values in the set of truth values. Modern multi-valued logics range from three truth values to logics with infinite truth values in their set of truth values. Within a set of truth values of a multi-valued logic 0 and 1 still represent the verum and falsum of classical logic and the other truth values represent degrees of truth. Across the range of multi-valued logics,

there is no standard agreed upon interpretation of these truth degrees, in fact the interpretation of truth degrees depends on the application of the logic [9]. However, any formalized language of multi-valued logic has as a feature truth degree constants. According to Gottwald [9], a system based on many-valued logic contains the following characteristics:

- the set of truth degrees;
- the truth degree functions which interpret the propositional connectives;
- the meaning of the truth degree constants;
- the semantical interpretation of the quantifiers;
- the designated truth degrees, which form a subset of the set of truth degrees and act as substitutes for the traditional truth value "verum".

Multi-valued logics were initially developed to handle paradoxes created in Aristotle's modal logic. Since then multi-valued logics have been used, for example, for work in linguistics, philosophy, and artificial intelligence.

2.5 Infinitesimal Logic

Like multi-valued logics, Infinitesimal Logic also has an expanded set of truth values. Infinitesimal logic introduces an expanded truth domain that has an uncountable linearly ordered set of truth values between *False* (the minimum element) and *True* (the

maximum), with a *Zero* element in the middle. The truth values below *Zero* are ordered like the countable ordinals, while those above *Zero* have the reverse order [2].

Infinitesimal logic was first presented by Wadge and Rondogiannis as a pure model-theoretic semantics for *negation-as-failure* in logic programming. One of the outcomes of Infinitesimal logic and its expanded set of truth values is to distinguish, in a meaningful way, between classical negation ($\neg A$) and negation-as-failure ($\sim A$). This in turn allows for a meaningful distinction of degrees of truthfulness. In other words, negation-as-failure is considered to be classical negation but is subject to multiplication by an infinitesimal value ε. This has the effect of diminishing the truth or falseness of a truth value. Therefore T_0 and F_0 remain the representatives of the classical notions of verum and falsum and the first order truth values T_1 and F_1 are equivalent to $T_1 = \varepsilon T_0$ and $F_1 = \varepsilon F_0$. Second order infinitesimals are similarly represented to $T_2 = \varepsilon^2 T_0$ and $F_2 = \varepsilon^2 F_0$ and so on to infinity.

> Informally, we extend the domain of truth values and use these extra values to distinguish between ordinary negation and negation-as-failure (in fact, classical negation can be seen as strictly stronger that negation-as-failure in the sense that $\neg A$ is a more forceful statement than $\sim A$). [2]

The net effect is a set of infinitesimal truth values which allows a prioritization of truth values into a hierarchy of truth. This hierarchy of priorities corresponds to the degree of

infinitesimals and as such has applications in Information Retrieval systems insofar as creating priorities in the propositions or search string tokens.

2.6 Qualitative Choice Logic

The work of Brewka [10] is worth noting as it presents a propositional logic which allows context dependent preferences to be represented. The propositional logic is called Qualitative Choice Logic (*QCL*) and contains a connective "x" which represents ordered disjunction. In this logic, a statement like A x B means: if possible A, but if A is impossible then at least B. The Qualitative Choice Logic propositional logic is combined with logic programming to create logic programs based on rules with ordered disjunction. These programs are called logic programs with ordered disjunctions (*LPODs*) [11]. Ultimately, an approach which enhance logic programming with preference connectives is intended to increase the relevance of knowledge representation in information retrieval systems.

2.7 Information Retrieval Systems

Search engines are the most visible manifestation of the field of study known as Information Retrieval (IR). Information Retrieval is an old area of academic inquiry even predating modern computing. Information Retrieval as a discipline dates back to the 1940s and likely found inspiration in the evocative writings about the MEMEX by Bush [12]. Information Retrieval focuses on how to best search for textual information

in documents. The textual information is generally speaking unstructured. The information may be structured for humans to read but it is not structured in any way to facilitate searching and retrieval, especially when compared to a relational database which imposes substantial structure on the data to facilitate the search and retrieval of the information in an efficient manner.

In his acceptance speech for the Salton award, W. Bruce Croft [13] outlined the major contributions of Information Retrieval in computer science:

- provided the basic research on the algorithms and data structures for today's search engines;
- championed the statistical approach to language;
- developed an evaluation methodology which was compared to human judgment;
- has always acknowledged the importance of the user and interaction as a part of information access.

Another key contribution of Information Retrieval has been the development of performance measurements. These performance measurements help define a well performing Information Retrieval system. Three important Information Retrieval performance measurements are:

1. Precision
2. Recall

3. Fall-Out

Precision represents the proportion of retrieved *and* relevant documents compared to all the documents retrieved. *Recall* is the proportion of relevant documents that are retrieved from all relevant documents in the universe of documents available to the Information Retrieval system. Lastly, *Fall-Out* is the proportion of irrelevant documents that are retrieved from all of the irrelevant documents in the universe of documents available to the Information Retrieval system.

In [14], van Rijsbergen notes that *Precision* and *Recall* are the two most used and most paired measures of the effectiveness of Information Retrieval systems. These measures help assess the effectiveness of the Information Retrieval system but are only somewhat effective in truly determining relevance. The issue is that, regardless of what framework we may have to assess documents and mechanically determine relevance, relevance is subjective and therefore determining if a document is actually relevant is ultimately something that only the individual user can decide for themselves.

Despite the subjectivity of relevance, Information Retrieval systems are at their core frameworks for determining the relevance of documents based on the document's contents. Without the automatic analysis of relevance provided by IR systems the user would have to sift through mountains of documents. With an IR relevance framework the mountain is smaller and the user can apply their own subjective determination of relevance to a much smaller set of pre-qualified relevant documents.

Google is an Information Retrieval system and as such has a framework of relevance. Relevance on Google is determined by page specific factors, such as the anchor text of inbound links and the Google Page Rank algorithm. The page rank algorithm ranks a page as more relevant if there are many other highly ranked pages which link to the page in question. Other factors that determine the relevance of a page are discussed later.

2.8 Google Architecture and Algorithms

Google is a sophisticated, modern information retrieval system. The underlying algorithms of Google attempt to do four things:

1. Determine the universe of available pages.
2. Index the universe of pages to enable fast retrieval.
3. Determine the relevance of each page.
4. Retrieve an ordered list of relevant pages for a query.

Unfortunately, finding authoritative and comprehensive information on how Google works is difficult. One authoritative document is an early academic paper by the founders of Google [15]. This paper goes into some detail about the anatomy of the Google search engine. This document is now quite old and likely does not accurately describe the current state of the Google search engine. Nonetheless, this document does give a fairly accurate description of the Google search engine circa 1999.

Current authoritative and comprehensive documentation is scarce and today Google Inc. discloses few details about how their indexing and page ranking systems work. However, one way to indirectly determine the true functioning of the Google search engine is through analysis of patent applications submitted to the United States Patent and Trademark Office. For example, Google's US Patent Application #20050071741 - Information Retrieval Based on Historical Data [16] demonstrates the increasing importance of page longevity – how long ago the page was created – and how often the content of the page changes as key components of the page rank that the page receives. Therefore, for this thesis, various secondary sources have been used as references for understanding the mechanics of the Google search engine.

2.8.1 Googlebot

The Googlebot is a web crawling robot. Its job is to find and retrieve pages on the web. It forwards found pages to the Google Indexer. Essentially, the Googlebot searches for pages on the web in one of two ways. The first way is by web site owners directly submitting their URL to Google via the http://www.google.com/addurl/ web page. The second method is by finding links through crawling the web.

