Relaxation in Web Search: A New Paradigm for Search by Boolean Queries

Parke Godfrey
Army Research Laboratory
Adelphi, Maryland, U.S.A.
godfrey@arl.mil
and
Department of Computer Science
University of Maryland, College Park

Abstract

Search by boolean queries suffers from a paradox of precision: if the query is too general (with respect to the corpus), the response is an avalanche; if the query is too specific, the response is empty. It is difficult to cast a query balanced on this scale of specificity that retrieves a reasonable number of items. Nowhere is this more evident, perhaps, than in search on the World Wide Web.

For this reason, many Web search engines employ information retrieval (IR) techniques other than boolean search. Indeed, this problem of specificity is a key reason, in general, that other IR techniques have been researched. For instance, AltaVista's standard interface employs an IR scoring technique across the document list to rank-order a result list. (AltaVista's "advanced" interface is for boolean search.) However, the IR searches have problems too. Quite often, IR search queries result in avalanches. If the IR scoring is not "natural" for the query at hand, the documents-of-interest may be buried in the results, never to be found.

The utility of boolean queries could be saved if a good method to recover from over-specified queries were evident. In this paper, we work to devise such a method. We show how boolean queries can be relaxed *automatically, by throwing away just as many keywords as necessary for the query to succeed. With this relaxation facility, a new search paradigm becomes possible: ask a specific query; it will be relaxed to the points of success to find the closest related documents.*

1. Introduction

Search and retrieval over large collections is plagued by a fundamental problem of specificity. If one's query is too general, the result is an avalanche. So many items are returned, it is pragmatically impossible to sift through them. However, if one's query is too specific, the result is empty. No items are returned. This is useless as well, as one has accomplished nothing in one's search for items of interest.

One has recourse with a too-general query. One can refine the query to make it more specific, so perhaps to return a manageable set of items. This refinement process can be quite cumbersome, however. Furthermore, it often happens that one refines the query too far, into a too-specific query that returns nothing. The case with too-specific queries is worse. There is no standard recourse. If a too-specific query occurs during one's attempt at query refinement, one can back up and try another variation. In essence, the only recourse is to give up and to begin again.

As collections (*corpa*) grow, this problem of specificity worsens. More queries move into the status of too general as they start resulting in avalanche responses. To counter this problem, one must ask more specific queries. However, this strategy often incurs queries that are too specific, resulting in the empty response. Since there is no recovery from such *failing* queries, search by query over the collection becomes decreasingly effective. This trend has been perhaps nowhere more evident recently than on the World Wide Web.

We propose a way to address this problem of specificity, by providing a recovery method for too-specific queries. When a query fails, it can be automatically *relaxed* to result in a query that succeeds. This can be accomplished by revising the query by removing just as many conditions as necessary (or, say, *keywords* in a keyword boolean search on the Web) so that the revised query succeeds.

In this paper, our focus is on boolean queries. With a boolean query, an item from the collection simply either matches or it does not. So the result of a boolean query is the *set* of items that it matches. More specifically, we focus on *conjunctive* queries. That is, the query's primary structure is a list of conditions that are conjoined together. Thus, any item that matches the query must satisfy each of the query's conditions. We focus on the domain of the World Wide Web, and the search by query that the Web search engines provide. Of course, this work applies equally well to any information retrieval domain and corpus.

In Section 2, we discuss further our domain of interest and provide background on this idea and related cooperative answering work. In Section 3, we motivate and formalize the relaxation paradigm. In Section 4, we show how the relaxation paradigm can be implemented, and we consider the algorithmic complexity involved. We discuss a new dynamic programming search algorithm that is suited to finding relaxations. That the relaxation paradigm has not been implemented before is, at least in part, because of computational complexity issues that had not been properly resolved. In Section 5, we consider the design of a Web search engine relaxation facility that could provide query relaxation for web search queries. In Section 6, we conclude.

2. Domain and Background

We focus on the World Wide Web as our target domain in this paper, but our interest is any large collection that is available "on-line" for which boolean search is useful. Online library systems offer automated query search to help to identify relevant documents, and then to help to locate them. Digital libraries further afford immediate access to documents, and the opportunity to search across the *content* of documents as well as over their catalog information. Indeed, the World Wide Web is akin to a digital library in this sense (although its catalog information, sometimes called *metadata*, is quite deficient).

