Improving Retrieval System Performance with a Domain Algebra

Diploma thesis submitted to the
Swiss Federal Institute of Technology, Zurich

by

Guido Hoss

August 17, 1989

Direction: H.P. Frei
Supervised by: Peter Schäuble

Table of Contents

1. Introduction 1

2. Program Structure 3

2.1 Program Call Format 3
2.2 Document Collection Parsing 3
2.3 Initial Domain Algebra Generation 4
2.4 Sign Weight Calculation 5
2.5 Determination of Atomic Concepts 5
2.6 Inverted Concept/Document List Generation 7
2.7 Initial Atomic Weight Calculation 7
2.8 RSV Calculation 8
2.9 Generation of Preference Relations 8
2.10 Atomic Weight Optimization (Original Problem) 10
2.11 Atomic Weight Optimization (Reduced Problem) 11
2.12 Precision/Recall Graph Generation 11
2.13 Various Utilities 12

3. Data File Formats 13

4. Experimental Results 17

4.1 The CACM Domain Algebra 17
4.2 CACM Partial Optimization 18
4.3 The CISI Domain Algebra 21
4.4 CISI Stopword/Situation Evaluation 24
4.5 Department Library Catalog 25

5. Summary and Conclusions 27

Appendix A. File Directory Contents 28

References 29

1. Introduction

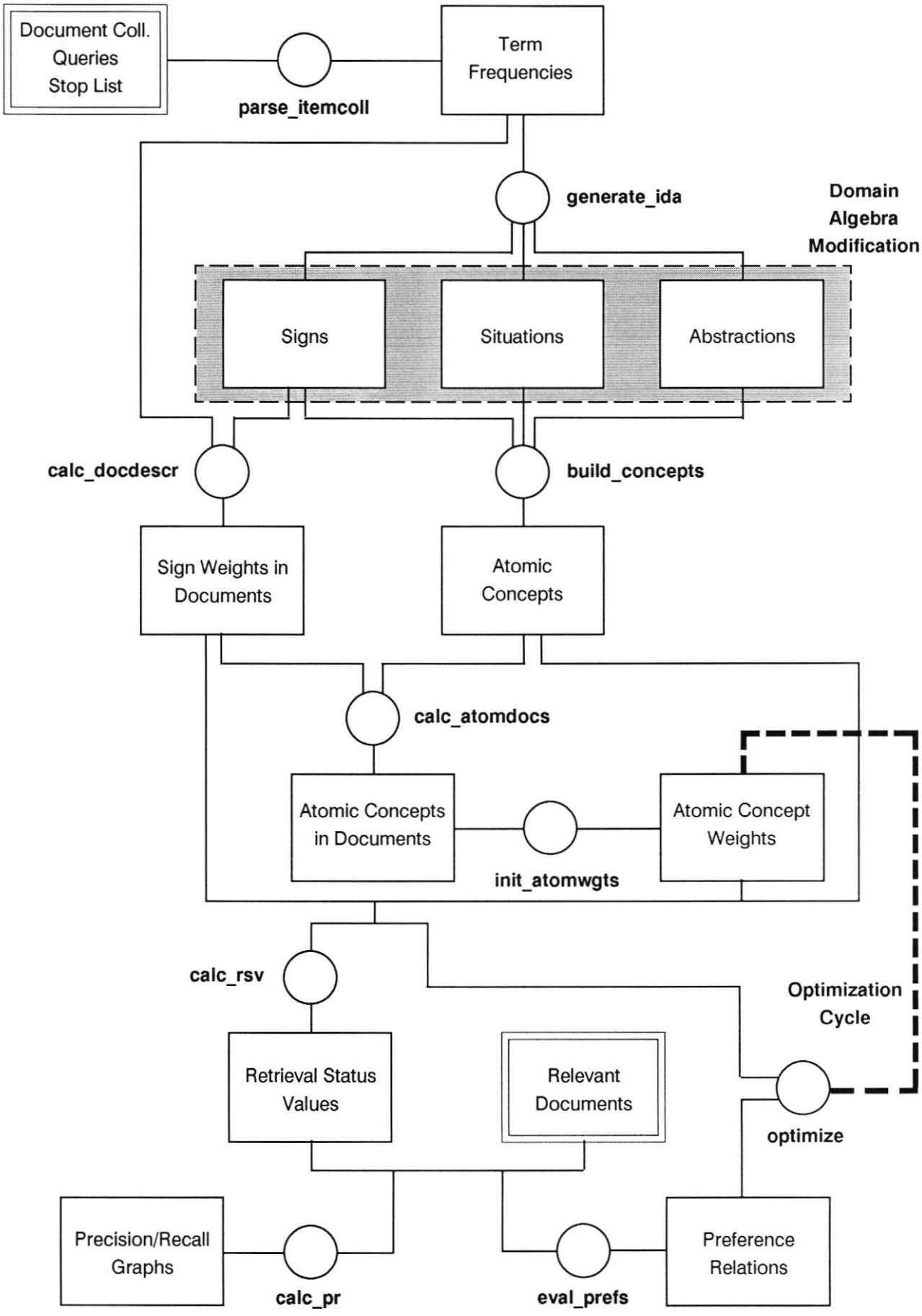
In his recently published dissertation [1], Peter Schäuble defines the mathematical background and a construction process for a new information structure, called *domain algebra*, which is supposed to improve the effectiveness of an information retrieval system. While it was shown that the construction process always yields an improvement, the magnitude of the improvement and therefore the usefulness of the method was yet unknown. The subject of this diploma thesis is the actual implementation and evaluation of the construction process.

During the four-month period of this work, an experimental system was developed. This systems consists of a number of programs which are invoked sequentially and communicate via intermediate data files in human-readable format. Each program corresponds to one step in the construction process. This approach was chosen to allow easy verification of the correctness of intermediate results. The fact that the programs were not bound together in one huge application rendered expansion and modification of the retrieval system painless. All programs were written in **C** on SUN UNIX workstations. The **C** language was chosen for its sophisticated input/output library, which was a must for a file-intensive application like this.

In the first part of this work, technical implementation details are given and the formats of the inter-program data files are defined. Although graphical representations of the construction process and the program data flow are included, familiarity with the construction process ([1], chapter 4) is assumed.

In the second part, the results of experiments performed on three test document collections are presented. The CACM/CISI collections serve as a "standard" to compare results to earlier retrieval methods. On the other hand, the department library catalog is of interest because of its multilingual nature. Finally, conclusions are drawn and an outlook on possible areas of improvement and further development is given.

Domain Algebra Construction: File Setup



2. Program Structure

2.1 Program Call Format

Basically, all programs read their input from files named on the command line, and write results to the standard output. The standard output can be redirected to a file by the user. Status messages are displayed on the standard error output. The format of input and output files is defined in the chapter "*Data File Formats*".

If a program name is entered without any parameters, a short help message describing the call format appears. During long file reads, the programs display running counts on the terminal. This display can be turned off (during batch execution, for example) by specifying the string `QUIET` (or `Q`) as the last parameter on the program's command line.

Unless otherwise noted, the call format of the programs follows directly from the data flow diagrams in each section; for example,

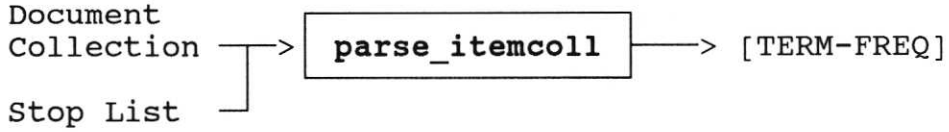
implies the call format

init_atomdocs *doc-descr concepts* `[QUIET]` > *atom-docs*

UNIX **make** files [5] have been written to facilitate the processes of building the domain algebra and compiling modified sources.

2.2 Document Collection Parsing

The document parser reads a document collection and creates a list of terms and their frequencies in the collection.



parse_itemcoll *docfile queryfile stoplist* `[QUIET]` > *term-freq*

The stop list is a text file which simply contains words separated by white space. The filenames of the stop list files are listed in the chapter "*Experimental Results*".

The document and query collections are assumed to be in the SMART format and are processed as follows:

1) Each document begins with an **.I** header and a unique, nonzero document number. Since the generated *term-freq* file contains terms of queries and documents, all documents in *queryfile* receive negative numbers.

2) Text following **.T** (title), **.W** (abstract), and **.A** (author) headers is scanned; all other text is ignored. Keywords (following the **.K** header) are deliberately discarded since they impose an information structure on their own and thus interfere with the effects of a domain algebra.

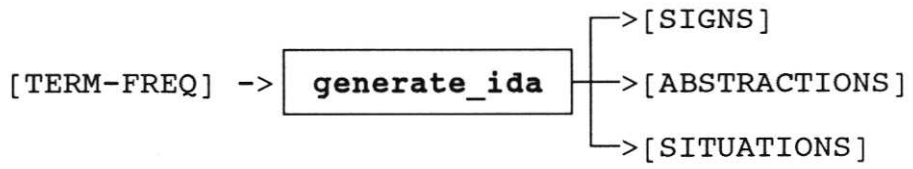
3) A *word* is a nonempty sequence of letters in the range A..Z and underscores ("_"). Each word in the text is run against the stoplist specified on the program command line. The remaining words are converted to uppercase and are stemmed by a modified version of Porter's algorithm [4]. We only performed plural-to-singular conversion since the full algorithm introduced unwanted term dependencies (example: *optimized* and *optimal* are both reduced to *optim*). The conversion rules are

```
SSES  ->  SS        IES  ->  Y
SS    ->  SS        S    ->  null
```

4) In addition to single word terms, multiword terms are generated from pairs of adjacent words which are not separated by a stopword. A more elaborate implementation would consider a term extraction algorithm similar to the one described in [3]; however, we wanted to keep the term space as compact as possible.