When crawling the web, the Googlebot harvests pages for the indexer. It also gathers all the hyperlinks on those pages for future crawling. In this way, the Googlebot

organically increases the universe of pages that it visits, harvests and forwards to the Google Indexer. Crawling the web is, in fact, done in a distributed fashion with many Googlebots harvesting pages as directed by a URLserver which provides lists of URL for the Googlebots to fetch [15]. Harvested pages are passed onto another process which assigns a unique document identification number for each unique URL and then compresses and stores each page in a repository. The full HTML contents of the page is stored in the repository. Each page is prefixed by its docID, length and URL. The uncompressed representation is seen in the following table which is taken from Brin and Page [15].

docID	ecode	urllen	pagelen	url	page

Table 3: Uncompressed Google Repository Contents

2.8.2 The Google Indexer

Pages returned by the Googlebot and stored in the repository are processed by the Google Indexer. The Google Indexer builds an index of pages by mining the repository and parsing the pages. For each page a set of word occurrences, known as *hits*, is created. The hits record the key word, its position in the page, font size and other characteristics. Each hit record is then stored in barrels to create a partially sorted forward index. The index function also parses out the hyperlinks contained in each page and stores them in a separate file, called an anchor file, associated with the unique document id for that indexed page. These anchor files are in turn the source for a links

database which is used for computing PageRank [17] for indexed pages. The index is sorted alphabetically by search term with each index entry maintaining a list of pages that contain the search term. Common words such as *the*, *and*, *is*, *on*, *or*, and *of* are not indexed by Google. In fact, Google strips common words from user's query strings as it preprocesses the query. Omitting such common words improves performance and does not have an effect on search results.

2.8.3 The Google Query Processor

Query processing does occur after the Googlebot and the Google Indexer have done their work. The Google Query Processor consists of the user interface that allows the end user to enter and submit a query, the query engine which evaluates the query and matches relevant pages from the Google index, and the page results formatter which returns page results for the query in order based on page relevance.

3 Prioritized Constraint Expressions

3.1 Infinitesimal Logic and Prioritized Query Operators

This thesis is based on two notions of soft constraints: preferential constraints and backup constraints. These constraints are represented by the unary operators μ and ω respectively developed in Agarwal [1]. Increasing the degree of μ and ω allows the creation of a hierarchy of constraints. Expressions using μ and ω to describe infinitesimal truth values are called prioritized constraint expressions (PCE).

In turn, the concept of these soft constraints is based on the work by Rondogiannis and Wadge [2] on infinitesimal logic. Infinitesimal logic presents an uncountable linearly ordered set of truth values where T_0 and F_0 represent the classical notions of true and false.

$$F_0 < F_1 < ... < F_\omega < ... < F_\alpha < ... < 0 < ... < T_\alpha < ... \; T_\omega < ... \; T_1 < T_0$$

Illustration 1: Uncountable linearly ordered set of truth values

The element *0* represents the notion of *undefined*. The intervening truth and false values, such as T_k and F_k, where *k* is a transfinite ordinal, express differing levels of truthfulness or falseness. As F_k approaches F_0 it is more false and as T_k approaches T_0 it is more true. (i.e. F_k is infinitely more false than F_{k+1} and T_k is infinitely more true than

T_{k+1}) The value of *k* is derived by multiplying T_0 or F_0 – the classical truth values – with an infinitesimal value denoted by ε. Thus there is

$$T_k = \varepsilon^k T_0$$

and

$$F_k = \varepsilon^k F_0$$

The infinitesimal logic truth values are linear ordered; this logic is therefore a *fuzzy logic* in the most general sense of the term [18]. However the phrase "fuzzy logic" normally refers to a logic in which the truth domain is the interval [0,1], which is continuous; infinitesimal logic, by contrast use distinct, discrete levels.

Applying infinitesimal truth values to search words in a standard Google search query requires some preprocessing of valid prioritized constraint expressions (PCE) because Google has no notion of prioritized search words. However, the use of preferential and backup constraints for Google searches addresses the rigidity inherent in Boolean based search technologies and users with information retrieval tasks when they are not quite sure what they are looking for.

3.2 Semantics of the Prioritized Query Operators

The preferential constraint and backup constraint as represented by the unary operators μ and ω respectively are used when creating a valid Priority Constraint Expression (PCE). The semantics of these Prioritized Query Operators are straightforward.

3.2.1 Preferential Constraint Operator μ

The unary operator μ represents the notion of a preferential constraint. Applying a preferential constraint operator to some string token (or word) accords both preference and priority to that string token/word in the Priority Constraint Expression. Applying a preferential constraint to a string token means that the string token is "preferred but not required" as a member of the Priority Constraint Expression and if concurrently there are other preferential constraint operators then the importance of the preferential constraints, relative to one another, is specified by the integers associated with the preferential operator. The μ operator can be combined with an integer to indicate the priority of the preferential constraint compared to other preferential constraints in the Priority Constraint Expression.

For example:

A μB μ2C

represents a valid Priority Constraint Expression which contains one *required* string token 'A' and two preferential constraint operators acting on two string tokens, 'B' and 'C'. (The three queries are implicitly conjoined; we want them *all* to be true, if possible.) Notice that one of the preferential operators has no integer associated with it. In fact, there is an implied '1' associated with μB, therefore μB and μ1B are equivalent. The lower the value of the integer the higher the priority of the preferential query operator and its string token. Therefore, μB (or its equivalent μ1B) is the more preferred preferential operator string token than is μ2C. Thus, in the prioritized constraint

expression

A μB μ2C

the string token A is required and B and C are “preferred but not required” with B having a higher priority – more preferred – than C.

As mentioned earlier, there is also an implicit boolean AND when combining required string tokens and preferential query operators. This is most apparent when the most preferred outcome is considered. In the prioritized constraint expression

A μB μ2C

the most preferred outcome would occur when the required string token 'A' is combined with all of the “preferred but not required” string tokens. Therefore the combined set of string tokens that would provide the most preferred results would be A and B and C. This can be spoken as “A and if possible B and if possible C”.

The truth table for preferential constraint is the following table.

A	B	C	Query	Result
T	T	T	A B C	T_0
T	T	F	A B -C	T_1
T	F	F	A -B -C	T_2
F	F	F	-A -B -C	F_0

Table 4: Preferential Constraint Truth Table

This last row of the truth table represents total failure and is designated as the F_0 query and result set. In theory the result set for F_0 would contain results which do not match the users request whatsoever and therefore the query would seek to exclude all preferred terms that are specified by the PCE. F_0 queries and potential result sets represent a theoretical aspect of the infinitesimal framework presented here.

3.2.2 Backup Constraint Operator ω

The unary operator ω represents the notion of a backup constraint. Applying a backup constraint operator to some string token designates that string token as a backup string for the string token to the immediate left. In other words, backup constraints are left associative. Like preference constraints, backup constraints can have an associated priority that indicates its priority compared to other backup constraint operators in the Priority Constraint Expression. For example:

$$A\ \omega B\ \omega 2C$$

represents a valid Priority Constraint Expression which contains one *required* string token 'A' and two backup constraint operators acting on two string tokens, 'B' and 'C'. (These constraints are implicitly disjoined; we want *one* of them to be true, preferably A.) As with the preferential operators, there is an implied '1' associated with ωB, therefore ωB and ω1B are equivalent. Again with regard to priority, the lower the value of the integer the higher the priority of the backup query operator and its string token. Thus, in the prioritized constraint expression

A ωB ω2C

the string token A is required and B and C are simply backup options in the event that A is false (has no result). Note that B has a higher priority than C thus it will be the first backup constraint token used.