The digital age has made search by query a viable, and increasingly indispensable, tool There is pressure for better search strategies and methodologies. As examples, other relevant domains today include the following. Lexis-Nexis is a company that offers on-line access to legal, news, and business documents. They purpose to have a combined corpus of one billion documents, to which they add nine and a half million documents each week. (Thus Lexis-Nexis's domain is substantially larger than the Web currently.) Lexis-Nexis offers on-line search services over its corpa, and handles more than 300,000 searches per day [21]. Woods [28, 29] describes work at SUN Corporation to provide a sophisticated search engine (and search paradigm) for searches across UNIX manual pages. It is estimated that NASA's Earth Observing System project will generate a tera-byte of data a day at its peak [26]. It is commonplace to have a query search facility for file systems. *Glimpse* is a tool that provides an inverted index facility for file systems with a boolean search capacity [23]. *WebGlimpse* meanwhile is a project to combine browsing and searching over the Web by defining *neighborhoods* of given documents for browsing [22].

We focus on boolean queries in this paper. We should note, however, that many Web search engines do not provide a boolean query interface as their standard facility. Instead, they process keyword queries by other *information retrieval* (IR) techniques. For instance, AltaVista's standard interface is an IR engine, whereas their "advanced" interface is boolean [1]. IR methods *score* the documents of the corpus (through judicious use of indexes) with respect to the query, and sort the results to return a ranked list of hits, ordered by best score to worst. (Usually there is a cut-off below some threshold score.)

Information retrieval techniques offer an advantage in that they *order* the results by *relevance* (the scores). Second, they seemingly appear to be immune to the problem of being too-specific (thus failing) that boolean queries can suffer. Indeed, IR techniques were researched and developed, in part, to address this very problem. If the query does not "match" closely any documents, the return list will still start with whatever the most relevant hits are (according to the scoring function). So results are always returned.

However, IR engines do incur the avalanche problem. They tend to return huge results. As before, as corpa grow, so do the results of queries. There is no easy way to escape the avalanche problem with IR techniques. Adding keywords to a query (hence, making it more "specific") can only *improve* candidate documents' scores. Thus, the avalanche does not go away, but potentially worsens as the query is "refined". New keywords do serve to reorder the previous results by adjusting their scores, but they may also bring other documents above the threshold.

The scoring functions of IR techniques are heuristic at best. A scoring function a given IR engine employs may not be well tuned to what the user intends in a given query. For instance, documents that mention the words *query* and *optimization* many times may score highly for the query $\{$*semantic*, *query*, *optimization*$\}$, even when they do not mention *semantic*. However, since *semantic query optimization* is a specific type of query optimization technique for databases, these other documents are probably not of interest to the user. While IR engines might work well for one given query, for another, the true items of interest may be well buried in the returned list. This is effectively again the avalanche problem.

So it appears that IR techniques only offer an illusion of solving the specificity problem. They may often be the search engine technique of choice simply because the over-specificity problem can be so devastating for boolean queries. It would be interesting to see how boolean and IR techniques would fare if the specificity problem could be adequately addressed. This is our intent with the relaxation paradigm.

Keyword queries—whether they are interpreted as boolean (conjunctive) queries or as IR queries—are not ideal. For instance, one might use the word *car* in one's query, and a relevant document uses the word *automobile* (but not *car*). It would be better if we could query with concepts instead of keywords. Woods [28, 29] seeks to address issues synonyms and word choice. The Web search engine Excite operates by a type of concept matching rather than keyword matching [8]. (They perform cluster analysis on the web to create automatically clusters of words that then represent "concepts".) AltaVista's *Live Topics* likewise provides conceptual clustering [1]. Other work seeks to capture the linguistic content of documents so one might query over the semantic content of documents, not necessarily over just their words. For instance, one might ask for documents that discuss new information retrieval techniques and that compare them with existing techniques. This work is important. In it, people are attempting to address numerous other problems in high quality information retrieval. These techniques, however, are beyond our scope.

Starting in the the late 1970's and in the 1980's, people pursued query relaxation for database queries. (The focus was more on *false presuppositions*, which is the dual problem.) Any sub-query—the query with some of its *conditions* eliminated—can be called a presupposition of the query itself. The sub-query must evaluate *true* (that is, it results in some answers) if the query is to evaluate *true*. When a query evaluates *false* (that is, it results in no answers), it is worthwhile to identify presuppositions (sub-queries) that evaluate *false*. The dual is to find those sub-queries which succeed.

