Signature File Methods for Semantic Query Caching

Boris Chidlovskii[1] and Uwe M. Borghoff[2]

[1] Xerox Research Centre Europe, Grenoble Laboratory
6, Chemin de Maupertuis, F–38240 Meylan, France
childovskii@xrce.xerox.com
[2] Institut für Softwaretechnologie, Fakultät für Informatik
Universität der Bundeswehr München, D-85577 Neubiberg, Germany
borghoff@informatik.unibw-muenchen.de

Abstract. In digital libraries accessing distributed Web-based bibliographic repositories, performance is a major issue. Efficient query processing requires an appropriate caching mechanism. Unfortunately, standard page-based as well as tuple-based caching mechanisms designed for conventional databases are not efficient on the Web, where keyword-based querying is often the only way to retrieve data. Therefore, we study the problem of semantic caching of Web queries and develop a caching mechanism for conjunctive Web queries based on *signature files*. We propose two implementation choices. A first algorithm copes with the relation of semantic containment between a query and the corresponding cache items. A second algorithm extends this processing to more complex cases of semantic intersection. We report results of experiments and show how the caching mechanism is successfully realized in the Knowledge Broker system.

1 Introduction

Digital libraries operating in a networked environment process user queries by contacting heterogeneous Web repositories that provide bibliographic data in particular domains. Such systems invoke so-called *wrappers* to convert user queries into the target query language, and to control the return flow of data from these servers [19–21]. As data are transferred over the network in HTML/XML format, the wrappers also extract answer documents from the retrieved HTML/XML files before they report the final answers (often locally pre-filtered) to the user.

As in any client-server system, high performance in such a networked digital library is often reached by efficient utilization of computational storage resources at the client sites. In the networked environment, where data from remote servers are brought to clients on-demand, local client memory is largely used to cache data and minimize future interaction with the servers. This data caching is particularly important in Web-based digital libraries, as network traffic and overloaded servers can lead to long delays in answer delivery. As standard page caching is not possible on the Web, and tuple-caching has certain limitations,

much effort has been spent to cache user queries and corresponding answers (instead of pages or tuples) for possible future reuse [4, 9, 13].

Query caching is particular advantageous when the user of a digital library refines a query several times, for example, by adding or removing a query term. In this case, many of the answer documents may already be cached and can be delivered to the user right away.

A typical query to a Web data repository is a conjunction of terms. Each term in the query is a keyword, possibly negated with the operator `NOT`, and applied to one or more attributes (title, author, etc.). In most Web repositories allowing the search over the site contents, the operator `NOT` is equivalent to `AND NOT` to force a query to contain at least one non-negated term.

Semantic caching. Semantic caching manages the client cache as a collection of semantic *regions*; access information is managed and cache replacement is performed at the unit of semantic regions [9]. Semantic regions group together semantically related documents covered, for example, by a user query.

Each semantic region has a constraint formula which describes its contents, a region signature, a counter of tuples in the contents, a pointer to the set of actual tuples in the cache, and the additional information that is used by the replacement policy to rank the regions. Like a query, any region formula is a conjunction of terms.

When a query is posed at a client, it is split into two disjoint pieces: (1) the portion of the answer available in the local cache, and (2) a *remainder* query, which retrieves any missing information from the server. If the remainder query is not null (i.e., the query asks for some documents that are not cached so far), the remainder query is sent to the server for further processing [5].

A semantic model for query caching mechanisms in a client-server architecture was discussed in [9]. Query caching in heterogeneous systems was discussed in [13], where it is reduced to a Datalog query evaluation, which, however, may by computationally hard. Intelligent query caching is also used in the SIMS project [4], where some important principles for any intelligent caching mechanism were developed. These principles are: 1) a query cache should process both containment and intersection cases; 2) a cache item should not be large; 3) a cache item should have a simple formula to avoid too complex reasoning on the query remainders.

Semantic caching versus page and tuple caching. In a standard client-server architecture the transfer units between servers and clients are pages or tuple sets, unlike in a networked digital library. *Page caching* mechanisms assume that each query posed at the client can be processed locally and be broken down to the level of requests for individual pages. Then, if a requested page is not present in the client cache, a request for the entire page is sent to the server. Such a query processing is improper in a Web-based digital library, where keyword-based querying is often the only way to retrieve data and where the data organization at the servers is completely hidden from the clients.

With *tuple caching*, the cache is maintained in terms of individual tuples or documents, allowing a higher degree of flexibility than pure page caching. On the Web, tuple caching is possible though not very attractive, as there is no way to inform the servers about qualified tuples in the client cache. Moreover, clients can not detect whether their local caches provide a complete answer to the queries. As a result, clients are forced to ignore their cache entries while performing the query. Once the query is sent to the server and all qualifying tuples are returned, the clients detect and discard the duplications.

Our contribution. In this paper, we develop a new mechanism for caching Web queries which is based on so-called *signature files*. Each semantic region in the cache is associated with a signature. For a user query, the signature is created in a similar way and verified against the region signatures stored in the cache. The proposed caching mechanism includes a procedure that identifies all cache items qualified for the query, i.e., it detects which cache items can be re-used immediately, and which missing information must be requested from the servers.

This mechanism has three main advantages. First, it processes both critical cases in the same elegant way, 1) when a query is contained in the cache, or 2) when it intersects some regions. As a result, the proposed mechanism avoids most cases of tuple duplications, and has a moderate storage requirement. Second, it supports efficient reporting of partial answers and generating of query remainders. Finally, it provides a simple solution for the region coalescing and the replacement policy.

Although the main motivation and the targeted use of our proposed algorithms are Web-based digital libraries, we point out that the presented approach can be easily generalized to full-fledged distributed database environments.

The remainder of the paper is organized as follows. Section 2 introduces the signature file methods and discusses the cache architecture based on semantic region signatures. Section 3 studies the semantic containment between a query and semantic regions. The first caching algorithm is presented. In Section 4, we develop a second algorithm which covers the semantic intersection too. Results of experiments with the caching algorithms are reported in Section 6. Section 7 discusses some open issues and concludes the paper.