As mentioned earlier, there is an implicit boolean OR when combining required string tokens and backup query operators. In the prioritized constraint expression

A ωB ω2C

the most preferred outcome would occur when the required string token 'A' alone returns a result. Failing that, the next most preferred result would be for the highest priority backup constraint string token 'B' to return a result. Lastly, the least preferred result would be for the lowest priority backup constraint string token 'C' to return some result. Therefore the set of string tokens that would provide the most preferred results would be, in this order, A or B or C. This can be spoken as “A or failing that, B or failing that, C”.

The truth table for a backup constraint is theoretically infinite therefore the following table is truncated for practical purposes. For a backup constraint prioritized constraint expression such as A ωB ω2C, the following truth table applies.

A	B	C	Query	Result
T	anything	anything	A	T_0
F	T	anything	-A B	T_1
F	F	T	-A -B C	T_2
F	F	F	-A -B -C	F_0

Table 5: Backup Constraint Truth Table

This last row of the truth table represents total failure and is designated as the F_0 query and result set. In theory the result set for F_0 would contain results which do not match the users request whatsoever and therefore the query would seek to exclude all preferred terms that are specified by the PCE. F_0 queries and potential result sets represent a theoretical aspect of the infinitesimal framework presented here.

With an understanding of the semantics of the unary prioritized query operators μ and ω in prioritized constraint expressions, the next step is implementing these prioritized constraint expressions as a series of ordered standard Google queries which will, as best as possible, represent the intention of the user when they set up preferential or backup constraints in their prioritized constraint expression.

3.3 Implementing Prioritized Constraint Expressions as Google Queries

This thesis attempts to bridge the gap between theory and reality. The associated software implements prioritized queries on top of the Google search engine by taking well formed Prioritized Constraint Expressions and generating a number of standard Google queries which will, as best as possible, fulfill the intension of the PCE

expression. The steps involved in generating the Google queries from a well formed PCE follow.

3.4 The Preference Prioritized Query Operator

Let A, B, and C represent propositions. Then the Prioritized Constraint Expression

$$A\ \mu B\ \mu 2C$$

means that A is required and B and C are preferred (but not required). Furthermore, the 'B' term is more preferred than the 'C' term. Recall that the priority of the preference operator is specified with an integer associated with the μ operator. As the integer gets larger the priority of the search word associated with the preference operator decreases. Thus μ1 has higher priority than does μ2. When implementing the preference constraint the symbol '&' is used to represent μ. The integers that specify priority are inferred using a positional priority in the search string. Thus, the highest priority search word appears to the far left of the search string and the lowest priority preference operator and its associated search word are to the far right. Thus, using a Conjunctive Normal Form with an infix notation and choosing the symbol '&" to represent the preference operator, this is written by the user for the PCE parser as:

$$A \,\&\, B \,\&\, C$$

The result table for the above conjunctive normal form preferential constraint is below.

A & B & C	T_0
A &B	T_1
A	T_2
F_0 Query (-A -B -C)	F_0

Table 6: CNF Preferential Constraint Truth Table

3.4.1 Conjunctive Normal Form (CNF)

This implementation of Prioritized Constraint Expressions uses a form that is similar to Conjunctive Normal Form (CNF) formulas found in Boolean logic.

> *A statement is in conjunctive normal form if it is a conjunction (sequence of ANDs) consisting of one or more conjuncts, each of which is a disjunction (OR) of one or more literals (i.e., statement letters and negations of statement letters [19].*

Following are examples of valid Conjunctive Normal form statements:

$$A \wedge B$$

$$(A \vee C) \wedge (B \vee C)$$

$$A \wedge (B \vee D) \wedge (B \vee E)$$

$$\neg A \wedge (B \vee C)$$

$$(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$$

Valid connectives in CNF are the $\wedge$ which represents the logical AND, the $\vee$ which

represents the logical OR, and the ¬ which represents a logical NOT (negation).

3.4.2 Preference Prioritized Query Operator as Google Queries

The Preference Prioritized Query Operator formulated and submitted by the user

A & B & C

would be sent to Google as four separate queries. Four queries are required because we need to partition our queries to create the illusion that Google understands PCE expressions when, in fact, it is the pre-Google code that makes sense out of the PCE expression and then forwards the appropriate number of well formed standard Google queries to the Google SOAP Search API.

Since the goal is to satisfy a user's need for relevance of information retrieved via the notions of preference and priority in the query propositions, the first query result set should return the most preferred results. Note also that Google only returns pages that include all of the terms in the query string, therefore there is an implicit boolean AND between terms.

> *By default, Google only returns pages that include all of your search terms. There is no need to include "and" between terms.* [20]

This is how the Google query for the most preferred result would look like:

A B C

Any query results from this would be marked as "T0" because they represent complete success in satisfying the user's query. With the standard Google query interface, if the most preferred query returns nothing then the user is left to manually re-query by changing the terms used and re-submitting their modified query. With a PCE query the user will be given "less preferred" results ordered by the preference term's implicit priority based on the left to right order or terms. The system will attempt to return multiple result sets ordered from the most preferred result set to the least preferred result set.

The second query would return the second-most preferred result set. The B term is μ1 and is a more preferred term than C which is μ2, therefore the search string is resubmitted with term C omitted. This is what the query sent to Google would look like.

A B -C

Any results from this query would be marked "T1" and represents a somewhat less preferred result. The reason for the "-C" is that Google will return pages that contain the terms A, B and C if a page just satisfies the A and B terms. In this case, the inclusion of pages with C is a coincidental result and would in fact be redundant given that the page containing A, B and C would appear in the T0 result set. Any page that would include A, B and C should rightly be only represented in the T0 result set and not appear in this T1 result set.

The the next Google query submitted would be:

A -B -C

Any result set would be labeled "T2" recognizing the exclusion of the μ1 term B and the μ2 term C and represents an even less preferred result set. Finally, the last Google query generated would be a negation of all the terms. The actual Google query would look like:

-A -B -C

This query represents total failure and is designated as the F0 query and result set. In theory the result set for F0 would contain results which do not match the users request whatsoever. However in practice Google does not allow you to submit a query with all terms negated with the "-" character. F0 queries and potential result sets represent a theoretical aspect of the infinitesimal framework presented here.

3.5 The Backup Prioritized Query Operator

The prioritized query operator ω specifies backup options. For example, the expression:

A ωB ω2C

means that we would like A or, failing that, we would settle for B or, failing that, then we would settle for C. The degree of ω specifies the priority for the backup options.

Similarly, the priority of a backup operator is specified with an integer associated with

the ω operator. As the integer gets larger the priority of the search word associated with the backup operator decreases. Thus ω1 has higher priority than does ω2. When implementing the backup constraint operator the symbol '|' is used to represent ω. Exactly like the preference operator, the integers that specify priority are implemented using the notion of positional priority within the search string. Thus, the highest priority backup operator and it's associated search word appear to the far left of the search string and the lowest priority backup operator and it's associated search word are to the far right of the search string. An infix notation Disjunctive Normal Form is used to implement the backup constraint operator and the symbol "|" is used to represent the backup operator ω. This is written by the user for the PCE parser as:

A | B | C

The result table for the disjunctive normal form of the backup constraint is also theoretically infinite therefore the following table is truncated for practical purposes. For a Disjunctive Normal Form (DNF) backup constraint prioritized constraint expression such as A | B | C, the following result table applies.