Strawson [27] noted the importance of presuppositions in logical formulas, and questioned whether one can meaningfully assign a truth value to a formula which has a false presupposition. Belnap and Steel [2] demonstrated the relevance of false presuppositions for information systems. In [20], Lee developed a CODASYL database system that could detect and presents false presuppositions to database queries. Kaplan [18, 19] likewise noted the relevance of false presuppositions to databases, and built a natural language query system for a CODASYL database to identify false presuppositions to users. Colmerauer and Pique addressed the problem of false presuppositions in their work to translate natural language queries into a logical formalism [4]. Janas [16] was the first to study the computational feasibility of reporting the smallest sub-queries that fail, and studied queries in a relational setting.

Corella et al. [5] considered finding the relaxations of conjunctive boolean queries for library searches. (We discuss this in the next section.) Motro [24, 25] extended on the notion of false presuppositions and relaxation: instead of eliminating *conditions* from a query, they can be replaced by *more general* conditions. Gal and Minker [12, 13] considered false presuppositions in coordination with *misconceptions*, which arise when a query *must* fail with respect to the database's semantics. Gaasterland [9] further considered coordination issues. (See [15] for more background, and [11] for more background on cooperative query answering in general.)

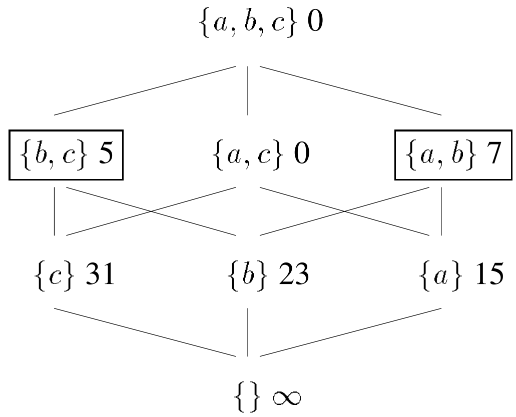

Figure 1. Lattice of sub-queries of a three keyword query.

Interest has seemed to wane in query relaxation since. Despite the previous work, no relaxation facility exists in any commercial database system, or information retrieval system, today. This is most likely due to issues of computational complexity that had heretofore been unresolved. In [15], we analyzed the computational complexity of query relaxation, and based on the complexity profile, developed the first efficient algorithm to compute relaxations. There has been no work on relaxation in information retrieval systems, aside from [5], to our knowledge.

3. The Query Relaxation Paradigm

First, let us define a simple model of corpa and queries.

Definition 1. Define a *document* to be a set of words. Let the *world*, or *corpus*, $\mathcal{W}$ be a collection of documents. Likewise, define a *query* to be a set of words. The words appearing in a query are often referred to as its *keywords*.

A query $\mathcal{Q}$'s *evaluation* $[\![\mathcal{Q}]\!]$ with respect to $\mathcal{W}$ is the set of documents from $\mathcal{W}$, each of which contains $\mathcal{Q}$. That is,

$$[\![\mathcal{Q}]\!] \quad = \quad \{\mathcal{D} \in \mathcal{W} \mid \mathcal{Q} \subseteq \mathcal{D}\}$$

We call each $\mathcal{D}$ in $[\![\mathcal{Q}]\!]$ a *hit* of $\mathcal{Q}$.

We say a query *fails iff* $[\![\mathcal{Q}]\!] = \emptyset$, and that it *succeeds* otherwise. Note by definition that the empty query succeeds; in fact, $[\![\emptyset]\!] = \mathcal{W}$.

Queries defined above act simply as conjunctive boolean queries. Now, it is a simple matter to define the *relaxations* of a query, with respect to the corpus.

	prol.	**lang.**	**sys.**	**comp.**	**surv.**	**rat.**	**stat.**	**#hits**	**#probes**	Σ
1.	prol.	lang.	sys.	comp.	surv.		stat.	3	8	8
2.	prol.	lang.	sys.			rat.		5	4	12
3.	prol.		sys.	comp.	surv.	rat.		1	7	19
4.		lang.	sys.	comp.	surv.	rat.	stat.	1	7	26
								10	0	26

(a) The relaxations.

	prol.	**lang.**	**sys.**	**comp.**	**surv.**	**rat.**	**stat.**
1.	prol.	lang.		comp.		rat.	
2.	prol.	lang.			surv.	rat.	
3.	prol.					rat.	stat.

(b) The minimal failing sub-queries.

Figure 2. The relaxations and minimal failing sub-queries of a given seven keyword query.