2 Signature Files and Cache Architecture

Signature files were originally designed for the retrieval of full-text fragments containing query words [10, 11]. Consequently, signature files where used in a variety of applications, including navigation in OODBS, indexing of Prolog items, and multimedia office management [11, 12, 15]. The best known technique uses the superimposed coding to associate each semantic region with a formula in the conjunctive form. Each term in a region formula is assigned a term signature represented as a binary sequence of ones and zeros. The region signature is generated by superimposing (bit-wise OR-ing) all term signatures generated from the region formula. Figure 1.a shows the signature generation for the semantic

Region formula: "query ∧ caching"	
Term signatures:	
query	0010 0010 1000
caching	0100 0100 0001
Region signature:	0110 0110 1001

a)

Queries	Query Signatures	Results
1) Web	1000 0001 1000	no match
2) caching	0100 0100 0001	region containment
3) query ∧ caching	0110 0110 1001	equivalence
4) Web ∧ query ∧ caching	1110 0111 1101	query containment
6) false ∧ drop	0110 0110 1001	false drop

b)

Fig. 1. a) Region signature construction; b) Sample queries and their signatures

region "query ∧ caching". For a user query – which is also a conjunction – all terms are assigned signatures and superimposed onto a query signature in a way similar to regions. Then, the query signature is matched against each region signature in the signature file to provide a partial answer and construct the query remainder.

The two caching algorithms proposed in this paper work with different semantic relations between semantic regions and the query. The first caching algorithm copes with *semantic containment* between the query and a region, where one contains the other. The second caching algorithm described in Sect. 4 extends this processing to the more frequent and complex cases of *semantic intersection*, when neither region contains the query nor vice versa. We begin with the semantic containment which consists of three cases. Query Q is *equivalent* to a region R if their formulas are equivalent. A region R *contains* query Q if the query formula can be obtained from the region formula by dropping one or more terms. In this case, the answer to the query is a proper subset of the region contents. Inversely, the semantic region R is contained in a query Q if the region formula can be obtained from the query by dropping one or more query terms. The region containment implies that the answer is a superset of the region contents. In any of the three cases described above, the region R is *qualified* for query Q.

Let S_Q and S_R denote a query signature and a region signature, respectively. With the bit-wise comparison of the signatures, the semantic containment is detected as follows:

Region containment, $S_Q \subset S_R$: for each bit in the query signature set to one, the corresponding bit in the region signature is also set to one (see Query 2 in Fig. 1.b).

Equivalence, $S_Q = S_R$: the region and query signatures have the same bits set to one (see Query 3 in Fig. 1.b).

Query containment, $S_Q \supset S_R$: for each bit in the region signature set to one, the corresponding bit in the query signature is also set to one (see Query 4 in Fig. 1.b).

A signature file eliminates most, but not all of the regions which are not qualified for the query. Query 6 in Fig. 1.b represents a false drop. *False drops* are semantic regions where the signatures are qualified for the query, but they should not have qualified. False drops are eliminated by further comparing the query with the regions. If false drops are numerous, the performance degrades dramatically.

Much work has been done on minimizing the false drop probability [14, 12]. It has been shown that in order to minimize the false drop probability, the expected number of zeros and ones in a signature must be the same [12]. When the length of the signatures increases – for the same number of distinct keywords in a region or query formula – the density of ones in the signatures decreases. The chance of getting false drops will decrease correspondingly. However, the storage overhead increases. If the signature length is F bits and the maximal number of terms in the query is t, the optimal number k_{opt} of bits set to one in a term signature is given as

$$k_{opt} = \frac{F \cdot \ln 2}{t}. \tag{1}$$

In information retrieval, the signature file method is a compromise between conventional inverted file and full-text scanning methods. The advantage of the method over an inverted file method lies in its moderate storage overhead. The storage overhead is some 10-20% for signature files compared to 100% and more for inverted files. On the other hand, the retrieval speed of the signature file method is higher than in the case of full scanning of text blocks, but lower than in the case of inverted files [15].

In semantic caching of Web queries, the number of terms in a query or region formula can vary. Still, the number remains small when compared to the number of words in a text block. Consequently, the signature storage overhead is again reduced when compared to the signature methods used in information retrieval. When different regions intersect and tuple duplicates are stored, unfortunately, there is another source of storage overhead in the case of semantic caching. In Sect. 4, we discuss this problem in some detail.

Cache organization. To process a query faster, our cache architecture maintains region signatures separately from region contents (see Fig. 2). Apart from a signature, each entry (region) in the signature part contains the region formula, the counter of tuples, the link to the corresponding region contents, and the value of a replacement function. Qualified regions are detected in the signature part. Once a semantic region is qualified for a full or partial answer, tuples in the region contents that match the query are reported to the user.

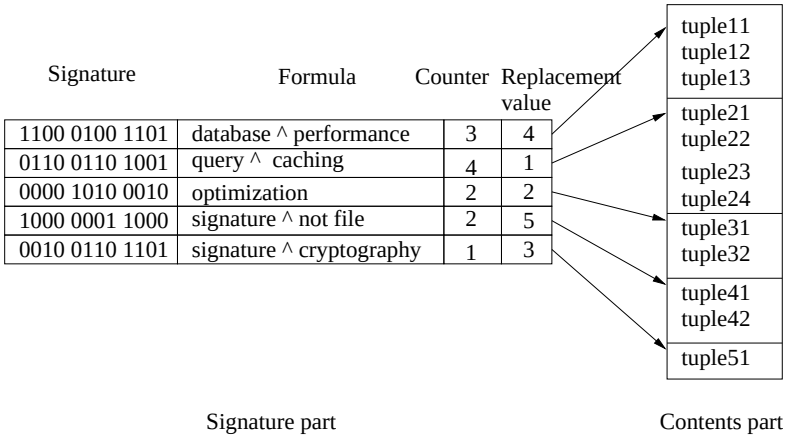

Fig. 2. Cache architecture

Negation. Any region formula contains keywords as well as their negations. To provide a smooth processing for queries with negations, signatures for a keyword and its negation can be related. A negated term is coded as a signature with a bit-wise negation of the basic term signature. However, as the number k of bits set to one in a term signature is much smaller than the signature length F, this would result in $F - k$ bits set to one in the negated term signature. Therefore, this solution would have a considerably higher false drop probability, for any region's formula containing the negated term. To avoid this problem, we treat a keyword and its negation (and their signatures) as two independent terms, with k bits set to one in both signatures.