5) The terms in each document and query are output along with their respective term frequency.

2.3 Initial Domain Algebra Generation



generate_ida *term-freq* `[QUIET]` > *signs*
cp `/dev/null` *abstractions*
cp `/dev/null` *situations*

The initial domain algebra consists only of a set of independent, non-polysemic single word terms. No term dependencies exist, so the files containing situations and abstractions are empty.

1) The *term-freq* file is read; multiword terms are discarded. A list of all unique single word terms is built in memory.

2) Each single word term in the memory list is output along with a unique, positive ID number.

2.4 Sign Weight Calculation

Since all signs are regarded as non-polysemic, the function $wgt_d(t)$ ([1], p.67) is used to calculate the sign weights. The calculation is done in two passes:

1) Read *signs* and build a memory list of signs and associated terms.

2) Read *term-freq* and calculate the inverse document frequency *idf* for all signs in the list (first pass).

3) Read *term-freq* again, this time calculating the weight of each sign in each document and query (second pass).

4) For each document: only output signs with nonzero weights.

2.5 Determination of Atomic Concepts

As situations and abstractions are specified in the domain algebra, dependencies between terms are expressed as intersections of corresponding term concepts. The program **build_concepts** determines the structure of the concept space.

1) A list of atomic concepts is built for each sign. Each atomic concept is identified by a unique number. The lists consist of such numbers as well as *references* to other signs. If the list of a sign *s* contains a reference *r(s')* to a sign *s'*, all atomic concepts of *s'* also belong to *s*. References may be recursive and cyclic; the latter case occurs if several signs are synonyms.

   At the start, an initial atomic concept α_i is associated with each sign s_i.

2) A situation is defined by four signs <*agn*, *obj*, *iag*, *iob*>. As the set of concepts *c(iag)* is a subset of *c(agn)*, a reference *r(iag)* is added to the

atomic concept list of *agn*. Likewise, a reference *r*(*iob*) is added to the list of *obj*. Finally, a new atomic concept α_s is added to the lists of *iag* and *iob*. α_s represents the intersection $c(agn) \cap c(obj)$. This process is repeated for each situation in the *situations* file.

3) An abstraction is defined by the tuple <*spec, gen*>. Since all atomic concepts of the specific term *spec* also belong to the concept set of the general term *gen*, a reference *r*(*spec*) is added to the list of *gen*. This process is repeated for each abstraction in the *abstractions* file.

4) The contents of each atomic concept list are output. Two situations must be accounted for: a) if a reference is encountered, the atomic concepts of the referenced sign must be output recursively; and b) if the algorithm runs into a cycle during such a recursion, then the initial atomic concept numbers of all signs which belong to the cycle must be set to the same value. This is because the concept sets of synonyms are identical.

The algorithm is illustrated by the following example. The domain algebra *D* initially consists of 3 term signs, 2 interacting agent/object signs, 1 situation, and 1 abstraction. In step 1), one initial atomic concept is associated with each sign.

0	MONITOR	α_0
1	COLOR MONITOR	α_1
2	BITMAP	α_2
3	IAG_1	α_3
4	IOB_1	α_4

sit_1 = (MONITOR, BITMAP, IAG_1, IOB_1)
abs_1 = (COLOR MONITOR, MONITOR)

In step 2), the situation is processed. The interacting agent/object are subsets of the corresponding agent and object; their intersection is the atomic concept α_5.

0	MONITOR	α_0 , **ref**$_3$
1	COLOR MONITOR	α_1
2	BITMAP	α_2 , **ref**$_4$
3	IAG_1	α_3 , α_5
4	IOB_1	α_4 , α_5

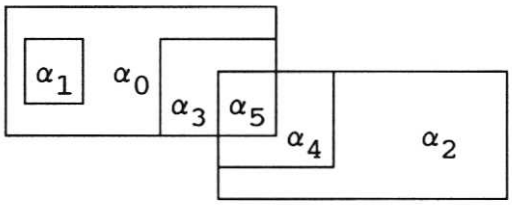
The abstraction is handled in step 3). The concepts of *color monitor* are a subset of the concepts of *monitor*.

0	MONITOR	α_0 , **ref**$_3$, **ref**$_1$
1	COLOR MONITOR	α_1
2	BITMAP	α_2 , **ref**$_4$
3	IAG_1	α_3 , α_5
4	IOB_1	α_4 , α_5

In step 4), references are resolved. The final output is as follows:

0	MONITOR	α_0 , α_3 , α_5 , α_1
1	COLOR MONITOR	α_1
2	BITMAP	α_2 , α_4 , α_5
3	IAG_1	α_3 , α_5
4	IOB_1	α_4 , α_5

which is consistent with the graphical interpretation of this domain algebra:



Atomic concept numbers are nonzero and positive with two exceptions: intersections of concepts and initial atomic concepts of signs with negative index are identified by negative numbers. This feature was implemented for the second series of tests (stopword and situation evaluation).

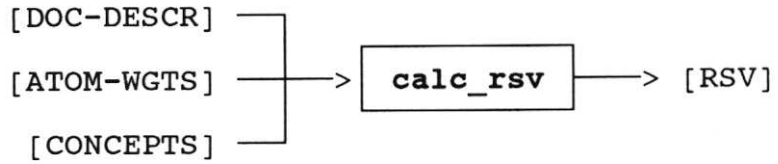
2.6 Inverted Concept/Document List Generation

This program processes the relations (**document #**, sign #) in the *doc-descr* file and (**sign #**, atom #) in the *concepts* file to produce the new relation (**atom #**, document #).

2.7 Initial Atomic Weight Calculation

For each atomic concept α, the squared inverse document frequency $IDF^2(\alpha)$ is calculated according to [1], p. 76. As the number of documents $|D|$ is specified explicitly in *atom-docs*, the calculation is straightforward.

2.8 RSV Calculation



The **calc_rsv** program calculates the value of the similarity function ([1], p. 62) for each query/document pair. The function, as defined in the thesis, is not normalized, so RSV values may exceed 1 if signs contain more than one atomic concept. For optimized weights, RSV values are in the range

$$c_1 * RSV(q,q) \leq RSV(q,d) \leq c_2 * RSV(q,q)$$

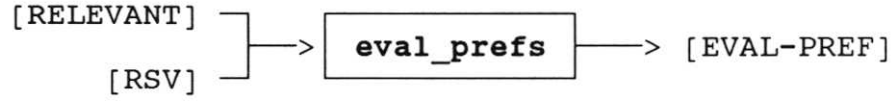
for all $q,d \in D$.

1) The information in *doc-descr* and *concepts* is used to build a list of documents. For each document d, a list of atomic concepts which occur in d is constructed. For each concept α, the weight $wgt_d(\alpha)$ is calculated.

2) The atomic concept weights in *atom-wgts* are read into memory.

3) The value of $RSV(q,d)$, based on common atomic concepts in q and d, is calculated for each query/document pair. To conserve disk space, zero RSV values are not output.

2.9 Generation of Preference Relations



Preference relations appear in the constraint equations of the optimization problem ([1], p. 78). Since the amount of relations, especially satisfied ones, can become very large, the following test is introduced.

The change of an atomic weight in the optimization process is limited by the constants c_1 and c_2 ([1], p. 77). Therefore, preference relations exist which will always remain satisfied resp. unsatisfied regardless of the new weight distribution after the optimization. In the optimization problem given in [1], these relations represent constraints which are always fulfilled; therefore, they can be omitted.

1) To turn an unsatisfied preference $d' <_q^- d''$ into a satisfied one, it is necessary to decrease the weights of concepts common to q and d' while increasing the weights of concepts common to q and d''. It is possible to convert an unsatisfied preference if

$$c_1 * RSV(q,d') - c_2 * RSV(q,d'') < -\epsilon$$

Note that from this condition it does not follow that the preference will actually be converted.

2) Likewise, a previously satisfied preference can become unsatisfied if

$$c_2 * RSV(q,d') - c_1 * RSV(q,d'') \geq -\epsilon$$

3) Although a number of unsatisfied preferences are eliminated in step 1), *all* $<^-$ preferences contribute to the cost function. Preferences which fail to satisfy 1) are output with a **C** tag.

The **calc_rsv** program does not output zero RSV values. Therefore, **eval_prefs** must determine relevant documents with zero RSV values because they can appear in unsatisfied preferences. It is not necessary to consider non-relevant documents with zero RSV values.

In the CACM/CISI collections, documents are either relevant or non-relevant to a given query. With respect to the department library catalog tests, this implementation also supports the notion of *relevance levels*. For a given query, R_i denotes the set of documents with relevance level i. A document d is more relevant than a document d' if $d \in R_i$, $d' \in R_j$, and $i > j$. Documents in R_0 are non-relevant. In the case of the CACM/CISI collections, all relevant documents are in R_1.