Definition 2. Call any query $\mathcal{S}$ a *succeeding sub-query of* query $\mathcal{Q}$ *iff* $\mathcal{S} \subseteq \mathcal{Q}$ and $\llbracket \mathcal{S} \rrbracket \neq \emptyset$. Call $\mathcal{S}$ a *maximal succeeding sub-query of* query $\mathcal{Q}$ *iff* there is no other succeeding sub-query $\mathcal{R}$ of $\mathcal{Q}$ such that $\mathcal{S} \subset \mathcal{R} \subseteq \mathcal{Q}$. Let us more simply call any maximal succeeding sub-query of $\mathcal{Q}$ a *relaxation* of $\mathcal{Q}$.

Likewise call any query $\mathcal{S}$ a *failing sub-query of* $\mathcal{Q}$ *iff* $\mathcal{S} \subseteq \mathcal{Q}$ and $\llbracket \mathcal{S} \rrbracket = \emptyset$. We define $\mathcal{S}$ to be a *minimal failing sub-query of* $\mathcal{Q}$ *iff* there is no other failing sub-query $\mathcal{R}$ of $\mathcal{Q}$ such that $\mathcal{R} \subset \mathcal{S}$.

Let us assume that the corpus $\mathcal{W}$ is not empty. Then a query always has a relaxation. At one extreme, if the query succeeds, it has one relaxation: itself. At the other extreme, it has one relaxation: $\emptyset$. A succeeding query $\mathcal{Q}$ has no failing sub-queries. A failing query $\mathcal{Q}$ must, however. In the extreme, the query itself is its one minimal failing sub-query. Note that the number of hits is *monotonic* with respect to query ordering: a sub-query must have more or the same number of hits as the query.

Figure 1 shows the lattice of sub-qeuries of a three keyword query, $\{a, b, c\}$. Each sub-query has been marked with a hypothetical number of hits that it produces. In [5], Corella, Kaplan, Wiederhold, and Yesil propose a system for an on-line library catalog which would present for a (failing) query an annotated lattice as in Figure 1. This way, the user immediately sees which sub-queries succeed, and which fail. The user also sees how big the result set is of each sub-query. This lets the user make an informed choice about which query (probably one of the succeeding sub-queries in the presented lattice) with which to proceed.

This approach is promising, but only for quite short queries (up to, say, five keywords). With even slightly longer queries, we run into problems quickly. This is, of course, because the number of sub-queries goes up exponentially with respect to the length of the query (for instance, $2^7 \approx 100$, $2^{10} \approx 1,000$, and $2^{20} \approx 1,000,000$). So by seven keywords, presentation is already difficult. It is pragmatically difficult at ten keywords, and impossible on today's displays at twenty keywords. Even if we could reasonably display such results (say for up to ten keyword queries), people could not reasonably assimilate the results. And, as we noted earlier, longer queries are needed for effective search as corpa grow.

The lattice display approach as in Figure 1 has a second problem as well. Each sub-query must be evaluated in order to annotate it with its number of hits. This expense is substantial for longer queries, and intractable generally as queries grow. We want a technique that would be much less expensive, on average, to evaluate.

The relaxation paradigm suggests that we just identify the relaxations of the query to the user. The results of the entire lattice are not necessary. The query $\{a, b, c\}$ in Figure 1 has two relaxations, which are boxed in the Figure: $\{b, c\}$ and $\{a, b\}$. (It has a single minimal failing sub-query, $\{a, c\}$.)

Figure 2 (a) shows the relaxations of a seven keyword query asked to AltaVista's advanced query interface as a conjunctive query. The query was $\mathcal{Q}$: $\{$*prolog, language, system, comparison, survey, rating, statistics*$\}$. It and its relaxations were evaluated against AltaVista on 7 February 1997. The column **#hits** shows the number of hits each relaxation produced when evaluated. (Ignore the columns **#probes** and Σ and the last row for now. We discuss these

	que.	**dat.**	**het.**	**opt.**	**log.**	**age.**	**Sta.**	**web**	**alg.**	**mul.**	**#hits**	**#probes**	Σ
1.	que.	dat.	het.	opt.	log.	age.	Sta.	web	alg.		1	11	11
2.	que.	dat.	het.	opt.	log.	age.	Sta.		alg.	mul.	2	3	14
3.	que.	dat.	het.	opt.	log.	age.		web	alg.	mul.	2	2	16
4.	que.	dat.	het.	opt.	log.		Sta.	web	alg.	mul.	4	2	18
5.	que.	dat.	het.	opt.		age.	Sta.	web	alg.	mul.	2	2	20
6.	que.	dat.	het.		log.	age.	Sta.	web		mul.	6	2	22
7.	que.	dat.		opt.	log.	age.	Sta.	web	alg.	mul.	1	2	24
8.			het.		log.	age.	Sta.	web	alg.	mul.	1	6	30
											19	0	30

Figure 3. The relaxations of a given ten keyword query.

later in Section 4 when we develop an algorithm for query relaxation.)