3 A Simple Caching Algorithm

The first algorithm processes three cases of the semantic containment, namely, (1) equivalence, (2) query containment, and (3) region containment. If the query is equivalent to a region in the cache, the query answer coincides with the region contents. If a region contains the query, the complete answer can also be produced from the region contents, with the query formula used as a filter. Moreover, if two or more regions contain the query, any of them can produce the answer. To reduce the filtering overhead, the algorithm selects and filters the region where the content has the smallest number of tuples. In the case of region containment, the algorithm extracts a partial answer from the region contents and generates the query remainder which is then sent to the server. If several regions are contained in the query, any or all of them can produce the partial answer. As the number of such regions can be huge, the algorithm selects the top m regions with a maximal number of tuples.

If no semantic containment is detected, the cache is not used at all, and the initial query is sent to the server. When an answer to this query is received, a

new cache region is created. If the cache space is already used up, one or several "old" regions are released. As the basic replacement strategy, we use the well-known LRU ("least recently used"). The strategy is appropriate for the Web, where searching is always coupled with navigation and discovery. Typically, a new query refines a previous query [22].

With the algorithmic framework described above, three important issues require further analyses, namely, the construction of region remainders, region coalescing and cache region replacement.

3.1 Constructing query remainders

Assume that m semantic regions, $R_1, \ldots, R_m$, are contained in the query. Although the query remainder Q_r can be constructed as $Q_r = Q - R_1 - \ldots - R_m = Q \wedge \neg R_1 \wedge \ldots \wedge \neg R_m$, such a constraint formula, after simplification, can contain disjunctions, and may not be appropriate for a server that accepts conjunctive queries only. For example, for the query a and the region $a \wedge b \wedge c$[1], the constraint formula $a - a \wedge b \wedge c$ results in the following disjunction formula:

$$a - a \wedge b \wedge c = a \wedge \neg(a \wedge b \wedge c) = a \wedge \neg b \vee a \wedge \neg c.$$

To distinguish the regions which drive the query remainder to a conjunctive form from those which do not, we introduce a difference measure between the query and the region formulas. The *difference* is defined as the number of terms in the region formula not presented in the query. This definition splits the set of regions $R_1, \ldots, R_m$ into groups, where all regions in a group differ in l terms from the query, $l = 1, 2, \ldots$. In the example given above, the region formula $a \wedge b \wedge c$ has a two-term difference from the query a. Note, however, that the case $l = 0$ is also valid. When a query and a region are equivalent, or when a region contains a query, the difference is zero, and, correspondingly, the query remainder is null.

As stated in the following theorem, the difference measure allows us to guarantee that regions with one-term difference preserve the conjunctive form of the query remainder.

Theorem 1. *Assume a cache containing m region formulas have one-term differences, say $a_1, a_2, \ldots, a_m$, from a query formula Q. Then, the query remainder Q_r is $Q \wedge \neg a_1 \wedge \neg a_2 \wedge \ldots \wedge \neg a_m$.*

Proof. Let $\mathcal{E}$ denote the set of all possible query terms. As all m regions contain the query, we can denote the region formulas, without loss of generality, as $Q \wedge a_1$, $Q \wedge a_2$, ..., $Q \wedge a_m$.

Then we obtain:

$$Q_r = Q - (Q \wedge a_1) - \ldots - (Q \wedge a_m) = Q \wedge (\mathcal{E} - a_1 - \ldots - a_m) =$$

$$= Q \wedge (\mathcal{E} \wedge \neg a_1 \wedge \ldots \wedge \neg a_m) = Q \wedge \neg a_1 \wedge \ldots \wedge \neg a_m \square$$

[1] We use characters from the beginning of the alphabet to denote query terms.

3.2 Region coalescing

As we have explained before, when a query and a region are equivalent, or when a region contains a query, the query remainder is null. The query is not sent to a server but processed locally. The cache contents are kept unchanged. The region providing the answer to the query updates the replacement values (see Sect. 3.3).

In the region containment case, i. e. when a query contains a region, the query remainder is not null, and, moreover, it is a complement to a semantic region R (see Fig. 3.a). When an answer to the query remainder Q_r is received, there are two strategies to add the answer to the cache. With the *no-coalescing* strategy, a new cache region is created for the query remainder. This implies that smaller regions appear in the cache. However, such regions may result in the degradation of the cache reuse. With the *coalescing* strategy, no new cache region is added; instead, region R's contents are extended with the answer to the remainder. The region formula R is substituted (relaxed) with Q.

Obviously, both strategies require the same cache space to store tuples. Still, the coalescing strategy appears to be preferable. First, it uses only one region instead of two. Furthermore, if m semantic regions, $R_1, \ldots, R_m$, yield the region containment (see Fig. 3.b), the coalescing strategy is even more advantageous. The query remainder $Q_r = Q - R_1 - \ldots - R_m$ is a complement to the union of the regions. Here, the coalescing strategy will keep a single region (with formula Q), instead of m individual regions $R_1, \ldots, R_m$, and the query remainder. As regions $R_1, \ldots, R_m$ may contain tuple duplications, the coalescing strategy provides better storage utilization, both for the signature parts and for the content parts of the cache.

3.3 Replacement strategy

As the cache space is a scarce and limited resource, the cache may discard the least recently used regions to free space for new regions. The standard LRU strategy was designed for the replacement of pages or tuples in the cache. It acts upon objects of the same size, that is, the replacement unit is one page or one tuple.

In the query caching case, the situation is different. When a region R qualifies for a query, the involvement of the region in the answer can vary. If the query contains the region – as depicted in Fig. 3.a – the region contents is *completely* involved in responding, as all tuples from the region contents appear in the answer. By contrast, if the region contains the query (see Fig. 3.c), the region involvement is *partial*, as only some of the tuples in the region match the query.