A	T_0
-A B	T_1
-A -B C	T_2
F_0 Query (-A -B -C)	F_0

Table 7: DNF Backup Constraint Truth Table

3.5.1 Disjunctive Normal Form (DNF)

As mentioned, this implementation of backup constraints is similar to Disjunctive Normal Form (DNF) formulas found in Boolean logic.

> *A statement is in disjunctive normal form if it is a disjunction (sequence of ORs) consisting of one or more disjuncts, each of which is a conjunction (AND) of one or more literals (i.e., statement letters and negations of statement letters) [19].*

These are examples of valid Disjunctive Normal form statements:

$$A \vee B$$

$$(A \wedge B) \vee C$$

$$(A \wedge B) \vee (\neg A \wedge C)$$

Like Conjunctive Normal Form, valid connectives in DNF are the $\wedge$ which represents the logical AND, the $\vee$ which represents the logical OR, and the $\neg$ which represents a logical NOT (negation).

3.5.2 Backup Prioritized Query Operator as Google Queries

The Backup prioritized query which is formulated as an infix DNF and submitted by the user as

A | B | C

would be sent to Google as three separate queries. Three queries are required in order to preserve the order of results as specified by the degree of ω. The first query would return the most preferred result set which is any pages containing A. The actual query is simply

A

If A returns a non-empty result set, we get total success, and the result set is marked as T0. Otherwise, if A fails to return a result set, but B returns results, we get partial success, T1. The second T1 query is for B and would look like:

-A B

If both A and B are fail to return a result set, but C returns a result, we get modest success, T2. And the third query is:

-A -B C

The use of an infix notation to represent the Priority Constraint Expression preference and backup operators offers a simple and quite elegant solution to encode the two notions of user preference, with regard to a list of search terms and the notion of priority of terms when compared to one another. The left to right positional priority of terms is clean and captures the spirit of Priority Constraint Expressions.

The next section deals with the software implementation of the ideas discussed in this

section. In particular, a parser and associated grammar are developed which will parse the infix notation Priority Constraint Expression and determine if it is a valid Priority Constraint Expression. From there the Priority Constraint Expression is processed to generate the standard Google queries which will approximate the intention of the Priority Constraint Expression.

4 Priority Constraint Expression Implementation

Implementing Priority Constraint Expressions on top of the Google Search engine requires preprocessing. This preprocessing is handled with Perl. [21] In order to handle Priority Constraint Expressions, a grammar which defines valid Priority Constraint Expressions is developed and used to generate a top down parser. The parser validates a submitted Priority Constraint Expression and then breaks the expression down into a ordered series of standard Google queries which reflects the preferences and priorities implicit in the Priority Constraint Expression. The formal grammar and the generated parser are described below.

4.1 Formal Specification of the PCE Grammar

The formal grammar of Priority Constraint Expressions is given in this specification using a notation similar to an Extended Backus-Naur Form (EBNF) notation. Each rule in the grammar defines one symbol, in the form

symbol: expression

Symbols are written in lowercase letter. The expression on the right-hand side of a rule may contain some combination of symbols or literal strings. Literal strings are single quoted. The "|" character indicates the delimiter for a symbol with multiple expressions.

```
searchstring:       expression eofile
expression:          andIfPossible|orFailingThat
 andIfPossible:     (term '&')(s) term
 orFailingThat:     (term '|')(s) term
 term:              paren|google(s)
paren:              '(' expression(s) ')'
google:             alphanumeric
alphanumeric:       /\w+/
eofile:             /^\z/
```

The symbols *alphanumeric* and *eofile* have expressions which use predefined subpatterns commonly found in regular expressions. In Perl [21], which is used to implement this grammar, the subpattern *w* is equivalent to [a-zA-Z0-9]. The + sign associated with *w* is a quantifier meaning one or more of the characters in [a-zA-Z0-9]. For the eofile symbol, the ^\z subpattern is a zero-width assertion since it does not deal with characters in a string, rather it will match the ending contents of a string, including any new lines . [22] The implementation of the formal grammar is done with a Perl module.

4.2 Parse::RecDescent

Parse::RecDescent [23] is a Perl module developed by Damian Conway. Parse::RecDescent generates top-down recursive-descent text parsers based on a grammar specified by the user. A Parse::RecDescent grammar is yacc-like which in turn is similar to Backus-Naur grammar specifications. The Parse::RecDescent grammar for

the Priority Constraint Expression query engine follows.

```
my $grammar = q{
        searchstring:       expression eofile
        expression:         andIfPossible|orFailingThat
        andIfPossible:      (term '&')(s) term
        orFailingThat:      (term '|')(s) term
        term:               paren|google(s)
        paren:              '(' expression(s) ')'
        google:             alphanumeric
        alphanumeric:       /\w+/
        eofile:             /^\z/
};
```

Parse::Recdescent, as a top down parser, behaves differently than bottom up parsers. Specifically, Parse::RecDescent does not seek to find the longest possible match to a token. It will indicate a successful match at the earliest point in the analysis of a token so long as it satisfies the specified grammar. Given this behavior, two aspects of constructing a Parse::RecDescent grammar are worth noting. First, an eofile nonterminal must be used to ensure that the parser will parse the whole input text. Second, the order of the productions is important. Typically, longer tokens are placed before shorter similar tokens to prevent the parser from assuming success while having only parsed part of a token. For example, consider the following production:

Example Production: 'long'|longer'

When the parser encountered the token 'longer' and applied this production rule it would indicate success having only parsed the first 4 characters of 'longer'. This would leave the remaining string 'er' for the parser to match against it's production rules and it might fail. The better production rule will have the longer token first:

Example Production: 'longer'|long'

4.3 Using a Parse::Recdescent parser object

In order to use the Perl module Parse::RecDescent the module must be installed and then the keyword `use` allows for the subsequent use of the module in the code. The system must then instantiate a new parser for the infix Priority Constraint Expression that is submitted. Once instantiated the parser object can be referred to by it's handler and it's methods and data structures can be used. The following code snippet contains the key statements required to use the module Parse::RecDescent.

```
use Parse::RecDescent;

my $parser = new Parse::RecDescent($grammar)
or die "parser generation failure";

my $parsedSearchString = ( $parser->searchstring($theSearchString) );

if ( Dumper($parsedSearchString) =~ /[undef]/ ) {
  print "This is not a valid PCE search string\n";
  print "Please try again.\n";
  exit;
}
```

The final `if` clause tests to see if the instantiated parser returned *undef* as a result of

parsing the search string variable $theSearchString. When the parser returns *undef* it means that the the submitted search string does not conform to the grammar which defines a valid infix notation Priority Constraint Expression.

5 Prioritized Queries and the World Wide Web

This section demonstrates the browser based interface to the Prioritized Constraint Expression Search Engine. The following figure is a screen shot of the initial interface. In the query field is the well formed PCE string: *logic & wadge & infinitesimal*. This PCE will require four Google queries.