Figure 2 (b) shows the minimal failing sub-queries of $\mathcal{Q}$. This collection is the dual of the relaxations. Each of the three sub-queries in Figure 2 (b) evaluate to empty. Thus, for any follow-up query to succeed, one must eliminate at least one keyword from each of the three minimal failing sub-queries.

One might have found relaxations 1 and 4 in Figure 2 (a) by hand, since they involve only eliminating a single keyword. However, it is doubtful that anyone would find 2 or 3 by hand, which require eliminating three and two keywords, respectively. Relaxation 2 might certainly be of interest to the user. One might argue Relaxation 4 is the least interesting, since it eliminated the "first" keyword, *prolog*, which seems to be the topic. (We could consider the set of keywords in the query as an *ordered* set, with the order denoting the preference of the keywords in the query.)

Another benefit of relaxation is that the results of the relaxations of a query are necessarily disjoint. In other words, they cannot share hits. So the four relaxations in Figure 2 (a) must denote ten hits in all. For example, let $\mathcal{R}_1$ and $\mathcal{R}_2$ represent relaxations 1 and 2, respectively. We know that $[\![\mathcal{R}_1]\!] \cap [\![\mathcal{R}_2]\!] = \emptyset$. To demonstrate this, note that $\mathcal{R}_1$ contains the keyword *comparison*, while $\mathcal{R}_2$ does not. So every document in $[\![\mathcal{R}_1]\!]$ contains the word *comparison*, by definition. However, no document in $[\![\mathcal{R}_2]\!]$ can contain the word *comparison*. If there were such a document, then $\mathcal{R}_2 \cup \{comparison\}$ would be a succeeding sub-query, which contradicts our assertion that $\mathcal{R}_2$ is a maximal succeeding sub-query (that is, a relaxation). Since any two relaxations are necessarily incomparable sets, then their result sets are necessarily disjoint.

A second example is shown in Figure 3. This is a ten keyword query, $\mathcal{Q}$: $\{$*query, database, heterogeneous, optimization, logic, agent, Stanford, web, algorithm, multimedia*$\}$, asked to AltaVista's advanced query interface as a conjunctive query on 7 August 1997. This query had eight relaxations. Again, relaxations 6 and 8 would be hard to find by hand as they each require the elimination of more than one keyword. In this case, the query's relaxations spanned nineteen hits.

Note that the number of hits of each relaxation in both examples from Figures 2 and 3 are all quite small. Let us compare the number of hits of some non-maximal succeeding sub-queries with the relaxations in Figure 3: sub-query $\{$*query, …, agent*$\}$, the first six keywords of $\mathcal{Q}$, had 42 hits; $\{$*query, …, logic*$\}$, 512; $\{$*query, …, optimization*$\}$, 852; $\{$*query, …, heterogeneous*$\}$, 2,866; $\{$*query, database*$\}$, 76,659; and $\{$*query*$\}$, 269,906. This is another advantage of relaxation. Relaxed queries will tend to return small sets of hits. This is because relaxed queries are maximal succeeding sub-queries. All super-queries of them—that are also sub-queries of the original—must be failing queries. So the relaxations are the most specific sub-queries that still have hits. Therefore, the relaxation process is a method that automatically finds the balance between too general and too specific.

4. Relaxation Search

To find relaxations of a failed query, we can (somehow judiciously, we hope) choose sub-queries to *test* until we have identified the maximal succeeding sub-queries. Testing a sub-query is merely asking that query to the search engine to see whether it returns hits (succeeds). Our search space is the lattice of all the query's sub-queries. This approach is a generate-and-test strategy. Let us call a sub-query to be tested a *probe*. A query relaxation facility would ask probes to search for relaxations. A user would not be aware of this background work, except for the time required. A key objective for us is to minimize the number of probes needed to find a relaxation, several relaxations, or, perhaps, all the relaxations. (We learn that the cost of evaluating the probes in practice greatly outweighs the cost of computing the search.)

To find a single relaxation of a query is fairly simple and fairly inexpensive. Given the query consists of k keywords (or, more generally, conditions), a relaxation can be found in k probes. Algorithm 1 sketches how this is done. The search for relaxations is a type of abductive inference. It was known prior that to find a single minimal abductive set (for an abductive problem class that is analogous to our relaxation problem) was within $\mathcal{O}(k^2)$ [3]. We showed that this could be tightened to $\mathcal{O}(k)$—indeed, to within k steps—in [15].