Therefore, the replacement function should take into account the *region involvement* in the answer of the query. If the region involvement is complete, the new replacement value for the region is "the most recent one", as in the case when the answer to the query is shipped from the server. If the region involvement is partial, and there are tuples in the region contents not matching

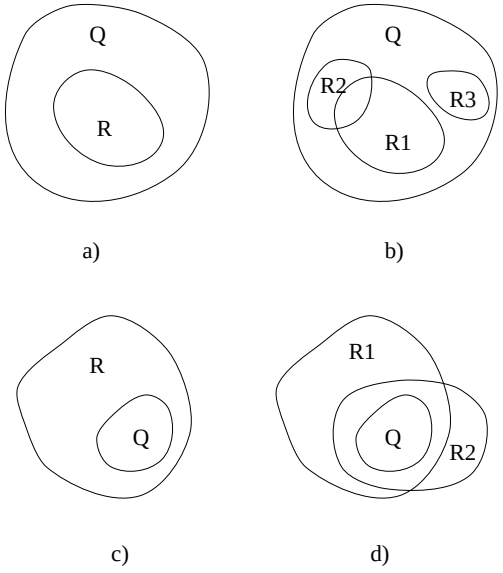

Fig. 3. Semantic containment cases: a) Single region containment; b) multiple region containment; c) single query containment; d) multiple query containment

the query, the change of the replacement value toward "the most recent one" depends on how large the portion of the matched tuples is.

The region involvement can be measured as $p = T_Q/T$, where T_Q is the number of tuples appearing in the answer to the query, and where T is the total number of tuples in R's contents.

Without loss of generality, we assume that "the most recent value", V_{top}, is incremented by one, each time a new query is issued. If the current replacement value of region R is V_R, $V_R < V_{top}$, and the region involvement is p, we calculate a new replacement function as $V_R^{'} = V_R + (V_{top} - V_R) \cdot p$. If $p = 1$, then $V_R^{'} = V_{top}$. If $p = 1/2$, then $V_R^{'} = (V_{top} + V_R)/2$. Note, that this replacement function can be implemented for any region in the cache, whether the region qualifies for the query or not. If a region does not qualify for the query, and, therefore, its involvement p is zero, the region replacement value is kept unchanged.

Example 2. The cache contains three regions with formulas $a \wedge b \wedge c$, $b \wedge d$ and $d \wedge \neg a$. Figure 4.a1 shows the regions with their replacement values (assuming $V_{top} = 6$). Assume a new query is d. The second and third regions yield the query containment. As both region formulas differ from the query formula in one word only (b for the second region and $\neg a$ for the third one), the generated query remainder is given as $d \wedge a \wedge \neg b$. Once the complete answer is produced, the second and third region, as well as the query remainder are substituted with one region with formula d (Fig. 4.a2). Its replacement value is $V_{top} = 7$.

Now we assume instead that the query is $b \wedge d \wedge f$ (see Fig. 4.b1). Two regions, $b \wedge d$, and $d \wedge \neg a$, contain the query. The former is selected as the answer to the query, as it has less tuples in the contents. Its replacement value is updated (from 3 to 4) accordingly to the portion of tuples matching the query in the region contents.

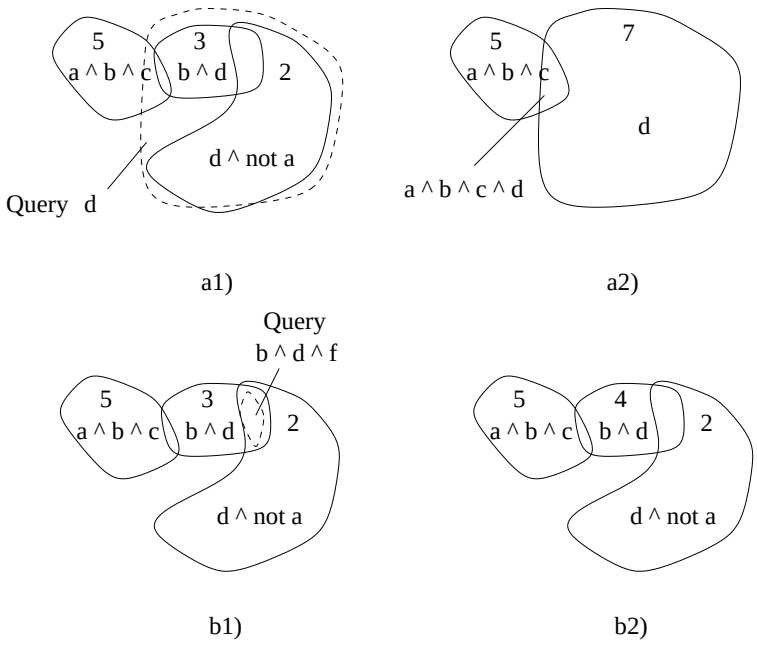

Fig. 4. Region coalescing examples: a1) Query $Q = d$ is issued; a2) Regions coalesced after the query; b1) Query $Q = b \wedge d \wedge f$ is issued; b2) Regions updated after the query

Caching algorithm 1. *Input*: cache with semantic regions and query Q.
Output: answer to Q and the cache updated.

1. Verify the query signature against all region signatures in the cache.
2. $S_Q = S_R$: If there is a region which formula is equivalent to the query, return the region contents as the query answer. Update the replacement function value of the region and stop.
3. $S_Q \supset S_R$: If one or more regions contain the query, choose the region with the minimal cardinality. Scan tuples in the region contents and return ones matching the query. Update the replacement function value of the region and stop.
4. $S_Q \subset S_{R_i}$: If several regions are contained in the query, choose top m regions, $R_1, \ldots, R_m$, with the maximal cardinality. Return all tuples from the

regions contents of $R_1, \ldots, R_m$, discarding duplications. Construct the query remainder as follows:

- Set the query remainder to query Q.
- For each region R_i providing the region containment, calculate the difference between the region formula and the query. If the difference is one term a_i only, constrain the query remainder with $\neg a_i$.

Send the query remainder to the server. When the answer is received, replace regions $R_1, \ldots, R_m$ with one region Q. Put V_{top} as the replacement value for Q and stop.

5. Otherwise, send query Q to the server. Once the answer is received, create a new region for Q in the cache. To free space for the region, remove the regions with the least replacement values, until query Q fits the cache.

4 The Advanced Caching Algorithm

The caching algorithm described in the previous section efficiently manages the semantic containment and equivalence cases. However, it does not manage the more frequent and complex case of semantic intersections. In a semantic intersection, a region can produce a portion of the answer, but it neither contains nor is contained in the query.