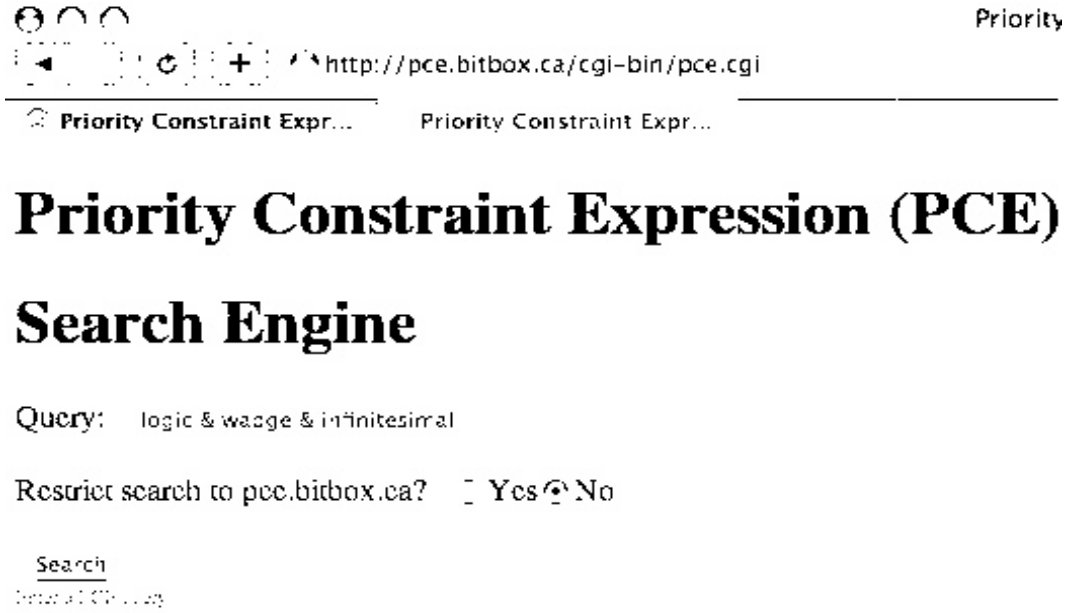

Illustration 1: PCE Search Engine Interface

The next screen shot shows that in fact the pce.cgi code generates and displays the four

required Google queries.

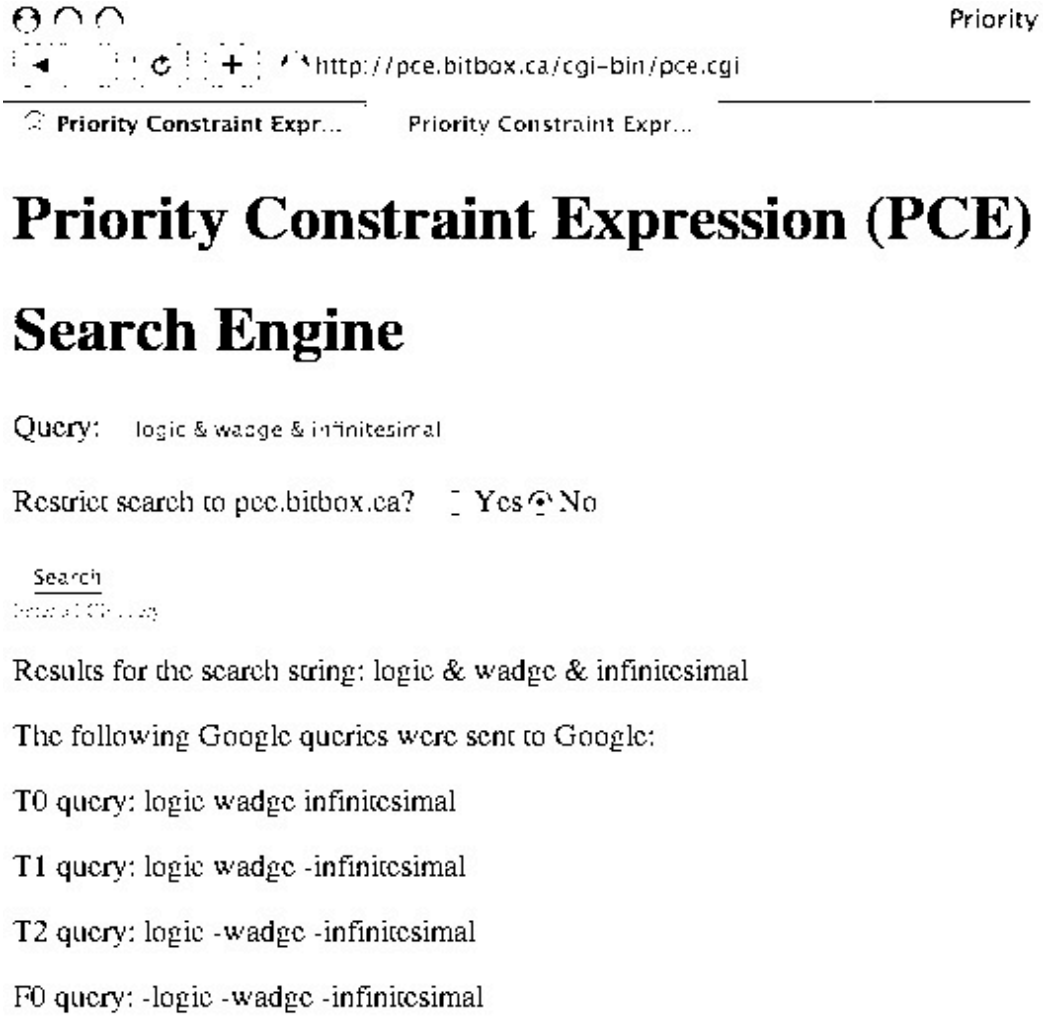

Illustration 2: PCE Search Engine Google Queries

The last screen shot and table show the partial and full result set for the PCE respectively. The results are grouped by truth value with the most preferred results at the top of the list along with a designation of T0 and the least preferred results are last and designated T2. Notice that the F0 query is generated by the system and in fact is

submitted to Google but as previously mentioned Google will not return results for a Google query containing all negated search words.