Algorithm 1 can also find the *best* relaxation, with respect to the following criteria. Let $w(K)$ be the *weight* of keyword K in query $\mathcal{Q}$. Let a relaxation $\mathcal{R}$'s score be $\sum_{K \in \mathcal{R}} w(K)$. Let $K_1, \ldots, K_n$ be the order of the keywords in our query. If we weigh keywords such that $w(K_i) > \sum_{j=i+1}^{n} w(K_j)$, for $i \in \{1, \ldots, n\}$ (that is, each keyword weighs more than all that follow, combined), we can ensure finding the best relaxation. Simply choose the highest weighing keyword from $\mathcal{S}$ each time in step 3a.

To find all relaxations is inherently hard. This is simply because there may be just too many of them. Consider the possibility that all sub-queries of length $\lfloor k/2 \rfloor$ are relaxations. This means there can be an exponential number of them, with respect to the length of the query (k).

To find *some* relaxations, however, is an interesting task. A question then is how difficult is it to find relaxations sequentially, or, that is, to enumerate them. In [15], we studied this case for finding relaxations of *relational* queries. (Let us call the queries we consider in this paper *keyword queries*, to distinguish.) Surprisingly perhaps, to find k relaxations of a relational query (of length k conditions—joins and selects) is an **NP**-*complete* problem. Thus, to find a single relaxation is inherently simple, but to find a few (k) is inherently hard.

This complexity result for relational queries does not necessarily transfer to keyword queries, although we shall show it remains relevant. The reason is that the corpus must be considered as an input parameter in addition to the query.[1] Thus it might be possible to analyze the corpus somehow to determine the relaxations, instead of employing a generate-and-test strategy as in Algorithm 1. However, the corpa we consider are exceedingly large. Many Web search engines index tens of millions of documents, and Lexis-Nexis indexes over a billion. There can be no simple IR-style scoring and sorting that could result in the relaxations, such as IR techniques employ.[2] This means that any direct approach is likely to be complicated. Since the corpus is very large, such an approach is likely to be infeasible. This means that it is reasonable for us to pursue a generate-and-test approach, even considering the intractability of enumerating too many relaxations.

1. $\mathcal{S} := \mathcal{Q}$
2. $\mathcal{R} := \{\}$
3. **While** $\mathcal{S} \neq \{\}$
    a. **Choose** $K \in \mathcal{S}$
       $\mathcal{S} := \mathcal{S} - \{K\}$
    b. **Evaluate** $\mathcal{R} \cup \{K\}$.
       If it evaluates non-empty **then**
          $\mathcal{R} := \mathcal{R} \cup \{K\}$
4. **Return** $\mathcal{R}$

Algorithm 1. Algorithm to Find a Relaxation.

Algorithms that people had proposed in the literature to find a number of relaxations (or, dually, MFSs) have been intractable for any enumeration. One natural approach is a depth-first recursive search. (Let us call this the naive algorithm.) On analysis, it has exponential worst-case for even finding two relaxations. (The naive algorithm can be shown to find the first relaxation within k^2 steps [15].) The problem arises because the algorithm treates the search space as a tree (the query and its sub-queries, recursively materialized). However, the search space is not a tree but is a directed acyclic graph (a boolean lattice). Thus, the search space is highly redundant. The problem that the naive algorithm has on searching for a second relaxation is that it may re-discover the first relaxation an exponential number of times before it finds the second. (Worse, we can show that this will usually happen.)

We want an algorithm that minimizes the number of probes needed and which enumerates the relaxations *ideally*; that is, it always finds the first in k steps (as does Algorithm 1), a second in $2k$ steps (this can be shown to always be possible), a third within k^2 steps (this is worst-case), and so forth, only decaying towards intractability as is inevitable.[3]

In [15], we developed a dynamic-programming algorithm for searching a lattice to enumerate minimals, or maximals, with respect to a monotonic test. We call it ISHMAEL for iterative search for minimal (or maximal) elements in a lattice. ISHMAEL has the following properties: it

- Optimally enumerates relaxations.
  - Finds the first in $k+1$ probes.
  - Slowly degrades as more relaxations are found.

- Never performs a redundant probe.
  - Never asks the same probe twice.
  - If a query was asked and evaluated empty, no super-query will then be asked.
  - If a query was asked and evaluated non-empty, no sub-query will then be asked.
- Exploits the decomposability of relaxation search.