It is not necessary to generate preferences between documents in all relevance classes. In addition to the conditions defined before, it is sufficient to check each relevance class R_i ($i \geq 0$) and consider pairs of documents $<d, d'>$, where $d \in R_i$ and d' belongs to the next higher non-empty relevance class.

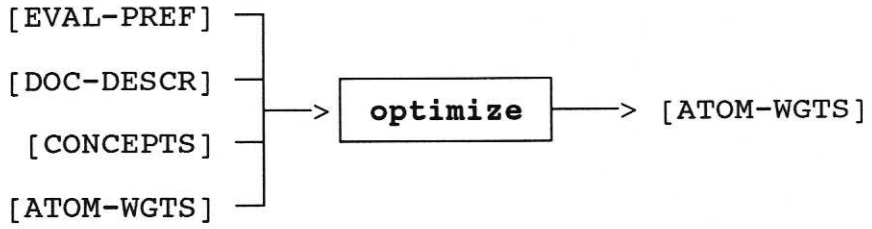
By default, **eval_prefs** produces preference relations for all queries involved. If the UNIX environment variable `QUERY` is specified, preference relations are produced for a particular query only. For example,

```
setenv QUERY 5
eval_prefs RELEVANT RSV QUIET > EVAL_PREF
```

generates preferences for query 5 only.

Since equations involving preference relations are part of the original optimization problem only, **eval_prefs** need not be executed if the reduced version of the optimization program is used.

2.10 Atomic Weight Optimization (Original Problem)



The optimization problem ([1], p. 78) for n atomic concept weights consists of a cost function, m preference relations, and $2n$ lower/upper bound constraints. In addition, n translation equations are necessary to transform the origin of the coordinate system to the point $(IDF^2(\alpha_0),...,IDF^2(\alpha_{n-1}))$. All equations and the cost function are stored in a dynamically allocated matrix structure of the form

```
TYPE Matrix = ARRAY [ 0 .. t-1 ] OF POINTER TO Row;
     Row    = ARRAY [ 0 .. n-1 ] OF REAL;
```

where $t = (3n + m + 1) * PointerSize$

Finally, the Simplex algorithm ([2], p. 67) is used to solve the problem. The following processing stages are necessary to prepare the matrix:

1) The file *eval-pref* is read to build a list of all documents which occur in some preference.

2) The information from the *concepts* file is read into memory.

3) Based on the information from 2) and the *doc-descr* file, a list of atomic concepts is built for each document in the document list.

4) The atomic concepts to be optimized are determined. For this, *eval-pref* is read again. Only atomic concepts occurring in *unsatisfied* preferences are regarded; for each such preference $d <_q^- d'$, all atoms common to q and d, as well as q and d', are considered for the optimization problem.

5) The initial atomic concept weights are read from *atom-wgts*. For each concept in the optimization problem, the value is stored. All other weight values are passed to the program output.

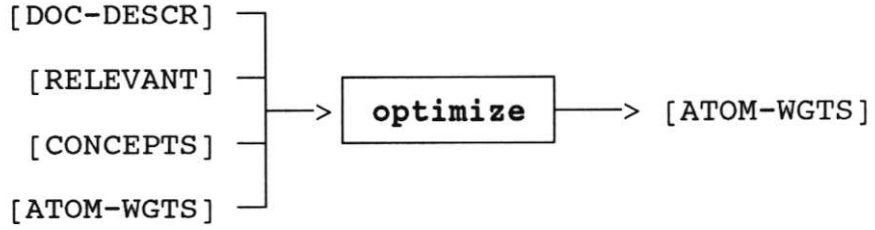
6) The size of the optimization problem is determined. The *eval-pref* file is read and all preferences containing at least one of the optimized concepts are counted. Based on the now known effective number of constraint equations, the row vector of type `Matrix` is allocated.

7) Preferences from the *eval-pref* file are converted into vector form according to [1], p. 62. The cost function is built from unsatisfied and C-tagged preferences; boundary and translation equations are added to the matrix.

8) The n translation equations are eliminated ([2], p. 74). Afterwards, a solution is found through finite iteration. Elimination and iteration both

have complexity $O(n^2m)$. After completion of the algorithm, the optimized weights are output. The new *atom-wgts* file now contains an entry for every concept in the domain algebra.

2.11 Atomic Weight Optimization (Reduced Problem)



Besides attempting to solve the original optimization problem, we also implemented a simpler version which serves only to find situations which downgrade the performance of the retrieval system with respect to the initial domain algebra. The reduced optimization problem consists of the constraints

$$c_1 * IDF^2(\alpha_i) \leq m_i \leq c_2 * IDF^2(\alpha_i)$$

and the cost function

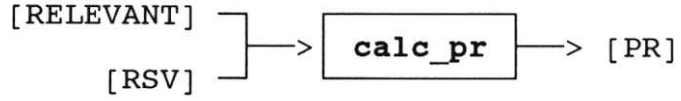
$$cost(\underline{m}) := \Sigma [\, d \text{ in } D_q, RSV_{\underline{m}}(q,d) \,] - \Sigma [\, d \text{ not in } D_q, RSV_{\underline{m}}(q,d) \,]$$

where D_q is the set of documents being relevant to q.

The minimal solution to the cost function is expected to reduce the RSV values of non-relevant documents, while at the same time increasing the RSV values of relevant documents.

The execution flow of the reduced implementation is similar to that of the original program, except that preference relations are no longer involved. In terms of memory usage, the reduced version compares favorably to the original implementation, since only $2n$ constraint equations are needed and the origin of the coordinate system is an edge of the polyhedron determined by the equations, thus making the elimination step unnecessary.

2.12 Precision/Recall Graph Generation



This program outputs the following data from the retrieval function values for each query:

- The ranks and ID's of the five worst-ranked relevant documents, as well as the five highest-ranked non-relevant documents. This information is necessary to construct suitable term dependencies and to determine stopwords.

- Relevant documents d with $RSV(q,d) = 0$.

- Monotonically decreasing precision values for recall values in 0.05 increments according to [1], p. 7, as well as an estimate of the precision function integral. These values can be used as input to the UNIX scatter graph program **qplot** [6].

2.13 Various Utilities

The following two programs have nothing to do with the construction process. They have been developed for verification of hypotheses and conclusions.

termfreq *doc-descr* `[QUIET]` | **sort** `-n -r`

This program produces a list of relative document frequencies. The relative document frequency of a term is defined as

$$rdf(t) := \Sigma [\, d, tf(d,t) \,] / |D|$$

The program calculates a value for each term in the collection. The output, piped through sort to obtain a list in decreasing order, can be used as input to **qplot** [6].

termdisc *doc-descr* `[QUIET]`

The second program estimates the density of the document space by calculating the average distance between a document and an imaginary centroid. The centroid is a vector of term weights w_c, where

$$w_c(t) := \Sigma [\, d, g_d(s) \,] / |D|$$

The average distance d_{avg} is defined as

$$d_{avg} := \Sigma [\, d, sqrt(\Sigma [\, t, (w_c(t) - w_d(t))^2 \,] \,] / |D|$$

3. Data File Formats

[SIGNS]

```
0       a(0)
1       a(1)
:       :
N-1     a(N-1)
```

$\{ \langle s, t \rangle \mid s \in S, t \in A^*; t = a(s) \}$

This table associates signs with terms, thus defining the sign set S and the mapping function a. Negative signs ($s < 0$) identify stopwords.

[SITUATIONS]

```
0   ag0  ob0  ia0  io0

1   ag1  ob1  ia1  io1
:   :    :    :    :
```

$\{ \langle u, ag, ob, ia, io \rangle \mid$
$u \in U; ag, ob, ia, io \in S;$
$ag = agn(u), ob = obj(u),$
$ia = iag(u), io = iob(u) \}$

This table defines situations; it specifies the set U and the functions *agn*, *obj*, *iag*, and *iob*. All numbers are separated by tabs or spaces. Anything following the *io* entry up to the end of the line is ignored; this feature can be used for comments.

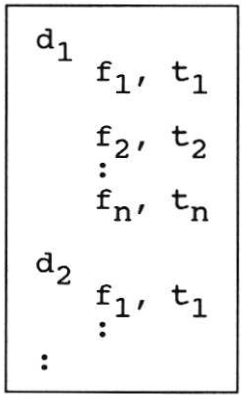
[ABSTRACTIONS]

```
a1   b1

a2   b2

:
```

$\{ \langle s_1, s_2 \rangle \mid s_{12} \in S; (s_1, s_2) \in F_0 \}$

This table defines the graph of the abstraction relation F_0. If two signs a and b are synonyms, the table contains the entries $\langle a, b \rangle$ and $\langle b, a \rangle$. All numbers are separated by tabs or spaces. Definition of synonyms by cycles is allowed; for example, if a, b, c are synonyms, it suffices to define the abstractions $\{ (a,b), (b,c), (c,a) \}$.

[TERM-FREQ]

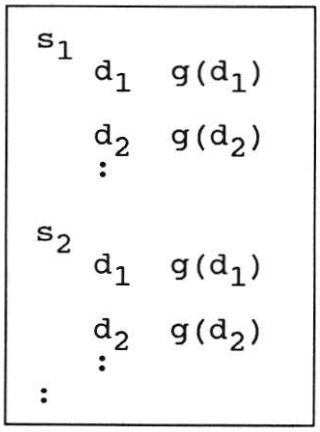


$\{ <d, f, t> \mid d \in \mathrm{Z}, f \in \mathrm{N}, t \in A^*;$
$f = tf(d, t) \}$

This table contains term frequencies for a particular document collection. For each document d in the collection, the table contains all indexing terms t in the document along with the *term frequency* $tf(d,t)$ of t in d. By convention, queries are assigned negative document numbers.