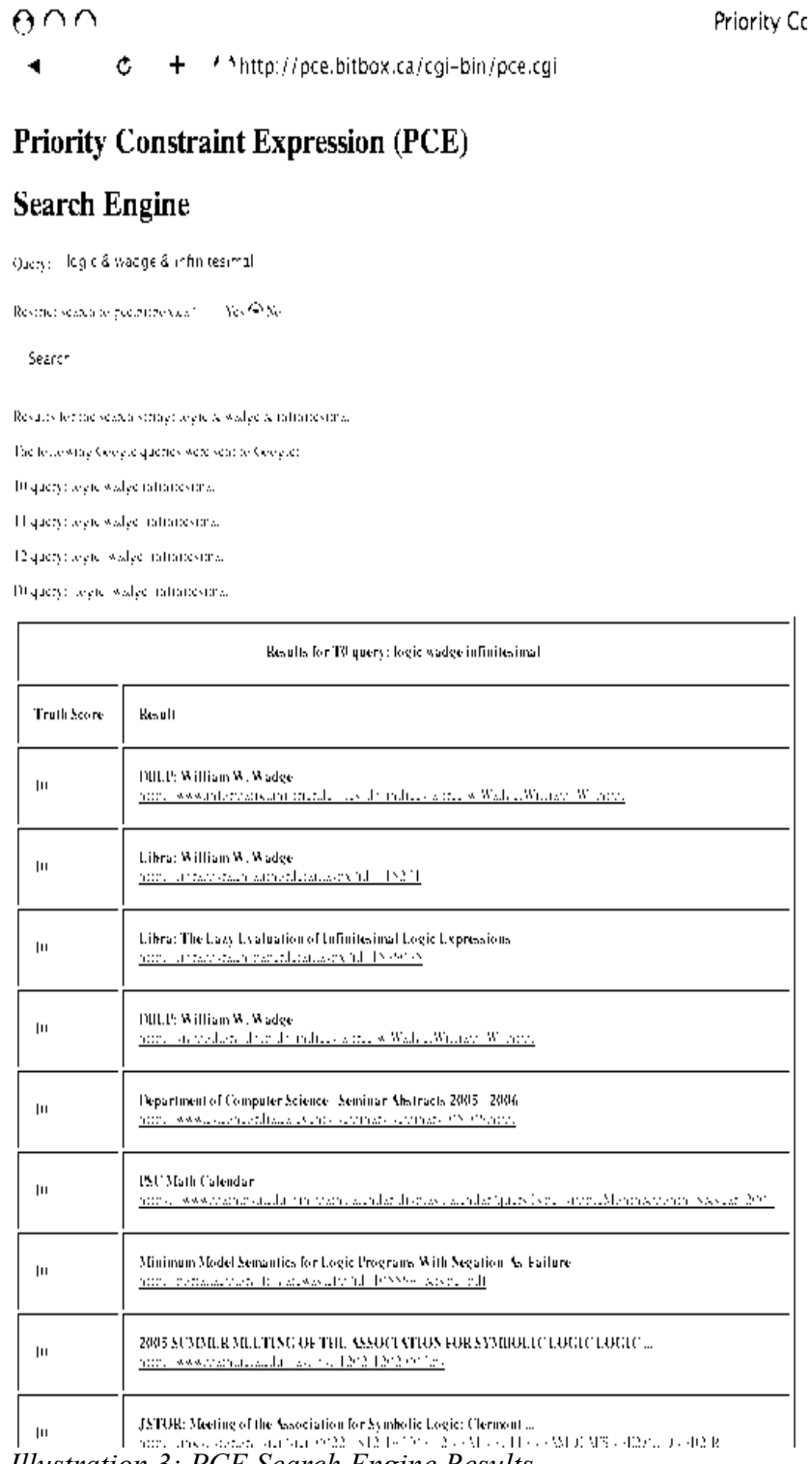

Illustration 3: PCE Search Engine Results

Results for T0 query: logic wadge infinitesimal

Truth Score	Result
T0	DBLP: William W. Wadge http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/w/Wadge:William_W=.html
T0	Libra: William W. Wadge http://libra.msra.cn/authordetail.aspx?id=415271
T0	Libra: The Lazy Evaluation of Infinitesimal Logic Expressions http://libra.msra.cn/paperdetail.aspx?id=1536035
T0	DBLP: William W. Wadge http://sigmod.org/dblp/db/indices/a-tree/w/Wadge:William_W=.html
T0	Department of Computer Science - Seminar Abstracts 2005 - 2006 http://www.cs.concordia.ca/events/seminars/seminars_05_06.html
T0	PSU Math Calendar https://www.math.psu.edu/bin/mathCalendar/displayCalendar?queryType=simpleMonth&month=8&year=2004
T0	Minimum Model Semantics for Logic Programs With Negation-As-Failure http://portal.acm.org/ft_gateway.cfm?id=1055694&type=pdf
T0	2005 SUMMER MEETING OF THE ASSOCIATION FOR SYMBOLIC LOGIC LOGIC ... http://www.math.ucla.edu/~asl/bsl/1202/1202-007.ps
T0	JSTOR: Meeting of the Association for Symbolic Logic: Clermont ... http://links.jstor.org/sici?sici=0022-4812(197703)42%3A1%3C113%3AMOTAFS%3E2.0.CO%3B2-R
T0	Publications of William W. Wadge http://dblife.cs.wisc.edu/dblp.cgi?type=author&entity=entity-23654

Results for T1 query: logic wadge -infinitesimal

Truth Score	Result
T1	Extending Temporal Logic Programming with Choice Predicates Non ... http://citeseer.ist.psu.edu/orgun94extending.html
T1	First-Order Functional Languages and Intensional Logic ... http://citeseer.ist.psu.edu/289.html
T1	Extending Temporal Logic Programming with Choice Predicates Non ... http://logcom.oxfordjournals.org/cgi/content/abstract/4/6/877?ck=nck
T1	Extending Temporal Logic Programming with Choice Predicates Non ... http://logcom.oxfordjournals.org/cgi/reprint/4/6/877
T1	eprintweb - Author Index Wadge, All archive http://eprintweb.org/S/authors/All/wa/Wadge/1/refs
T1	Minimum model semantics for logic programs with negation-as-failure http://portal.acm.org/citation.cfm?id=1055694
T1	Towards a unified theory of intensional logic programming http://portal.acm.org/citation.cfm?id=146091&dl=
T1	University of Athens - Department of Informatics and ... http://www.di.uoa.gr/en/research_publ.php?id=17
T1	JSTOR: Some Results in the Wadge Hierarchy of Borel Sets. http://links.jstor.org/sici?sici=0022-4812(199203)57%3A1%3C264%3ASRITWH%3E2.0.CO%3B2-7
T1	Panagiotis Rondogiannis http://cgi.di.uoa.gr/~prondo/conferences.html

Results for T2 query: logic -wadge -infinitesimal

Truth Score	Result
T2	Logic - Wikipedia, the free encyclopedia http://en.wikipedia.org/wiki/Logic
T2	Mathematical logic - Wikipedia, the free encyclopedia http://en.wikipedia.org/wiki/Mathematical_logic
T2	Apple - Logic http://www.apple.com/logicpro/
T2	Apple - Logic Express http://www.apple.com/logicexpress/
T2	Factasia Logic http://www.rbjones.com/rbjpub/logic/
T2	Aristotle's Logic (Stanford Encyclopedia of Philosophy) http://plato.stanford.edu/entries/aristotle-logic/
T2	Apple - Support - Emagic Legacy Product Support http://www.emagic.de/
T2	Atheism: Logic & Fallacies http://www.infidels.org/news/atheism/logic.html
T2	The World Wide Web Virtual Library:Logic Programming http://vl.fmnet.info/logic-prog/
Results for F0 query: -logic -wadge -infinitesimal	
Truth Score	Result

Table 8: Full Result Set for Example Query

6 The Google SOAP Search API

The implementation of Prioritized Constraint Expressions in this thesis relies on a stack of technology. Given that the Google SOAP Search API is implemented as a web service, that implies that the Prioritized Constraint Expression Search Engine would also be implemented on the web.

At its base the Prioritized Constraint Expression Search Engine uses an Apache [24] web server. The method of user interaction with the Prioritized Constraint Expression Search Engine is through the Common Gateway Interface (CGI) [25] which is a standard for interfacing external applications with information servers, such the Apache web server. In this case the information server is the Prioritized Constraint Expression Search Engine. The Prioritized Constraint Expression Search Engine is written in the Perl programming language. [26] Apache, Perl and many other freely available software owes a debt of gratitude to the GNU / Free Software Foundation. [27]

In 2002, Google released an Application Programming Interface (API) to allow software developers to programmatically interact with the Google Search Engine. The Google SOAP Search API is, as it's name implies, a SOAP based web service. SOAP [28] (Simple Object Access Protocol) is an XML [29] based mechanism for exchanging structured and typed information and to perform Remote Procedure Calls (RPC) usually over HTTP.

This code snippet shows how the Perl module SOAP::Lite is used to instantiate a Google search web service:

```
use SOAP::Lite;
my $google_search = SOAP::Lite->service("file:$google_wsdl");
```

Notice that the Google SOAP Search API instantiation has one parameter which is a reference to file called a WSDL file. The Google SOAP Search API is a SOAP based web service and therefore requires an associated WSDL file. A Web Service Description Language (WSDL) file gives a formal description of the programmatic interface of a web service. The Google Search WSDL file defines all the callable procedures for the Google SOAP Search API web service as well as definitions of the data structures that are used.