[1] For the relational case, it is possible to show that there are sufficiently small relational databases—on the order of the size of the query—for which the problem is still *hard*.

[2] Any ordering of sub-queries that puts relaxations first depends non-monotonically on the contents of the corpus.

[3] It is inevitable, assuming that **NP** $\neq$ **P**, and that there is no better approach for relaxing *keywords* queries than by probing.

(All minimal failing sub-queries of a failing sub-query are also minimal failing sub-queries of the query.)
- It finds the relaxations in order (as discussed above for Algorithm 1).
- Gracefully halts.
  If the query has n relaxations, ISHMAEL finds that there are *no more* in the same time it would take to find a $(n + 1)$*st* relaxation.

Again, k is the number of conjuncts in the query. (We unfortunately cannot sketch the ISHMAEL algorithm here. Please see [15] for this work.)

5. Designing a Query Relaxation Facility for the Web

The algorithms from the previous section make it possible to build a practical relaxation facility for Web search engines. We have prototyped such a facility that we call *Query Focus*. It currently interacts with AltaVista's advanced interface [1]. Any search engine offering boolean search could be front-ended. The examples in Figures 2 and 3 were generated by *Query Focus*.

The column **#probes** in Figures 2 and 3 shows the number of probes required between relaxations to find each.[4] (The column Σ keeps a running sum.) AltaVista offers a mode that just returns the number of hits (and not the URLs of the hits themselves). *Query Focus* uses this mode for its probes. Probes average about a second a piece. The last row in each table of Figures 2 and 3 show how many further probes were necessary to discover that there were no more relaxations. In both cases, no further probes were necessary, but this is not always the case.

It is remarkable how few probes were needed in the relaxation searches in Figures 2 and 3. Many other examples we have run have been well behaved. In practice, relaxation search might not be expensive. If so, relaxation may be a viable tool.

In the design of *Query Focus*, we return one relaxation to the user at a time. This amortizes the search cost. Since the user may be content with the results of the relaxation, or first several relaxations, other relaxations may not be needed. This avoids the more expensive search that the generation of further relaxations requires.

An incremental version of ISHMAEL is ultimately needed for *Query Focus* (so it can be restarted each time to find a successive relaxation). This is not a practical problem as the time the search algorithm spends (compared with the time for the probes) is insignificant.

[4] Note that the first relaxation in each case took $k + 1$ probes. The extra probe tested the query itself to see that it failed. Otherwise, no relaxation search would be needed.

Currently, *Query Focus* is run at its own HTTP site. It interacts with AltaVista (or, potentially, any other Web search engine site or several) itself; clients interact with our site. This does incur network overhead for *Query Focus*'s probes (and the overhead of how the probes' results are packaged, which is not insignificant). This could, of course, be eliminated if a query relaxation facility such as *Query Focus* were integrated with a search engine at the same site. This could greatly speed up relaxation generation. However, it is interesting that a practical query relaxation facility is feasible which is not directly coupled with a search engine, and which can be run as a gateway (at a separate site), as *Query Focus* is currently.

6. Conclusions

We hope to field *Query Focus* soon to the public. Realistic use of relaxation on Web queries will provide us with empirical results on *Query Focus*'s performance. It is also important to see whether people will find the relaxation paradigm useful, and whether they will adapt to it. A majority of Web search queries are known to be single keyword queries. It is unclear how willing people are to refine their queries by adding many more keywords. Relaxation changes the equation by assuring that specific queries always provide results. Whether this is sufficient so people will find composing long queries useful remains to be seen.

Relaxation can be extended for different types of queries, besides just keyword queries. Queries may consist of any conditions to be met. For instance, "*semantic* **near** *optimization*" might be one condition. How to integrate better relaxation with all the boolean operators, as well as the "non-monotonic" or "extra-logical" operators such as **near** that search engines provide, should be investigated. The relaxation paradigm can be easily extended too to allow for wider application. For instance, a query need not fail to be relaxed. Any query can be relaxed to the point at which *new* hits are generated. There is much work to design a full-fledged relaxation facility.

There is much longterm research that can follow as well. How relaxation can be integrated with information retrieval techniques and rank-ordered results is a question worth study. Eventually, relaxation techniques can be integrated with conceptual query facilities. In particular, relaxation techniques that replace query conditions by more general conditions, instead of just eliminating conditions, as was suggested by Motro should be explored. For instance, the concept *automobile* could be relaxed to *vehicle*.