Example 3. Assume, the cache contains region R_1 with formula "query $\wedge$ caching" and query Q is "optimal $\wedge$ query" (see Fig. 5.a). Since there is no containment, Algorithm 1 does not exploit R_1 for the partial answer, although the tuples in the region contents matching the formula "optimal $\wedge$ query $\wedge$ caching" match also the query. Moreover, when Algorithm 1 receives the answer to query Q from the server, it creates a new semantic region R_2 with the same formula "optimal $\wedge$ query" (see Fig. 5.b). Two semantic regions R_1 and R_2 contain tuple duplicates, which match their intersection formula "optimal $\wedge$ query $\wedge$ caching". In other words, in the semantic intersection cases, Algorithm 1 retains a low cache use and a high tuple duplication level.

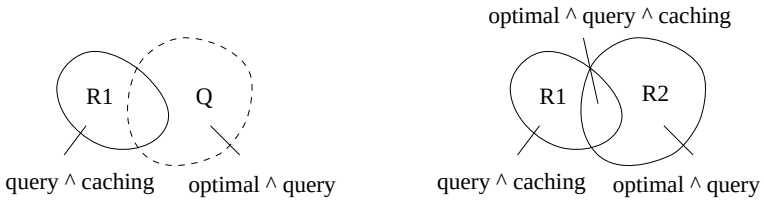

Fig. 5. Semantic intersection example

In this section we develop an advanced caching algorithm which, besides the containment cases, also processes the semantic intersection. This helps to improve cache utilization and to reduce tuple duplications.

The intersection of a semantic region R and a query Q is given by the intersection of their formula : $R \cap Q$. Given the region signature S_R and the query signature S_Q, we use their signature intersection $S_Q \cap S_R$, which is obtained by bit-wise AND-ing of S_Q and S_R. Then, for a signature S, the signature cardinality $|S|$ denotes the number of bits set to one in the signature.

In the semantic intersection of a semantic region R and a query Q, we distinguish two cases:

Complement: $Q \cap R = \emptyset$; the formula intersection is null. For instance, query $a \wedge b$ is a complement to the region $a \wedge \neg b$. Consequently, the region contains no tuples to answer to the query. However, in the complement case, the region coalescing is possible. For the query and region above, the coalescing would result in one region with formula a.

Intersection: The $Q \cap R \neq \emptyset$; the formula intersection is not null. There are two following sub-cases:

- Query and region formulas have some common terms appearing in the intersection (in Example 2, region "query $\wedge$ caching" and query "optimal $\wedge$ query" have term "query" in common).
- Query and region formulas have no common words. For instance, region a and query b have no common terms, but their intersection $a \wedge b$ is not empty, and, therefore, the region can contribute to the partial answer.

Semantic intersection in signature files. If the query Q and a region R have some common terms, their signatures have bits set to one which correspond to signatures of the common terms. The more terms formulas Q and R have in common, the larger the number of bits jointly set to one. The semantic intersection of Q and R could be measured as the number $|S_Q \cap S_R|$ of corresponding bits set to one in both signatures. Unfortunately, this is not always true. For example, even though the signature intersection of a region with formula a and a query b may have no bits set to one, the region with formula a might indeed have tuples matching the formula $b \wedge a$.

In the remainder of this section, we show how the signature file method allows for an efficient detection of region intersections with the query. Moreover, it supports partial answer deliveries and helps constrain the query remainder. Note that the following discussion about the semantic intersection assumes that neither equivalence nor query containment is detected in the cache, and, therefore, the query remainder is not null.

As in semantic containment, not all of the regions intersecting with the query can contribute to the query remainder; again because of the problem of conjunctive queries. To detect the regions that can contribute to a valid formula, we use again term differences, as introduced in Sect. 3. In addition, we make use of Theorem 1, which, while proven for the semantic containment only, also applies to the case of semantic intersection. We argue as follows: if a cache contains m

regions where the corresponding formulas are not contained in the query Q, but have one-term differences, say $a_1, a_2, \ldots, a_m$, the query remainder Q_r can be constructed as $Q \wedge \neg a_1 \wedge \neg a_2 \wedge \ldots \wedge \neg a_m$.

To use the theorem, we must revise one step in the proof which differentiates the semantic intersection from semantic containment. Indeed, with the semantic intersection, no region is contained in the query, and, therefore, no region formula can be presented as $Q \wedge a_i$. For the case $m = 1$, we have $Q_r = Q - R_1 = Q - Q \wedge R_1$. The constraint formula $Q \wedge R_1$ has one-term difference from query Q too, but is contained in Q. Hence, it can be represented as $Q \wedge a_1$. Therefore, $Q_r = Q \wedge \neg a_1$. The case $m > 1$ of the proof is derived in a similar way.

Example 4. As region "caching $\wedge$ query" has one-term difference from the query "optimal $\wedge$ query", the region can report the portion "optimal $\wedge$ query $\wedge$ caching" to the user and construct the query remainder "optimal $\wedge$ query $\wedge$ $\neg$ caching". Similarly, the region with formula a has one-term difference from query b. Therefore, the portion $a \wedge b$ is reported, and the query remainder is set to $b \wedge \neg a$.

This feature of semantic regions with one-term difference from the query in constraining the query remainder leads us to a double-scan evaluation of the query against the cache contents. The first, fast scan over the region signatures identifies all regions with one-term difference in order to quickly construct the query remainder and to produce the first partial answer. The second, slow scan checks whether other intersection cases can enrich the partial answer. The two scans over the region signatures differ in the filtering function applied to the region signatures.

Each region R filtered during the first scan should have at most one-term difference from the query. Therefore, if the region signature has $|S_R|$ bits set to one, and its intersection with the query signature has $|S_R \cap S_Q|$ such bits, the difference between the two numbers should be at most k bits, where k is the number of bits set to one in a term signature. The following theorem states this fact explicitly.

Theorem 5. *If region R has one-term difference from query Q, then*

$$|S_R \cap S_Q| \geq |S_R| - k. \tag{2}$$

The first scan verifies the condition (2) on the region signatures. If the condition holds for a region signature, the region formula is checked for a one-term difference. As in the case of semantic containment, a false drop appears if condition (2) holds but the region formula does not provide a one-term difference. In Sect. 5, we report the results of some experiments and show that the number of false drops when verifying the condition (2) can be kept small through some appropriate choices of signature file parameters, calculated using formula (1).