Within the GoogleSearch.wsdl the method declaration of most interest is the *doGoogleSearch* method. As its name implies this method is used to initiate a search on the Google index. In the Perl code of pce.cgi, this is how this method is invoked.

```
  my $results = $google_search -> doGoogleSearch ($google_key,
$query, 0, 10, "false", "", "false", "", "latin1", "latin1");
```

The 10 search parameters that must be provided to the *doGoogleSearch* method are listed, in order, in the table below:

Parameter	Description
key	Unique alphanumeric key given to each developer
q	Set of query terms that you wish to search on
start	Zero-based index of the first desired result
maxResults	Number of results desired per query. Maximum 10.
filter	Switch for filtering of similar results
restricts	Restriction by country or subject
safeSearch	Adult content filtering
lr	Language restriction
ie	Input encoding (deprecated and ignored)
oe	Output encoding (deprecated and ignored)

Table 9: Google Search Query Parameters

The most important parameters are: *key*, *q* and *maxResults*. The *q* parameter accepts the actual Google search string that will be submitted to the Google Index via the Google Search SOAP API. The *maxResults* limits the number of result items for the query. Notice the maximum is only 10. Also, the parameter called *key* uniquely identifies the developer issuing the search request and allows Google to enforce certain restrictions on the use of the Google SOAP Search API. These restrictions and limits are discussed later.

Having sent a search request via the *doGoogleSearch* method, a result of typed data is returned to the program. Of particular interest is the *GoogleSearchResult* complex type and its nested complex type *ResultElement*. Each result returned by the API is returned as a *GoogleSearchResult* which contains metadata about the search result and the actual result itself in the *ResultElement* type.

```
<xsd:complexType name="GoogleSearchResult">
<xsd:all>
<xsd:element name="documentFiltering"              type="xsd:boolean"/>
<xsd:element name="searchComments"                 type="xsd:string"/>
<xsd:element name="estimatedTotalResultsCount"     type="xsd:int"/>
<xsd:element name="estimateIsExact"                type="xsd:boolean"/>
<xsd:element name="resultElements"                 type="typens:ResultElementArray"/>
<xsd:element name="searchQuery"                    type="xsd:string"/>
<xsd:element name="startIndex"                     type="xsd:int"/>
<xsd:element name="endIndex"                       type="xsd:int"/>
<xsd:element name="searchTips"                     type="xsd:string"/>
<xsd:element name="directoryCategories"            type="typens:DirectoryCategoryArray"/>
<xsd:element name="searchTime”                     type="xsd:double"/>
</xsd:all>
</xsd:complexType>
```

Table 10: GoogleSearchResult Complex Type

```
<xsd:complexType name="ResultElement">
<xsd:all>
<xsd:element name="summary"                        type="xsd:string"/>
<xsd:element name="URL"                            type="xsd:string"/>
<xsd:element name="snippet"                        type="xsd:string"/>
<xsd:element name="title"                          type="xsd:string"/>
<xsd:element name="cachedSize"                     type="xsd:string"/>
<xsd:element name="relatedInformationPresent"      type="xsd:boolean"/>
<xsd:element name="hostName"                       type="xsd:string"/>
<xsd:element name="directoryCategory"              type="typens:DirectoryCategory"/>
<xsd:element name="directoryTitle"                 type="xsd:string"/>
</xsd:all>
</xsd:complexType>
```

Table 11: ResultElement Complex Type

The Prioritized Constraint Expression Search Engine Perl code (pce.cgi) pulls only two of the possible nine elements from the ResultElement complex type array and displays them as HTML[30]. The following code snippet shows how the system iterates through the returned result set and pulls the *URL* and *title* elements from the hash.

```
foreach my $result ( @{$googleResults} ) {
      my $url = $result->{URL};
      if ( $url =~ /index/ )
      {    # Don't print if it is the index.html file.
      }
      else {
         print Tr(
            td(
               [
                  $truthValueNumberPrefix . $truthValueNumber,
                  b( $result->{title} || 'no title' ) . br()
                    . a(
                      { href => $result->{URL} },
                      $result->{URL}
                    )
               ]
            )
         );
      }
  }
```

6.1 Limitations to the Google SOAP Search API

The Google SOAP Search API is a great tool for web developers that wish to have programmatic control over the search strings submitted to Google and the result sets that are returned from the Google index. There are however some limitations and restrictions on the implementation and use of the Google SOAP Search API. Some of these limitations are technical and some are legal.

6.1.1 Google Index 32 Word Limit

Although the Priority Constraint Expression Search Engine has no restrictions on the length of the search string in terms of the number of search words there is a underlying limit with the Google index itself. Prior to 2005 the number of search words that the Google index will accept was 10. Since that time the number of search words that the Google index will accept is 32. In the following screen shot notice the warning that all words are ignored after the 32 word limit is exceeded.

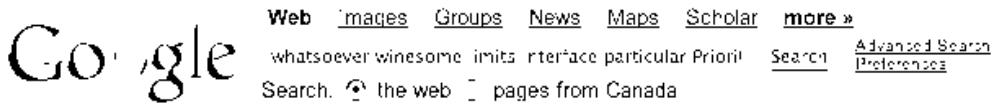

Illustration 1: Google 32 Word Limit

This of course means that the 32 search word maximum will limit the width/size of any Priority Constraint Expression.

6.1.2 Maximum Number of Results Returned

The Google SOAP Search API method called *doGoogleSearch* has a parameter called *maxresults*. This parameter allows the developer to limit the number of results returned by the web service. However, this parameter can only be set as high as 10. Thus, the largest number of results that can be returned per query is quite limited. In fact, this limit

really restricts the use of the Google SOAP Search API to a research/experimental API or, at best, a tool to create a “top 10” list web service.

6.1.3 One thousand queries per day limit per key

The last technical limit is the hard limit on the number of queries that can be submitted per day. The Google SOAP Search API method called *doGoogleSearch* has a parameter called *key*. This parameter submits the unique alphanumeric key given to each developer. This key is used to identify the source of the submitted query and then to enforce this limit of 1000 queries per day by key. The reasoning, given by Google, for the limit is that the Google SOAP Search API is experimental and therefore the resources allocated for supporting the system are limited. The limited resources were evident during the writing of this thesis, the Google SOAP API seemed to be not as robust as it could be. A common error returned was “ERROR 502: Bad Gateway”. The frequency which this error occurs is dependent on Google's load. This server load issue affected the likelihood of PCE query failure given that, in order to present a coherent set of results which fulfills the intent of the users PCE, all generated standard Google queries must succeed. The larger the number of PCE tokens in the Prioritized Constraint Expression the higher the chance of failure of one of the standard Google queries.

6.1.4 Restrictions on Commercial Usage

The licensing terms of use for the Google SOAP Search API state that the API cannot be used for commercial purposes. If a developer created a web service that was built on the Google SOAP Search API and that web service became very popular, they would not be able to turn it into a self sustaining (from a financial perspective) business because of this restriction.

6.1.5 Discontinued and Unsupported

In December 2006, during the creation of this thesis, Google announced that the Google SOAP Search API was deprecated. The company would no longer give out keys to developers and removed the software development kit (SDK) from the site. Any potential developers are now encouraged to use the Google AJAX API instead. However this AJAX API does not lend itself to programmatic control and pre or post processing of search strings or search results.

7 Usability of Prioritized Constraint Expressions on Google

This implementation of Prioritized Constraint Expressions over the Google Index via the Google SOAP Search API fulfills well what a PCE should do in terms of allowing a user to specify their preferences and priorities over the information retrieved while at the same time allowing for flexibility in the information retrieval process such that, in the event of failure of the most preferred results, there will be less preferred but possibly still usable results returned in subsequent result sets. The notion of building search strings up from tokens and joining those tokens with the connectives & (“and if possible”) or | (“or failing that”) provides a fail over mechanism in terms of retrieving more possibly relevant results while acknowledging that those additional results may be less preferred than the first query sent to Google. This is powerful because it moves beyond the standard Boolean connectives 'AND' and 'OR' that currently are available in Google queries. At the very least, the PCE provides a simplified tool to re-query the Google index with reasonable choices of contingency queries without the user having to reformulate and re-query over and over again.