Cooperative techniques such as relaxation may come to play a viable role in information retrieval systems and database systems. Indeed, such techniques may become indispensable as our information sources continue to grow in both size and complexity.

References

[1] Digital's AltaVista web search engine. Web site. URL = http://www.altavista.digital.com/.

[2] N. D. Belnap and T. B. Steel. *The Logic of Questions and Answers*. Yale University Press, New Haven, CT, 1976.

[3] T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson. The computational complexity of abduction. *Artificial Intelligence*, 49:25–60, 1991.

[4] A. Colmerauer and J. F. Pique. About natural logic. In Gallaire et al. [14], pages 343–365.

[5] F. Corella, S. J. Kaplan, G. Wiederhold, and L. Yesil. Cooperative responses to boolean queries. In *First International Conference on Data Engineering*, pages 77–85, Silver Spring, Maryland, 1984. IEEE Computer Society Press.

[6] V. Dahl and P. Saint-Dizier, editors. *Natural Language Understanding and Logic Programming*. North Holland, 1988.

[7] R. Demolombe and T. Imielinski, editors. *Nonstandard Queries and Nonstandard Answers*. Studies in Logic and Computation 3. Clarendon Press, Oxford, 1994.

[8] Excite web search engine. Web site. URL = http://www.excite.com/.

[9] T. Gaasterland. *Cooperative Answers for Database Queries*. PhD thesis, University of Maryland, Department of Computer Science, College Park, 1992.

[10] T. Gaasterland, P. Godfrey, and J. Minker. An overview of cooperative answering. *Journal of Intelligent Information Systems*, 1(2):123–157, 1992.

[11] T. Gaasterland, P. Godfrey, and J. Minker. An overview of cooperative answering. In Demolombe and Imielinski [7], chapter 1, pages 1–40. Appears orginally as [10].

[12] A. Gal and J. Minker. A natural language database interface that provides cooperative answers. *Proceedings of the Second Conference on Artificial Intelligence Applications*, December 11–13 1985.

[13] A. Gal and J. Minker. Informative and cooperative answers in databases using integrity constraints. In Dahl and Saint-Dizier [6], pages 277–300.

[14] H. Gallaire, J. Minker, and J.-M. Nicolas, editors. *Advances in Database Theory, Volume 1*. Plenum Press, New York, 1981.

[15] P. Godfrey. Minimization in cooperative response to failing database queries. *International Journal of Cooperative Information Systems (IJCIS)*, 6(2):95–149, June 1997.

[16] J. M. Janas. On the feasibility of informative answers. In Gallaire et al. [14], pages 397–414.

[17] A. Joshi, B. Webber, and I. Sag, editors. *Elements of Discourse Understanding*. Cambridge University Press, 1981.

[18] S. J. Kaplan. Appropriate responses to inappropriate questions. In Joshi et al. [17], pages 127–144.

[19] S. J. Kaplan. Cooperative responses from a portable natural language query system. *Artificial Intelligence*, 19(2):165–187, Oct. 1982.

[20] R. M. Lee. Conversational aspects of database interactions. In *Proceedings of the 4*th *International Conference on Very Large Data Bases*, Berlin, 1978.

[21] Lexis-nexis. Web site. URL = http://www.lexis-nexis.com/lncc/about/background.html.

[22] U. Manber, M. Smith, and B. Gopal. WebGlimpse—combining browsing and searching. In *Proceedings of the Usenix Technical Conference*, Los Angeles, California, Jan. 1997.

[23] U. Manber and S. Wu. GLIMPSE: A tool to search through entire file systems. Technical Report TR 93-34, Department of Computer Science, The University of Arizona, Tucson, Arizona, Oct. 1993.

[24] A. Motro. SEAVE: A mechanism for verifying user presuppositions in query systems. *ACM Transactions on Office Information Systems*, 4(4):312–330, October 1986.

[25] A. Motro. FLEX: A tolerant and cooperative user interface to databases. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):231–246, June 1990.

[26] NASA's earth observing system (EOS). Web site. URL = http://eospso.gsfc.nasa.gov/.

[27] P. F. Strawson. *Introduction to Logical Theory*. Methuen, London, 1974.

[28] W. A. Woods. Beyond ignorance-based systems (abstract). In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, page 646, Bonn, Germany, May 1994. Morgan Kaufmann.

[29] W. A. Woods. Beyond search engines. Talk at University of Maryland Institute for Advanced Studies Colloquium Series, Sept. 1997.