If a tuple $<d, f, t>$ does not occur, then $tf(d, t) = 0$.

[DOC-DESCR]

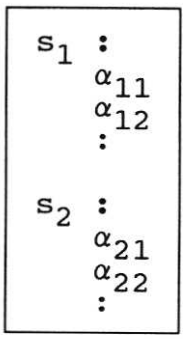


$\{ <s, d, g> \mid s \in S, d \in \mathrm{Z}, g \in \mathrm{R};$
$g = g_d(s) \}$

This table specifies the weight of each sign s in each document d.

If a tuple $<s, d, g>$ does not occur, then $g_d(s) = 0$.

[CONCEPTS]

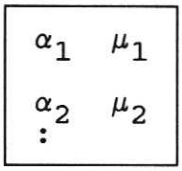


$\{ <s, \{\alpha_i\}> \mid s \in S, \alpha_i \in \mathrm{N},$
$C(s) = \mathrm{U}(\alpha_i) \}$

This table defines each sign as a composition of one or several atomic concepts. The concepts are identified by unique ID numbers.

To distinguish the signs from the atomic concepts, each sign must be followed by a colon.

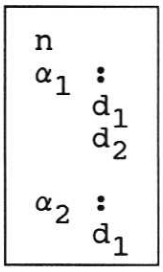
[ATOM-WGTS]



$\{ <\alpha_i, \mu_i> \mid \alpha_i \in \mathrm{N},\ \mu_i \in \mathrm{R},$
$\mu_i = \mu(\alpha_i) \}$

This table contains the weights of all atomic concepts, which are identified by their numeric ID.

[ATOM-DOCS]



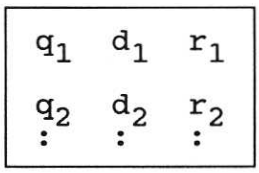
$\{ <\alpha_i, \{d_j\}> \mid d_j \in \mathrm{Z}$, concept α_i occurs in all $d_j \}$

This table lists all atomic concepts and the documents where each occurs.

To distinguish the atomic concepts from the document indices, each concept ID α must be followed by a colon.

The first line of the file must contain the total number of documents in the collection.

[RSV]



$\{ <q, d, r> \mid q \in \mathrm{Z}, d \in \mathrm{N}, r \in \mathrm{R};$
$r = RSV(g_q, g_d) \}$

This table contains the *retrieval status values* of all documents in the collection (except queries) with respect to one or more test queries. The queries q and documents d are identified by their negative, resp. positive document id numbers.

If a tuple $<q, d, r>$ does not occur, then $RSV(g_q, g_d) = 0$.

[EVAL-PREF]

```
c   q   di   dj
:
```

$\{ <c, q, d_i, d_j> \mid q \in \mathbb{Z}, d_{i,j} \in \mathbb{N},$
$c \in \{+, -, C\} \}$
$c = - \mid C: \; d_i <_q^- d_j$
$c = + : \; d_i <_q^+ d_j$

This table specifies unsatisfied and satisfied preferences. Minus signs denote unsatisfied preferences, plus signs denote satisfied preferences. The **C** letter is used for unsatisfied preferences which are of interest for the optimization cost function only.

[RELEVANT]

```
q1 :
     d1   r1
     d2   r2

q2 :
     d1   r1
```

$\{ <q, d, r> \mid q \in \mathbb{Z}, d, r \in \mathbb{N},$
d is relevant to q with
relevance level r $\}$

For each query, a list of relevant documents is given. Although, by convention, queries are listed with negative document numbers in other tables, q is here allowed to be positive or negative for convenience. Higher relevance levels imply higher document relevance, with 1 being the lowest allowable relevance level.

To distinguish the queries from the documents, each query number must be followed by a colon.

4. Experimental Results

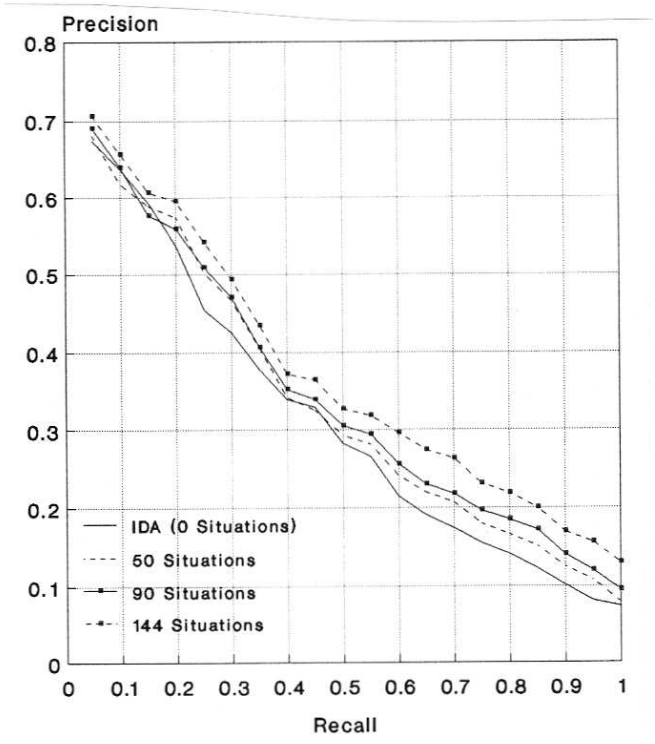
4.1 The CACM Domain Algebra

The CACM document collection consists of 3204 documents and 52 queries. On the average, 15 documents are relevant to each query. Indexing was done as described in the section "*Document Collection Parsing*". The 250-word stoplist `~ir/data/stoplists/sl1` was used.

After generating the initial domain algebra, the performance of the retrieval system was determined with the **calc_pr** program. Based on this information, term dependencies were specified for each document d relevant to a query q with $RSV(q,d) = 0$ by inspecting the document contents. If no such document d existed, we tried to improve the *RSV*'s of the worst-ranked relevant documents with respect to q. Abstractions were only used to identify synonyms. **calc_pr** was run again after specification of 50, 90, and 144 situations, and subsequent situations were specified based on the updated performance information.

The descriptors `IAG_1`, `IOB_1` etc. were used for the interacting agent/object signs. 96 multiword terms were added, which resulted in 9,576 signs after 144 situations, as opposed to 9,192 signs in the initial domain algebra. 144 situations correspond to approx. 3 situations per query.

The following precision/recall graph shows the performance improvement obtained with term dependencies, without optimization:



To estimate the performance improvement, the area F was calculated for a curve as follows:

$$F = \Sigma [\, i, (y_{i+1} + y_i) / 2 \,] * dx$$

where y_i are precision values and dx is the x-distance between subsequent recall values. The *absolute* improvement for a curve with area F is $(F/F_{IDA} - 1)$ where F_{IDA} is the curve area for the initial domain algebra. Likewise, the relative improvement is $(F/F' - 1)$, where F' is the area of the previous curve.

Relative and absolute performance improvements for the CACM collection are:

# Sit.	rel	abs
0	-	-
50	+ 6.49%	+ 6.49%
90	+ 3.11%	+ 9.80%
144	+ 2.01%	+19.68%

From these results, we can see the following:

1) The introduced term dependencies significantly improve recall for the CACM collection. This was expected since situations were mostly specified for documents with zero RSV values and thus lead to an increased number of relevant documents in the ranking list. There are two areas of improvement: recall levels 0.2-0.4 and 0.5-1.0. Increase of precision in the range 0.2-0.4 is presumably caused by the improved ranking of documents which were formerly retrieved in the high recall range. Precision increase in the range 0.5-1.0 can be attributed to the documents with a previous RSV value of zero, but which are now retrieved with a positive RSV because of term dependencies.

   Since the weight of the atomic concept $c(iag) \cap c(iob)$ is usually small, most newly retrieved relevant documents exhibit low RSV values and appear at the end of the ranking list, thus raising precision at high recall levels. The top ranks are mostly occupied by documents which contain query terms. Therefore, it is improbable that a document with a previous zero RSV can be lifted to a top ranking position by a situation alone.

2) Precision at low recall levels may decrease after specification of term dependencies. This is because non-relevant documents in top positions may also contain a situation's agent or object, which causes minor increases of the RSV value and the ranking of such documents.