The second scan detects regions where the corresponding formulas differ in two and more terms. These regions do not qualify to constrain the query remainder. By analogy with one-term difference, a region R where the corresponding formula differs in $l, l \geq 2$ terms from the query, satisfies the condition

$$|S_R \cap S_Q| \geq |S_R| - k \cdot l. \tag{3}$$

However, this condition can not be used to full extent for the second scan. First, the condition (3) loses its importance for increasing values of k. In fact, a typical Web query or region formula has an average of three or four terms. Condition (3) is often reduced to a simple $|S_R \cap S_Q| \geq 0$. This would sweep all the region signatures, resulting in a large number of false drops, and a high filtering overhead. Second, regions differing in two or more terms from the query, usually contribute much less to the answer than regions with a one-term difference. Third, the tuples they contribute will be duplicated in the answer to query remainder, as their formulas were not excluded from the remainder. Therefore, the second scan can be omitted for some of the Web-based data repositories. For instance, if we know that regions with two-term difference contribute less than 1% to the partial answer (the Library of Congress discussed in Sect. 5 is such a repository), the query processing can stop after the first scan.

If the regions with two-term difference appear to be useful for partial answers, we consider two options for the second scan:

- $|S_R \cap S_Q| \geq |S_R| - 2k$: this option fetches mainly the regions with two-term difference from the query. Therefore, some regions differing in more terms will not be fetched.
- $|S_R \cap S_Q| \geq 0$: all region formulas satisfy this option, yielding to numerous false drops. However, *all* tuples in the cache matching the query are retrieved.

In most cases, the first option is more preferable as it provides a good tradeoff between the number of false drops and the number of tuples retrieved. The second option can be used if the cache space is very small, or if the application is keen to retrieve all tuples from the cache matching the query.

Region coalescing and region replacement. The semantic intersection gives a new extension to the coalescing strategy. The strategy can coalesce the query and a region when their unified formula is a conjunction. For instance, it can coalesce query $a \wedge b$ and the region $a \wedge \neg b$ in one region. Three conditions are sufficient: 1) the region has a one-term difference, say a_1, from the query; 2) symmetrically, the query has a one-term difference, say a_2, from the region; 3) a_1 is a negation of a_2[2].

The replacement policy, as designed for the semantic containment, remains the same for the semantic intersection. When a new query is issued, any semantic region in the cache has its replacement value updated, i. e. towards V_{top}, proportionally to the region involvement in the answer.

The second caching algorithm covers both relations between the query and semantic regions, that is, the semantic containment, as discussed in Sect. 3, and the semantic intersection, as discussed above. Moreover, the algorithm does not distinguish between regions providing the query containment and semantic intersection. Both cases are processed uniformly.

[2] Note, however, that a_1 and a_2 have independent signatures due to the cache architecture (see Sect. 2).

Caching algorithm 2. *Input*: cache with semantic regions and query Q.
Output: answer to Q and the cache updated.

1. (*First scan*) Check the query signature against the region signatures in the cache.
2. $S_Q = S_R$: if there is a region which formula is equivalent to the query, return the region contents as the answer. Update the replacement value of the region and stop.
3. $S_Q \cap S_R = S_R$: if one or more regions contain the query, choose the region with the minimal cardinality. Scan the region contents and return the tuples matching the query. Update the replacement value of the region and stop.
4. $|S_Q \cap S_R| \geq |S_R| - k$: Identify all regions, say $R_0, \ldots, R_m$, $m \geq 0$, with one-term difference from the query. Return the tuples matching the query in the semantic regions of $R_0, \ldots, R_m$, discarding duplications. Construct the query remainder Q_r as follows:
   - Set the query remainder to query Q.
   - For each region R_i, $i = 0, \ldots, m$, calculate the difference a_i from the query and constrain the query remainder with $\neg a_i$.

   Send the query remainder to the server.
5. (*Second, optional scan*) Scan the region signatures with the condition T, where T is a choice between $|S_R \cap S_Q| \geq |S_R| - 2k$ and $|S_R \cap S_Q| \geq 0$. For each region R fetched, check the $Q \cap R$ formula intersection. If the formula is not null, report the tuples from the region contents matching the query.
6. When the answer to the query remainder Q_r is received, update the cache as follows.
   - If regions $R_{i_1}, \ldots, R_{i_p}$ contain the query, replace them with a new region with formula Q.
   - If a region R is complement to query Q and formula $R \cup Q$ is a disjunction, substitute R and Q with a new region.
   - Otherwise, add a new region to the cache with the formula Q_r.

   Update the replacement values for all regions contributed to the partial answer.

Example 6. Assume the cache contains the regions with formulas $a \wedge b$, $c \wedge d \wedge \neg e$, and the user query be $b \wedge c$; Figure 6.a shows the regions with their replacement values (assuming $V_{top} = 7$) and the query. The first scan detects that the region $a \wedge b$ has a one-term difference from the query and can constrain the query remainder Q_r which is $b \wedge c \wedge \neg a$. All tuples from the region contents matching the query report their partial answer. The second scan detects the semantic intersection for region $c \wedge d \wedge \neg e$. The region contents is scanned and tuples matching this query complete the partial answer.

Once the answer to the query remainder Q_r is received, a new region with the formula $c \wedge d \wedge \neg e$ is created. The replacement value is set to $V_{top} = 8$. Also, both regions $a \wedge b$ and $c \wedge d \wedge \neg e$ have their replacement values updated, in proportion to their contribution to the answer (see Fig. 6.b).