Having said that, there is little hope for the Prioritized Constraint Expression Search Engine now that the Google SOAP Search API is deprecated. Therefore the Prioritized Constraint Expression Search Engine will not have a long life. There are, however, programmatic interfaces to Yahoo and to Live Search but these use a different API.

8 Conclusion

With search services available on the web today users are limited in terms of the expressiveness of their search queries. Search queries are simple Boolean constructs and if the query doesn't retrieve relevant results, as determined by the user outside of the relevance/page rank framework of the search service, the user is required to reformulate their query and try again. This reformulate/re-query process is time consuming and inefficient. In this thesis, we have implemented a method of applying priority constraint expressions to Google search. This allows a user to express their preferences and priorities for each token of a search string.

Priority Constraint Expressions are based on infinitesimal logic which supplies a logic-based framework for expressing a hierarchy of truth values which are then applied to the Priority Constraint Expressions. These priorities and preferences of the user are expressed using two constraint types: preferential constraints and backup constraints. With these two constraints, a user can setup a hierarchy among the tokens of a search string to express which terms are more preferred and which combination of terms are less preferred. The use of backup constraints also allows more flexibility in the user's search efforts rather than being forced into a linear boolean try and retry search process.

The reader may have noticed that we did not make full use of infinitesimal logic. We used only the standard false value F_0 , and only nonstandard true values T_k for small k. Nor did the intermediate 0 value play any role. In fact the extended query notation could be specified, at least informally, without reference to any formal logic at all.

This is actually very good news, because it means that our preferential query technology is accessible to users with no formal training (the vast majority). But it also raises the question, why bother with the logic in the first place?

The first answer is, that there is a difference between using the system and designing it. The current system was based on Agarwal's which developed directly out of the Rondogiannis-Wadge system. In theory one could have designed our Google extension from scratch, but that is not how it happened; and no one else seems to have had the idea.

The second point worth making is that it seems to be very difficult to *extend* the system without using formal logic. It seems only natural that users would want to nest preference operations; for example

search preferences & ((google & rankings) | yahoo)

to look for pages about searching and preferences, preferably about google (especially dealing with google rankings) or at least Yahoo. It is not entirely clear what is meant by these and more complicated nested expression. However, using infinitesimal logic we can investigate this systematically, by looking at ways to extend the truth tables for & and | to cover cases when the operands themselves are nonstandard truth values. The general rule is still to display the queries in decreasing order of their values.

Bibliography

[1] R. Agarwal, “A Framework for Expressing Prioritized Constraints Using Infinitesimal Logic”. Master's Thesis, University of Victoria, Victoria, British Columbia, 2005.

[2] P. Rondogiannis and W. W. Wadge, “Minimum modelsemantics for logic programs with negation-as-failure,” ACM Transactions on Computational Logic, 2005.

[3] Smith, Robin, "Aristotle's Logic", *The Stanford Encyclopedia of Philosophy (Fall 2004 Edition)*, Edward N. Zalta (ed.), URL = <http://plato.stanford.edu/archives/fall2004/entries/aristotle-logic/>.

[4] Gabbay, Dov, “Classical vs non-classical logic”. Handbook of Logic in Artificial Intelligence and Logic Programming, volume 2, Oxford University Press. 1994

[5] Rose, Lynn E. (1968). Aristotle's Syllogistic, Springfield: Charles C Thomas Publisher.

[6] Chellas, Brian F. (1980). Modal Logic: an introduction., Cambridge: Cambridge University Press.

[7] (2007, Mar.) Multi-valued Logic - Wikpedia, the free encyclopedia. [Online]. Available:http://en.wikipedia.org/wiki/Multi-valued_logic

[8] Stephen C. Kleene. *Introduction to Metamathematics*. van Nostrand, Princeton, 1952.

[9] Gottwald, Siegfried, "Many-Valued Logic", *The Stanford Encyclopedia of Philosophy (Winter 2006 Edition)*, Edward N. Zalta (ed.), URL = <http://plato.stanford.edu/archives/win2006/entries/logic-manyvalued/>.

[10] G. Brewka, S. Benferhat, and D. Le Berre. Qualitative choice logic. In *Proc. Principles of Knowledge Representation and Reasoning, KR-02*. Morgan Kaufmann, 2002. available as IfI-Report under www.informatik.uni-leipzig.de/brewka.

[11] G. Brewka. Logic programming with ordered disjunction. In *Proc. AAAI-02*. Margan Kaufmann, 2002.

[12] Bush, V. "As we may think", Atlantic Monthly, 176, 1945, p. 101—108.

[13] (2003) Information retrieval and computer science: an evolving relationship evaluation – ACM Digital Library. [Online]. Available: http://doi.acm.org/10.1145/860435.860437

[14] (1979) Information Retrieval [Online]. Available: http://www.dcs.gla.ac.uk/~iain/keith/index.htm

[15] (1998) The anatomy of a Large-Scale Hypertextual Web Search Engine [Online]. Available: http://infolab.stanford.edu/~backrub/google.html

[16] (2005) Information retrieval based on historical data [Online]. Available: http://appft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=/netahtml/PTO/search-bool.html&r=1&f=G&l=50&co1=AND&d=PG01&s1=20050071741&OS=20050071741&RS=20050071741

[17] (1998) The PageRank Citation Ranking: Bringing order to the Web [Online]. Available: http://infolab.stanford.edu/~backrub/pageranksub.ps

[18] Hajek, Petr, "Fuzzy Logic", *The Stanford Encyclopedia of Philosophy (Fall 2 006 Edition)*, Edward N. Zalta (ed.), URL = <http://plato.stanford.edu/archives/fall2006/entries/logic-fuzzy/>.

[19] Mendelson, E. Introduction to Mathematical Logic, 4th ed. London: Chapman & Hall, 1997.

[20] (2007) Google Help: Basics of Search [Online]. Available: http://www.google.ca/intl/en/help/basics.html#and

[21] The Perl Directory. http://www.perl.org/.

[22] C. Frenz. Pro Perl Parsing. Apress, 2560 Ninth Street, Suite 219, Berkeley,CA 94710, USA, 2005.

[23] Damian Conway. The Parse::RecDescent Perl module. http://search.cpan.org/~dconway/Parse-RecDescent/

[24] The Apache Group. The Apache HTTP Server Project. http://www.apache.org/.

[25] NCSA HTTPd Development Team. The Common Gateway Interface. http://hoohoo.ncsa.uiuc.edu/cgi/.

[26] L. Wall. Programming Perl 5. UNIX Programming Tools. O'Reilly and Associates, 101 Morris St., Sebastopol CA, 95472, USA, January 1996.

[27] GNU / The Free Software Foundation. http://www.gnu.org

[28] World Wide Web Consortium. Simple Object Access Protocol(SOAP) 1.1, 2000.http://www.w3.org/TR/SOAP

[29] World Wide Web Consortium. Extensible Markup Language(XML) 1.0, 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

[30] World Wide Web Consortium. HTML 4.0 Specification, 1998. http://www.w3.org/TR/1999/REC-html401-19991224.

UNIVERSITY OF VICTORIA PARTIAL COPYRIGHTLICENSE

Title of Thesis:

An Implementation of Prioritized Queries using the Google Search API

Author _____

David Kerins

July 3, 2007