4.2 CACM Partial Optimization

Due to limitations of the available hardware, it was not possible to solve the full optimization problem; we therefore decided to optimize each query separately. Unsatisfied preferences were generated for one query at a time; satisfied

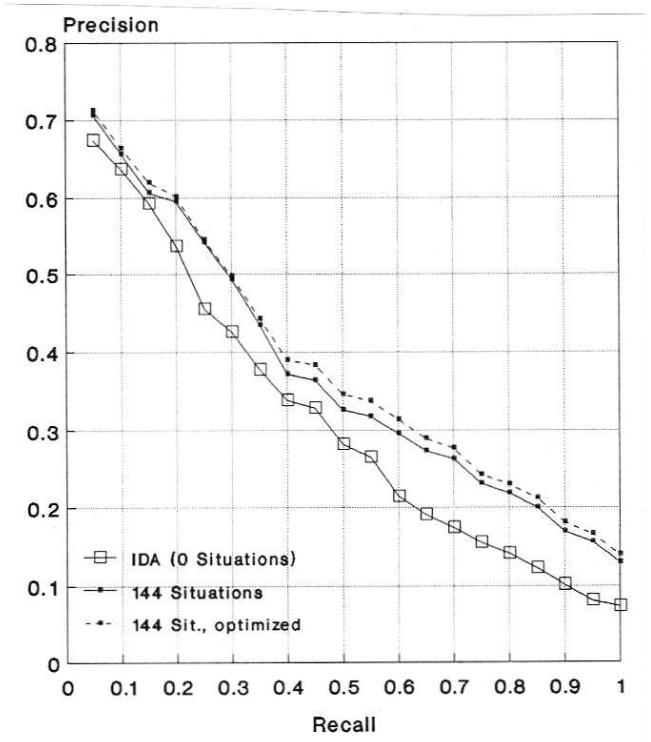
preferences were generated for all queries. The weights to be optimized were determined from the unsatisfied preferences, as detailed earlier. The optimization problem for each query was initialized with the weight distribution of the previous query instead of the atomic concept *IDF*'s. This scheme has the following properties:

1) The optimization is expected to improve the ranking of relevant documents with respect to each query.

2) Although each query is optimized separately, the performance for other queries does not suffer since previously satisfied preferences remain satisfied.

3) The overall performance improvement obtained by these *local* optimizations might not be as good as the results that could be achieved by solving the full, *global* optimization problem.

The actual calculation took over two weeks to complete. The constants c_{12} ([1], p. 77) were set to $c_1 = 0.5$ and $c_2 = 2.0$. The following graph shows the achieved results:

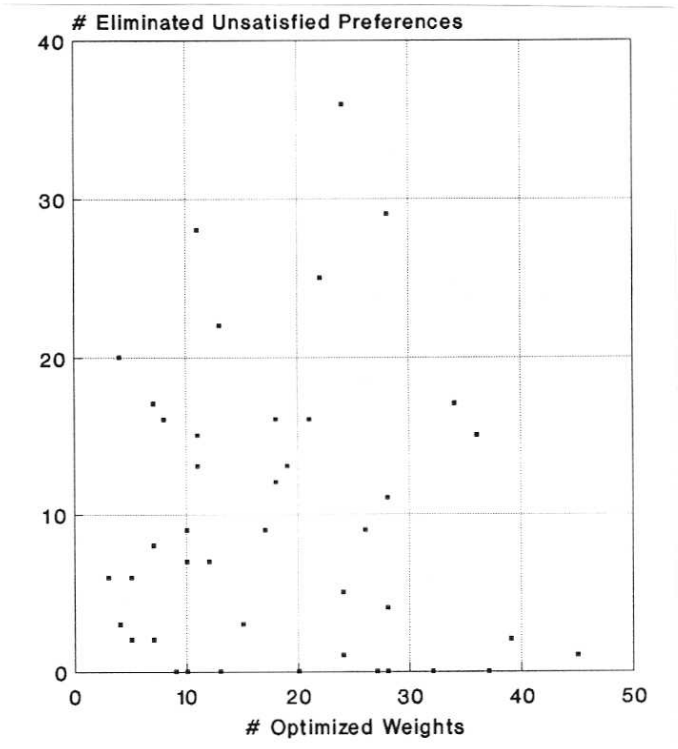


The corresponding overall improvement is

# Sit.	rel	abs
0	-	-
144	+ 19.68%	+19.68%
opt	+ 3.34%	+23.68%

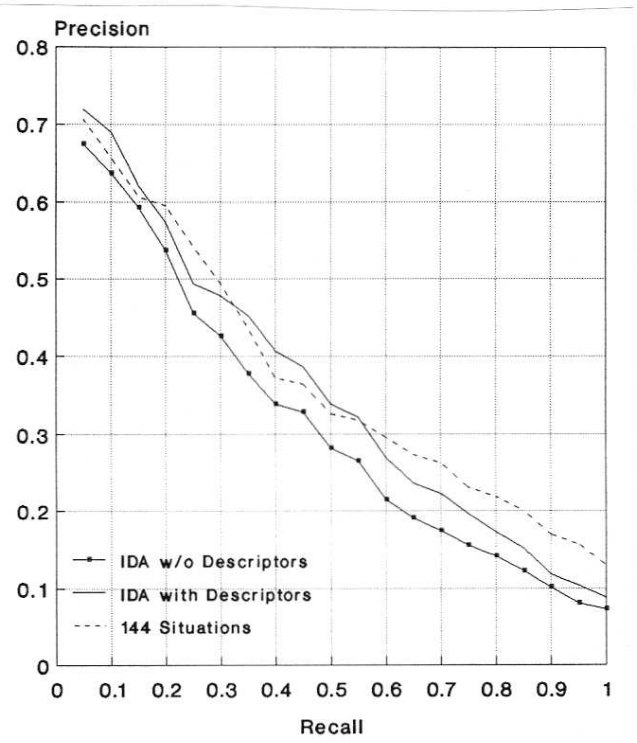
As we can see, the optimization increased precision in the high recall range only. This can be explained as follows: if a relevant document appears in a top position in the ranking list, then relatively few unsatisfied preferences are generated for it. In this sense, the cost function is not normalized; low-ranked documents contribute more than high-ranked ones. Thus, the ranking of documents in good positions is not improved significantly by optimization. In terms of satisfied preferences, the entire optimization increased the number of $<^+$ preferences from 770,663 to 771,225, or 0.07%, which must be considered a rather minor improvement.

Because all 52 queries were treated separately, the number of optimized weights in each run was small. One might assume that the solution of the full optimization problem would yield a significantly better weight distribution. The following scatter plot shows the number of atomic weights versus the number of eliminated unsatisfied preferences for all runs; since there is no clear correlation, the above hypothesis is questionable.



It is also not clear whether a better choice of c_{12} would result in greater improvements. Experiments with the department library catalog suggest that the settings of c_{12} have no significant effect on the outcome of the optimization.

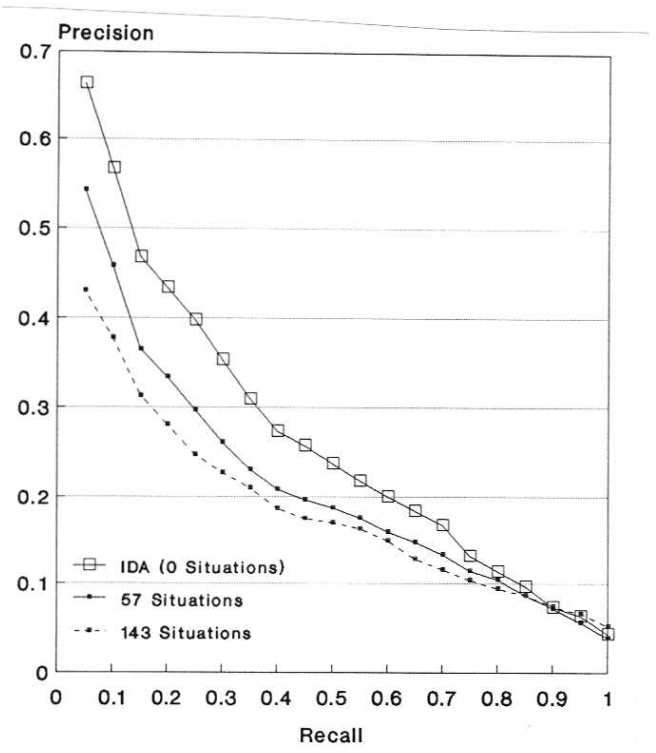
To compare the effectiveness of the domain algebra to the performance of manual indexing, an initial domain algebra was generated for the CACM collection including descriptors (text after the **.K** headers). The precision/recall of this information structure in comparison to the 144-situation domain algebra is shown below:



While the domain algebra increases precision in the high recall range > 0.5, descriptors perform better in the "visible" area of the ranking list. Since manual indexing and situations both introduce common concepts in query and document, the results suggest that it should still be possible to improve the present CACM domain algebra. The superior domain algebra performance in high recall levels is caused by the fact that common terms usually result in higher RSV values than merely common concepts.

4.3 The CISI Domain Algebra

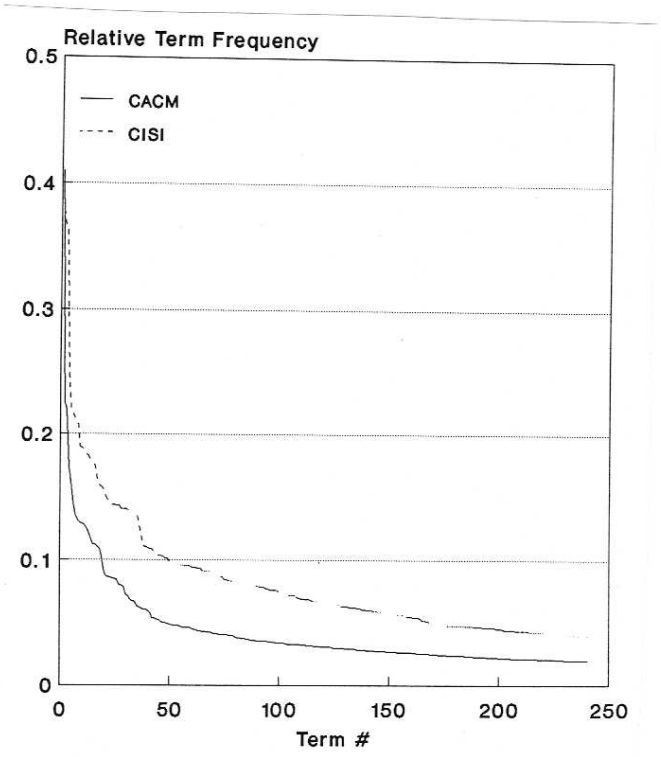
The CISI document collection consists of 1460 documents and 112 queries. On the average, 40 documents are relevant to each query. Indexing and specification of term dependencies was done under the same conditions as for the CACM collection. At the end, 143 situations were specified. The resulting performance is shown below.