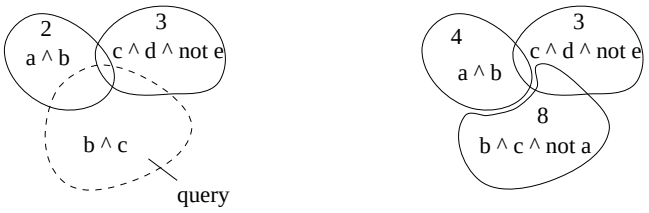

Fig. 6. Region coalescing for the semantic intersection: a) Query $c \wedge d$ is issued; b) after the query has been issued

5 Experiments

We have conducted a number of experiments to test the caching algorithms developed in the paper. As a Web information server, we have used the Library of Congress (about 6.3 million records) with the search page available at http://lcweb.loc.gov/. The search page supports one to three terms in the query; the first term must not be negated, while others can. Since no full-text retrieval is available, tuples are rather small with respect to the cache size. For any query, terms were randomly chosen from a dictionary containing some 80 terms in the field of computer science; these terms were taken from the Yahoo Classifier.[3]

Any query in the experiments contained one to three terms, with an equal probability for each case. If a second or third term was included, it was negated in one of three cases. Each algorithm tested in the experiments started with an empty cache and used the first queries just to fill it. Once the cache becomes full, the main parameters of the cache were evaluated during a series of s sequential queries. The main parameters, including values for F and k of the signature generation, are reported in Table 1.

Table 1. Experiment parameters

Parameter	Description	Value
S	Cache size	256k-1024k
F	Number of bits in signature	48-96 bits
k	Number of set bits for simple word	5-10 bits
s	Length of query series	100

In the experiments we have tested three main parameters:

Cache efficiency: the average portion of the answer provided from the cache. For one query, the efficiency is evaluated as r_c/r_t, where r_t is the total number

[3] http://www.yahoo.com/Science/Computer_Science/.

of answer tuples, and r_c is the number of the answer tuples retrieved from the cache. For a series of s queries, the cache efficiency is the mean of individual query efficiencies.

Duplication ratio: for one query it is evaluated as $(S - S_d)/S$, where S is the total cache size and S_d is the cache size when all tuple duplications are removed.

False drop ratio: the average number of false drops per query, taken over a series of s queries.

We have tested the following algorithms: Algorithm 1 (Sect. 3) and Algorithm 2 (Sect. 4, with the first scan only) combined with the coalescing[4] and no-coalescing strategies; in the graphs, they are named Coal-1, NoCoal-1, Coal-2 and NoCoal-2; The algorithms are tested over the parameters : (1) signature length F and the number k of bits set to one in a term signature; (2) cache size.

The following graphs summarize the experiment results.

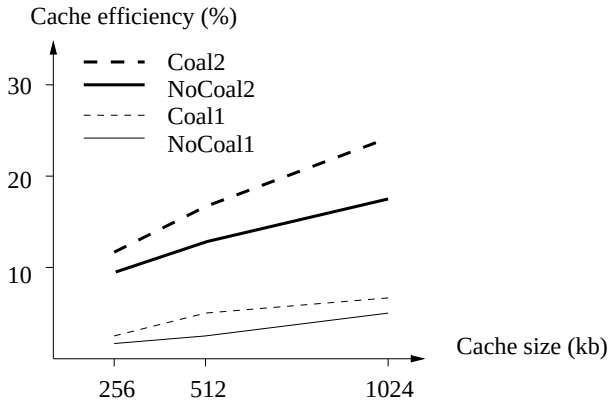

Fig. 7. Cache efficiency experiments

The cache efficiency grows almost linearly for all combinations, as the cache size increases (see Fig. 7). The efficiency is higher using Algorithm 2 as semantic intersection is more frequent than semantic containment. Similarly, the coalescing strategy works better than the no-coalescing strategy.

The duplication ratio graphs (see Fig. 8) demonstrate the difference between the coalescing and no-coalescing strategies. The ratio is higher applying the no-coalescing strategy which keeps different regions for queries which may semantically intersect. Algorithm 2 is slightly better – with respect to minimizing tuple duplications – than Algorithm 1 because it also detects query complements (though, this rarely happens).

[4] In all tests, the coalescing strategy was adopted so that regions are coalesced if the new region size is not superior to 10% of the total cache size.

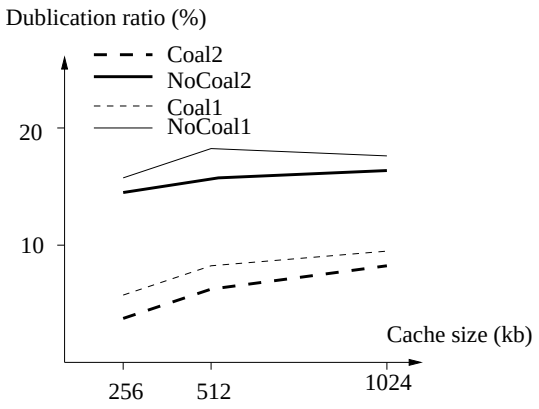

Fig. 8. Tuple duplication ratio in the cache

For all combinations of values F and k that determine the signature construction, Algorithm 2 gives a higher level of false drops than Algorithm 1 as shown in Fig. 9. To explain this fact, we recall that Algorithm 1 checks region signatures for two containment conditions, namely, $S_Q \cap S_R = S_Q$ and $S_Q \cap S_R = S_R$. Besides the same two conditions, Algorithm 2 also checks the condition (2) to detect all intersections with a one-term difference. Although the false drop ratio using Algorithm 2 is high for small values of F, it becomes reasonably low when F increases. We point out that the space overhead is kept low, since the main source of the space overhead is the tuple duplication in the content parts, and not the size of signature files.

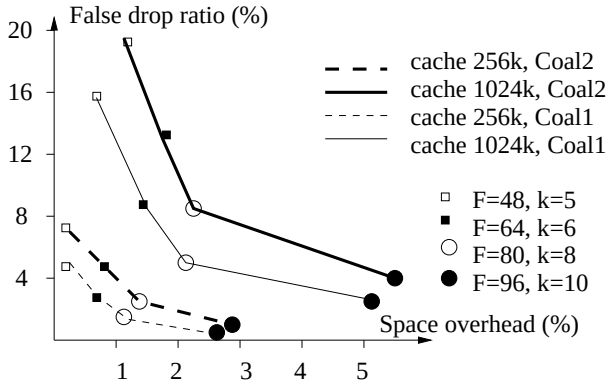

Fig. 9. False drops versus the space overhead

Our main conclusions from the experiments can be summarized as follows:

1. Algorithm 2 provides both a higher cache efficiency and a lower duplication ratio, when compared to Algorithm 1. The false drop ratio in Algorithm 2 is higher, but the difference is small. It can be neglected when using a slightly enlarged signature length.
2. The coalescing strategy is always better than the no-coalescing strategy when looking at the tuple duplications and the number of regions in the cache.
3. The experiments demonstrated the difference between the two major sources of space overhead in the cache, namely, the use of signature files and tuple duplications. For typical Web queries, the signature files do not occupy much space, allowing control of the false drop ratio by the appropriate values of F and k. However, tuple duplications can considerably reduce the cache efficiency. It is extremely worthwhile to reduce the duplication ratio.

6 Conclusion and Open Issues

We have presented a new caching mechanism for conjunctive Web queries as realized in the Knowledge Broker system [3]. The mechanism is based on signature files and allows for an efficient reuse of already obtained answers. Two caching algorithms were presented that cope with the relations of semantic containment as well as semantic intersection between a user query and the semantic regions, respectively.

The basic query model covers conjunctive queries only. With the superimposed coding used in signature files, the model cannot be extended to process disjunctive queries directly in the cache. Such a query must be split into conjunctions beforehand. A further analysis of signature file methods to overcome this problem is a real challenge.

The caching mechanism works efficiently for a single Web repository. In the case of a large set of different, possibly heterogeneous Web repositories, the cache management becomes more complicated. The attempt to put an additional constraint like "server=<repository-name>" does not solve the problem. It would change the query processing completely: the key element in the proposed caching mechanism is based on a one-term difference between the query and the semantic regions! Our plans are to study this problem so that we can adopt the signature method properly.

References

1. S. Adali, K. S. Candan, Y. Papakonstantinou, V. S. Subrahmanian. Query Caching and Optimization in Distributed Mediator Systems. In *Proc. SIGMOD '96 Conf.*, pp. 137–148, 1996.
2. R. Alonso, D. Barbara, H. Garcia-Molina. Data Caching Issues in an Information Retrieval System. In *ACM TODS* **15**: 3, 359–384, 1990.
3. J.-M. Andreoli, U. M. Borghoff, R. Pareschi. Constraint-Based Knowledge Broker Model: Semantics, Implementation and Analysis. In *Journal of Symbolic Computation* bf 21: 4, 635–667, 1996.

4. Y. Arens and C. A. Knoblock. Intelligent Caching: Selecting, Representing, and Reusing Data in an Information Server. In *Proc. CIKM '94 Conf.*, Gaithersburg, MD, pp. 433–438, 1994.
5. U. M. Borghoff, R. Pareschi, F. Arcelli, F. Formato. Constraint-Based Protocols for Distributed Problem Solving. In *Science of Computer Programming* **30**, 201–225, 1998.
6. M. J. Carey, M. J. Franklin, M. Livny, E. J. Shekita. Data Caching Tradeoffs in Client-Server DBMS Architectures. In *Proc. SIGMOD '91 Conf.*, pp. 357–366, 1991.
7. C.-C. K. Chang, H. Garcia-Molina, A. Paepcke. Boolean Query Mapping Across Heterogeneous Information Sources. In *IEEE TOKDE* **8**: 4, 1996.
8. C.-C. K. Chang and H. Garcia-Molina. Evaluating the Cost of Boolean Query Mapping. In *Proc. 2nd ACM Int'l. Conf. on Digital Library*, 1997.
9. S. Dar, M. J. Franklin, B. Jonsson, D. Srivastava, M. Tan. Semantic Data Caching and Replacement. In *Proc. 22nd VLDB Conf.*, Bombay, India, pp. 330–341, 1996.
10. C. Faloutsos. Signature files: Design and Performance Comparison of Some Signature Extraction Methods. In *Proc. SIGMOD '85 Conf.*, pp. 63–82, 1985.
11. C. Faloutsos and S. Christodoulakis. Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation. In *ACM TOIS* **2**: 4, 267–288, 1984.
12. C. Faloutsos and S. Christodoulakis. Description and Performance Analysis of Signature File Methods for Office Filing. In *ACM TOIS* **5**: 3, 237–257, 1987.
13. P. Godfrey and J. Gryz. Semantic Query Caching For Heterogeneous Databases. In *Proc. 4th KRDB Workshop on Intelligent Access to Heterogeneous Information*, Athens, Greece, pp. 6.1–6.6, 1997.
14. H. Kitagawa, J. Fukushima, Y. Ishikawa and N. Ohbo.. Estimation of False Drops in Set-valued Object Retrieval with Signature Files. In *Proc. 4th Int'l. Conf. FODO '93*, Chicago, IL. Springer-Verlag, LNCS **730**, 146–63, 1993.
15. D. L. Lee, Y. M. Kim and G. Patel. Efficient Signature File Methods for Text Retrieval. In *IEEE TOKDE* **7**: 3, 423–435, 1995.
16. A. Y. Levi, A. Rajaraman, J. .J. Ordille. Quering Heterogeneous Information Sources Using Source Descriptions. In *Proc. 22nd VLDB Conf.*, Bombay, India, pp. 251–262, 1996.
17. P. T. Martin and J. I. Russell. Data caching strategies for distributed full text retrieval systems. In *Information Systems* **16**: 1, 1–11, 1991.
18. A. Paepcke, S. B. Cousins, H. Garcia-Molina, et al. Towards Interoperability in Digital Libraries: Overview and Selected Highlights of the Stanford Digital Library Project. In *IEEE Computer Magazine* **29**: 5, 1996.
19. Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, J. Ullman. A Query Translation Scheme for Rapid Implementation of Wrappers. In *Proc. DOOD'95 Conference*. Springer-Verlag, LNCS **1013**, 161–186, 1995.
20. Y. Papakonstantinou, H. Garcia-Molina, J. Ullman. MedMaker: A Mediation System Based on Declarative Specifications. in *Proc. ICDE'96 Conf.*, pp.132–141, 1996.
21. Ch. Reck and B. König-Ries. An Architecture for Transparent Access to Semantically Heterogeneous Information Sources. In *Proc. Cooperative Information Agents*. Springer-Verlag, LNCS **1202**, 1997.
22. A. Yoshida. MOWS: Distributed Web and Cache Server in Java. In *Computer Networks and ISDN Systems* **29**: 8–13, 965–976, 1997.