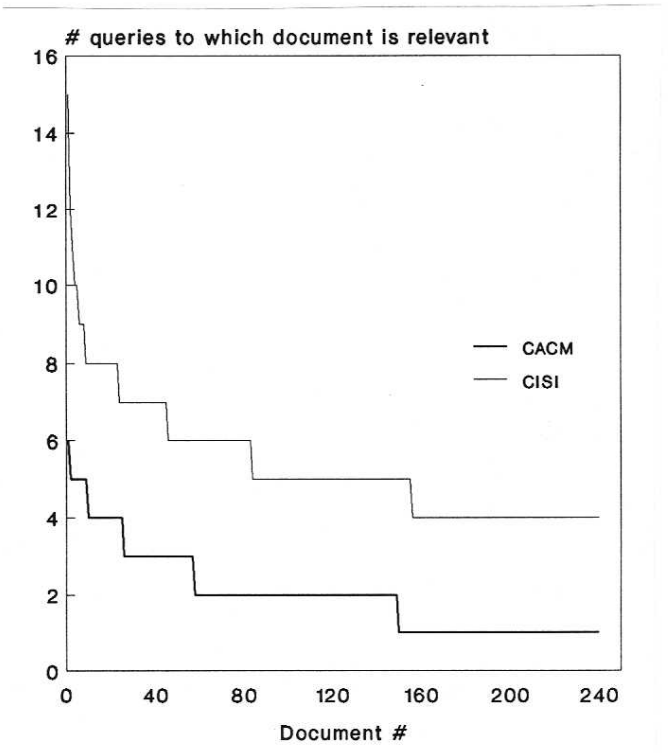
The decrease of performance in figures is

# Sit.	rel	abs
0	-	-
57	-20.90%	-20.90%
143	-12.01%	-30.40%

The results show a drastic negative effect of the specified term dependencies on the retrieval system performance. Contrary to the CACM results, situations improve the ranking of a large number of non-relevant documents while introducing nearly no relevant documents with former zero RSV values. This is due to two facts. First, the document term frequency of agents and objects is high, that is, relevant documents in the CISI collection often contain terms which are also found in non-relevant documents (e.g. *library* and *information retrieval*). It is therefore very difficult to specify adequate situations in this collection. The following graph shows the relative document frequencies of terms in the CACM and CISI collections, ordered by decreasing value. It is clear that the average frequency is much higher in CISI than in CACM.



Secondly, CACM documents are relevant to fewer queries than CISI documents. The following graph shows the number of queries a document is relevant to, with values in decreasing order.



Only 27% of the CACM documents are relevant to more than 1 query; this figure is 73% for the CISI collection, and 42% of the CISI documents are relevant to more than 2 queries. It follows that term dependencies specified for a CISI document are likely to interfere with the performance of other queries, as opposed to the CACM collection, where the sets of relevant documents do not overlap significantly.

The degree of intersection of relevant document sets along with the relative term frequency seem to be good criteria to decide whether a domain algebra will lead to any significant improvement for a given document collection. In addition, it would be nice to know *which* situations in particular cause an improvement or deterioration. The following section shows a possible approach to this problem.

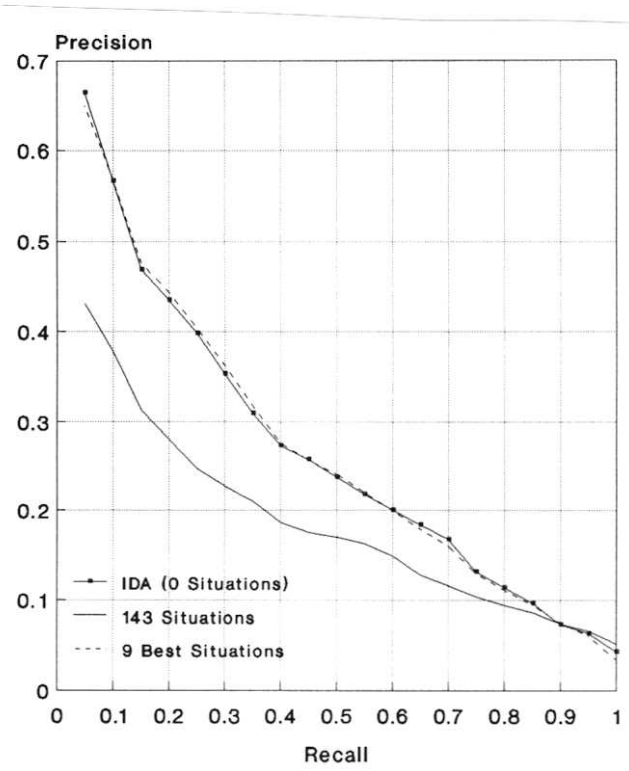
4.4 CISI Stopword/Situation Evaluation

Since the optimization of the good CACM domain algebra yielded no substantial improvements, we assumed that an optimization based on the poor CISI domain algebra would result in only minor improvements over the IDA performance. Instead, the available time was used to address the question how the influence of a given situation on the retrieval system performance can be determined. The idea was that the optimized weights of a situation's concepts reflect the importance of the particular situation. If the weight is increased by the optimization process, the situation has a positive influence on performance. Likewise, if the weight is decreased, then the situation is not appropriate and should be discarded.

The specification of inappropriate situations is not the only reason for performance degradation; common terms in a query and a large number of non-relevant documents also cause a negative effect. It is therefore desirable to reduce the weight of such high-frequency terms as well. These terms can be regarded as stopwords which are specific to a particular document collection. Thus, optimization can be interpreted as a dynamic extension of the collection-independent general stoplist which is used for the generation of the initial domain algebra.

Because the original optimization problem required too much computing power, a simpler problem was designed (see the section "*Atomic Weight Optimization (Reduced Problem)* "). The reduced problem was expected to decrease the weights of inappropriate situations and stopwords. 112 stopwords were specified for the CISI collection by negating their sign numbers in the *signs* file. They were selected manually, based on inspection of the query and high-ranked non-relevant documents. The reduced optimization problem was solved based on the standard 143-situation domain algebra.

The results are not surprising: the weights of all but nine situations were reduced to zero. The performance of the modified domain algebra with the 9 situations with non-zero weights is shown along with precision/recall for the IDA and 143 situations:



The graph shows that the selected situations indeed improve performance. Since the reduced problem can be solved in relatively short time (30 minutes on a SUN workstation), this method should provide an efficient means to evaluate the impact of situations.

4.5 Department Library Catalog

We also examined the effect of a domain algebra on the multilingual department library catalog INF. Relevant data (titles, subtitles, authors, book signatures) was extracted from a relational data base system and converted to the SMART format. First, we had to deal with the problem that document abstracts and relevance assessments were not available. We constructed 2 test queries on information retrieval and computer graphics and specified 20 term dependencies between corresponding German, English, and French terms, as well as plural/singular dependencies (word stemming was disabled for this collection). This allowed us to obtain initial ranking lists from which relevant documents were chosen.

Documents in the ranking lists were grouped in five relevance classes ordered by decreasing relevance: class (5), the highest relevance class, contained textbooks and introductory material on the subject; (4) contained dissertations, since these often also provide a general overview of the topic; (3) contained textbooks with an emphasis on some particular field within the subject; (2) contained highly specialized material, such as proceedings and technical reports; and (1), the lowest relevance class, contained other books which were somehow related to the subject.

Due to the lack of abstracts, it was not possible to create a useful information structure based on book titles alone. Therefore, we decided to add additional descriptors to the queries and relevant documents. For each relevant book, the main chapter titles were added from the book's table of contents. These descriptors provided an adequate representation of the actual book contents.

In a second step, new ranking lists were generated based on the relevance information. Based on the new ranking, 19 situations and abstractions were specified to introduce term dependencies between words in different languages. Finally, optimization steps with $c_1 = 0.5$, $c_2 = 2.0$ were carried out on the augmented domain algebra.

The direct calculation of precision/recall graphs is not possible for document collections with more than 2 relevance levels. We thus judged the performance p based on the number of satisfied preferences:

$$p := \Sigma [\, q \in Q, n_+ / (n_+ + n_-) \,] / |Q|$$

where n_+ and n_- are the number of satisfied, resp. unsatisfied preferences. The results for the INF collection are presented together with the corresponding figures for the CACM/CISI collections in the following table:

	CACM	CISI	INF
IDA	**0.7973**	**0.7773**	0.8306
with DA, w/o opt.	**0.9054** +13.6%	**0.7202** -7.35%	**0.9312** +12.11%
with DA, after opt	**0.9060** +0.07%	no opt	**0.9925** +6.58%

c1 = 0.5
c2 = 2.0
(all collections)

The CACM/CISI figures basically confirm the results of the previous sections. For the INF collection, the improvement obtained with the domain algebra is smaller than the improvement after optimization of the domain algebra. This is because (1) adding descriptors to relevant documents caused most relevant documents to appear at the top of the ranking list, and (2) optimization adjusted the order of documents with different relevance levels within this cluster.

To examine the effect of c_2 on the optimized weights, the initial domain algebra of the INF collection was optimized for one query with $c_1 = 0$ and $c_2 = 2, 8, 128$. The table below shows the resulting values of p. For values above $c_2 = 8$, optimization of the INF collection does not yield any further improvement, which confirms that the poor outcome of the optimization cannot be attributed to an unlucky choice of $c_{1,2}$.

	c_2=2	c_2=8	c_2=128
INF 1 query	**0.9838**	**0.9841** +0.03%	**0.9841** +0.00%

5. Summary and Conclusions

An experimental system for the practical evaluation of the domain algebra construction process has been implemented and tested on three document collections. The results can be summarized as follows:

1) The role of optimization in the construction process has been overestimated. While extremely time-consuming, the optimization process failed to produce significant improvements.

2) It is possible to obtain significant improvements of both recall and precision with a domain algebra. However, improvement is only guaranteed in connection with the optimization step. If this step is omitted, the possibility arises that inadequate term dependencies degrade performance of the retrieval system. The reduced optimization problem represents a feasible method to locate such inappropriate dependencies.

3) The domain algebra is a promising information structure for multilingual retrieval systems, since terms in different languages can be related via situations. Still, various problems need to be solved, among them the design of a multilingual stemming algorithm, recognition of multiword terms in different languages, and multilingual stoplists.

On the other hand, some important questions remain to be answered. For example, it is still unknown whether the possible improvement for a domain algebra approaches a fixed limit for large numbers of situations. It was shown that term dependencies tend to increase precision in the high recall range. Unfortunately, this range is usually not considered by a human user. Therefore, the question arises whether there is a method to find term dependencies such that documents in the medium recall range are lifted into top positions of the ranking list.

Due to lack of time, several points could not be resolved by this work. For example, the results for the INF collection can be questioned, since additional indexing was done only for relevant documents, thus giving them an "unfair" advantage. Also, the number of terms in the documents must be high enough to allow the specification of appropriate situations. From this point of view, the INF collection was not a suitable test object for our purposes.

The chosen stand-alone program structure turned out to be a good idea for verification and maintenance. However, user-friendliness has been neglected; the current method of specifying term dependencies by modifying files with a text editor is rather primitive. It would be interesting to combine the current experimental system with the *dtool* domain algebra editor described in [1], and provide an enhanced user front-end to obtain a functional interactive retrieval system based on a domain algebra.

Appendix A. File Directory Contents

~/bin

links to objects in **~/src**

~/docs

cacm,cisi,info	Directories with data files for document collection runs
makefile	Makefile for domain algebra

~/old

evalpref.c.wolevel	Old *evalpref.c* source without relevance levels
simplex.new.c	Program to solve reduced optimization problem
simplex.prefs.c	Program to solve full optimization problem

~/src

Makefile	Makefile for generation of sources
atomdocs.c	*calc_atomdocs* source
boolean.h	BOOLEAN type definition
calc_pr.c	*calc_pr* source
calc_rsv.c	*calc_rsv* source
calcswgt.c	*calc_docdescr* source
concepts.c	*build_concepts* source
evalpref.c	*eval_prefs* source
generate.c	*generate_ida* source
initatom.c	*init_atomwgts* source
limits.h	Definition of c_1, c_2 for *optimize* and *eval_prefs*
list.c	list manager module, imported by all programs
list.h	header file for *list.c*
parse.c	*parse_itemcoll* source
simplex.c	simplex optimization source
stemtest.c	Interactive utility to check stemming algorithm
termdisc.c	Centroid distance calculation source
termfreq.c	Term frequency calculation source
util.c	Utility module, imported by all programs
util.h	Header file for *util.c*
wordstem.c	Porter word stemmer source
wordstem.h	Header file for *wordstem.c*

~ir/data/testcoll/{cacm,cisi,delphi}/DOC_FREQ

Term frequency files for cacm/cisi/delphi collections

~ir/data/testcoll/delphi/query.text

File with information retrieval/computer graphic queries

~ir/guido.cacm

log	Logfile for automatic run
autorun	Shell script for automatic query optimization
results,	
autores	Directories with precision/recall graphs for each query

References

[1] Peter Schäuble. *Information Retrieval Based on Information Structures.* D.Sc. Thesis ETH, No. 8784, ETH Zurich, 1989.

[2] H.R. Schwarz. *Numerische Mathematik.* Teubner-Verlag Stuttgart, 1986 (p. 60 ff.)

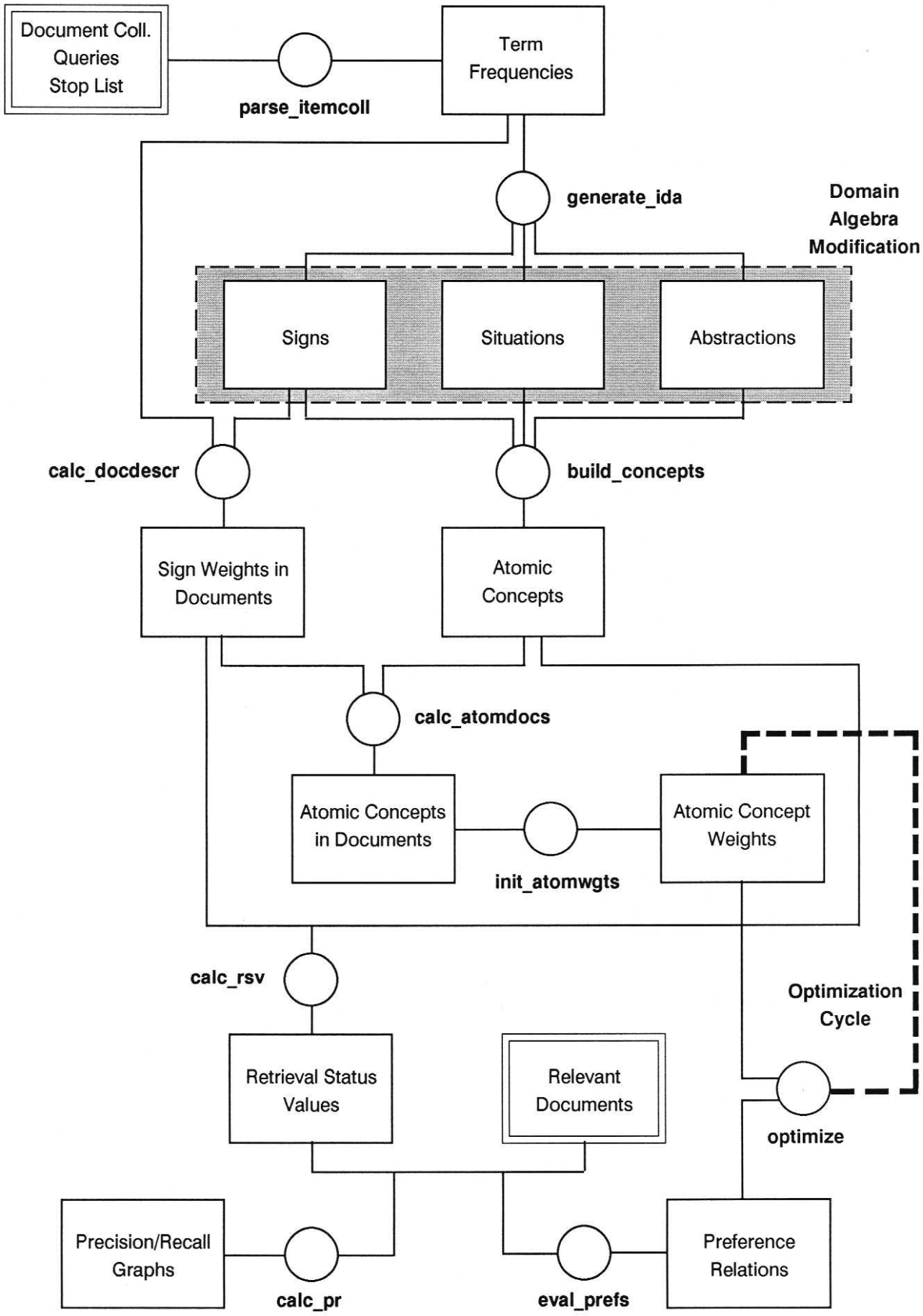
[3] Martin Bärtschi. *Term Dependence in Information Retrieval Models.* D.Sc. Thesis ETH, No. 7525, ETH Zurich, 1984 (p. 94).

[4] M.F. Porter. *An Algorithm for Suffix Stripping.* Program 14(3), 1980 (p. 130-137).

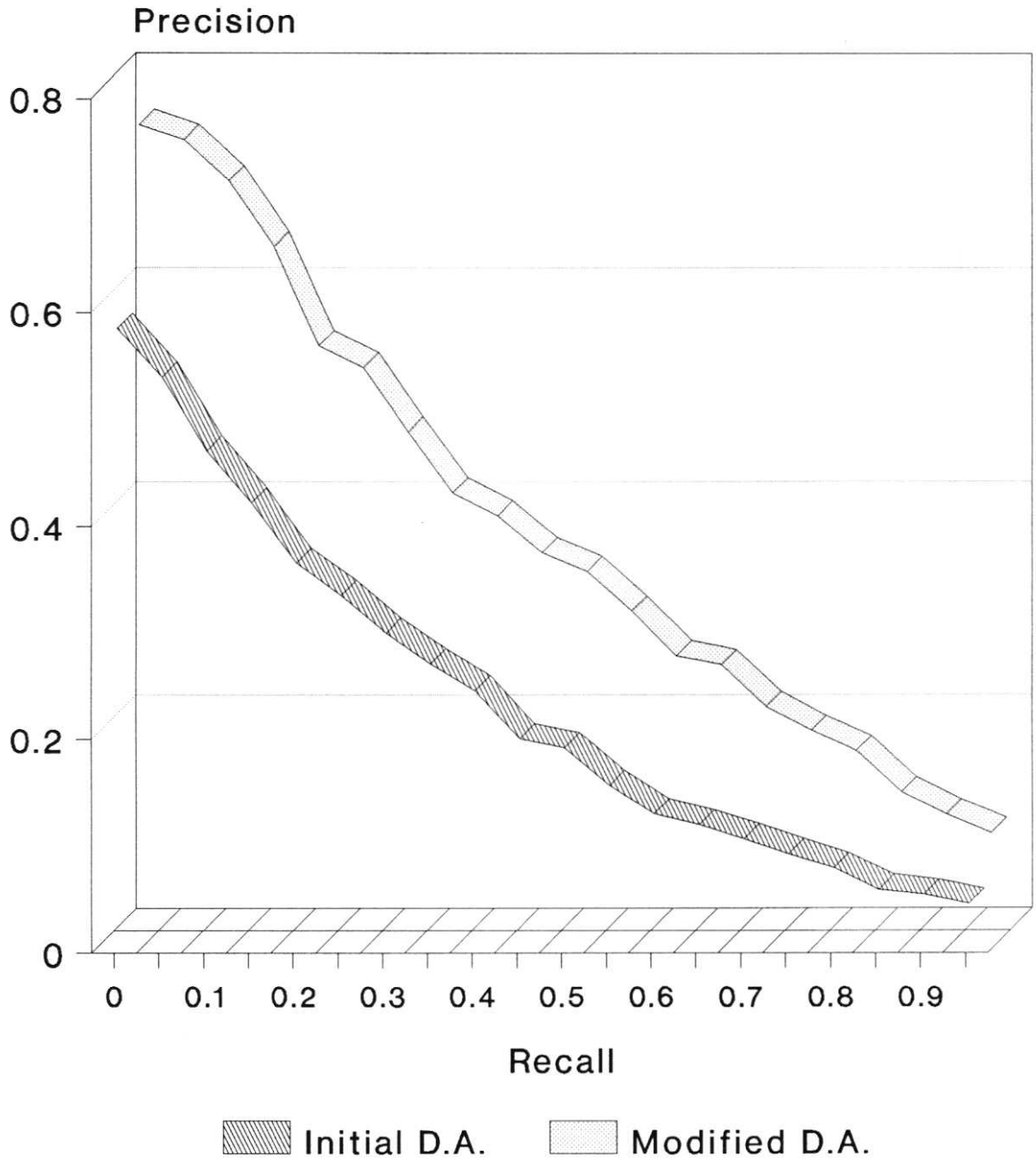
[5] UNIX *make*(1) manual page.

[6] UNIX *qplot*(1) manual page.

Domain Algebra Construction: File Setup



CACM Test Collection
(3 of 52 Queries, 30 Situations)



Word Stemming

Advantages

- Fewer term dependencies necessary ("parsing", "parser", "parse")
- Reduction of term space dimension (CACM: 25%)

Disadvantages

- Current implementation (Porter-algorithm) introduces undesired side effects

A) "optimized code", "optimal code"

-> "optim code"

B) "parallel computing", "parallel computer"

-> "parallel comput"

Consequences

- Full word stemming causes loss of precision
- No word stemming leads to more term dependencies
- Plural-to-singular conversion might be best

Optimization Problem

Theoretical Simplex matrix for CACM collection:

1,500,000	rows (preferences)
8,500	columns (atomic weights)

-> Optimization not feasible on available hardware

-> Reduce number of preferences (a) and weights (b)

<u>(a) Preferences</u>

- **Discard stable preferences (always satisfied)**
- **Approach reduces preferences to 23,000**

<u>(b) Atomic Weights</u>

- **Distribute optimization problem**
- **Optimize clusters of atomic weights**
- **Approach yields local (not global) optimum**

Term Dependency Types

A) General Term - Specific Term

A = "compiler"

B = "optimizing compiler"

C(A), the set of atomic concepts of A, is a subset of C(B)

B) Situations

A = "register allocation"

B = "code optimization"

C(A) and C(B) intersect

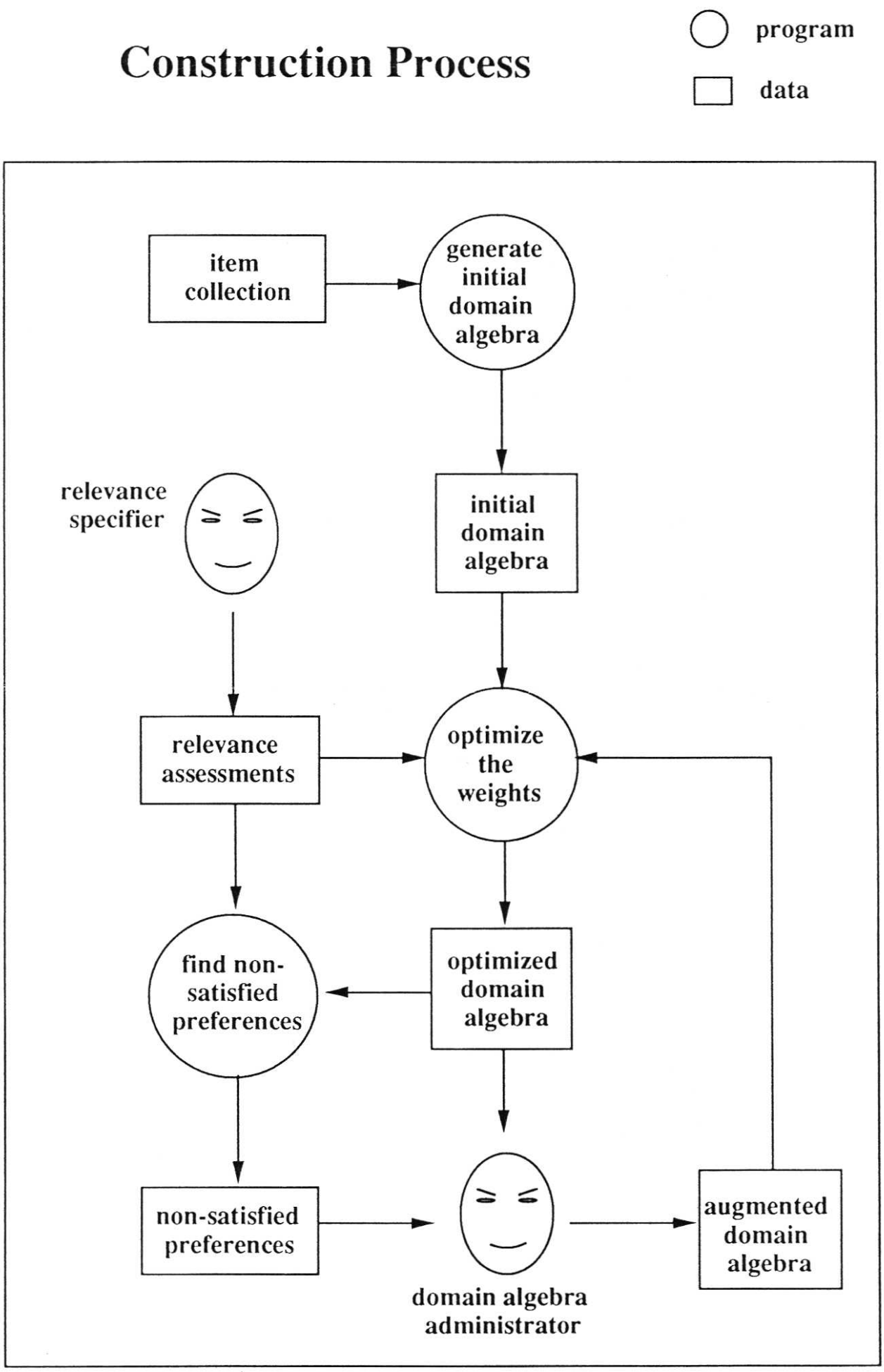
C) Synonyms

A = "syntax analysis"

B = "syntactic analysis"

C(A) and C(B) are equal

Construction Process



Next Steps

- Examine effects of simplified stemming algorithm
- Implement clustering approach for optimization problem
- Performance analysis of revised system