Semantic Caching of Web Queries*

Boris Chidlovski¹, Uwe M. Borghoff²

¹ Xerox Research Center Europe, Grenoble Laboratory, 6, chemin de Maupertuis, F-38240 Meylan, France  
   email: chidlovski@xrc-e.xerox.com  
² Institut für Softwarentwicklung, Fakultät für Informatik, Univ. der Bundeswehr München, D-85577 Neubiberg, Germany  
   email: borghoff@informatik.unibw-muenchen.de

Received: date / Revised version: date

Abstract   In meta-searchers accessing distributed Web-based information 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. In this work, we study the problem of semantic caching of Web queries and develop a caching mechanism for conjunctive Web queries based on signature files. Our algorithms cope with both relations of semantic containment and intersection between a query and the corresponding cache items. We also develop the cache replacement strategy to treat situations when cached items differ in size and contribution when providing partial query answers. We report results of experiments and show how the caching mechanism is realized in the Knowledge Broker system.

1 Introduction

The World Wide Web has become de facto the largest collection of data. Literally, millions of Web sites publish data and allow users to search in their repositories. Nevertheless, efficient retrieval of relevant information remains a problem since Web repositories largely vary in the organization, the search policy and the publication methods of their data.

Data integration systems (DIS) try to overcome this problem by offering a unified interface that hides the heterogeneity of Web repositories. Since most Web repositories hide their data contents and allow an associative access through a search interface, so-called search service wrappers have been developed. Among other tasks, these wrappers convert user queries into the target source's query format, control the return flow of information (i.e., the answer stream from the servers), and deliver the answers in a uniform format to the user [8, 25, 27, 28].

As in any client-server system, high performance in such a networked environment is often reached by an efficient utilization of computational storage resources at the client sites. As data from remote servers are brought to clients on demand, local client memory is largely used to cache data in order to minimize future interaction with the servers.

This data caching is particularly important for Web repositories, as network traffic and overloaded servers can lead to long delays in the answer delivery. As standard page caching is not feasible on the Web, and tuple-caching exposes certain limitations, much effort has been spent to cache user queries and corresponding answers (instead of pages or tuples) for possible future re-use [5, 12, 18].

Query caching is particular advantageous when the user refines a query several times, for example, by adding or removing query terms. In this case, many of the answer documents may already be cached and can be delivered to the user right away. Moreover, if the answer can be composed from previous answer sets, the server would not be contacted at all. This is of some interest when data integration systems access payment sources that charge on a per-click or on a per-delivered-document basis.

Data caching and query caching. A Web-based data integration system differs from a standard client-server architecture where the transfer units between servers and clients are pages or tuple sets. When users navigate the Web, special proxy servers [2, 29] maintain the set of recently accessed Web pages and re-use a page from the cache each time its URL is asked by a client again. The caching mechanism used in this case is very close to the tuple caching used in operating database and information retrieval systems [3, 7, 23].

However, when Web repositories are searched with Boolean queries, this sort of caching does not work prop-
erly for two reasons. First, the user query does not contain the URLs of the answer pages. Instead it contains a filled-out search form. Once the server receives this form, the URLs of the answers are dynamically calculated by CGI-scripts. This makes the proxy cache worthless for pre-fetching the answer items. Second, there is no way to inform the repositories about qualified tuples in the client cache and thus reduce the cardinality of the answer set. Similarly, clients can not detect whether their local caches provide a complete answer set to the queries or not. As a result, clients are forced to ignore the cached tuples while performing the query. Once the query is sent to the servers and all qualifying tuples are returned, the clients detect and discard the duplications. As a result, cache proxies do not keep dynamically generated Web pages.

An alternative to the plain page caching is the semantic caching where the client cache is managed as a collection of semantic regions. Access information is maintained and cache replacement is performed at the unit of semantic regions only [12]. Semantic regions group together semantically related documents covered, for example, by a user query.

When a query is launched at a client, it is immediately split into two disjoint pieces: (i) a probe query that extracts the relevant portion of the answer set already available in the local cache, and (ii) a remainder query that fetches the missing (i.e., not cached) documents from the server. If the remainder query is not empty (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 [6].

Web queries posed against a singular Web source differ from queries in standard databases in that they neither contain JOIN operators nor its equivalents. Therefore, the problem of conjunctive Web query containment is — unlike the general case — algorithmically tractable; the problem is reduced to the set containment problem where the containment can be verified in time linear in the number of cached objects. Hence, semantic caching of Web queries can be positioned in the middle between the caching of complex database queries and the caching of simple Web data items. It combines, on one hand, the management of the containment/intersection relationships between a query and the relevant cache items and, on the other hand, the efficient cache management (replacement policy, etc).

There are two main classes of data integration systems, namely meta-searchers and mediator systems [19]. Meta-searchers assume the horizontal decomposition of data [24] among the Web repositories. Here, data stored at the repositories share a similar data schema. Queries in meta-searchers contain no joins and the benefit of mechanisms for Web query caching is immediate.

Mediator systems — unlike meta-searchers — assume the vertical decomposition of data [24] in the Web repositories and use join-like operators in order to combine data from sources with different data schemas [6, 22, 27]. However, as any mediator query is decomposed into a sequence of join-less sub-queries, the semantic caching of Web queries can be used at the lowest level of a mediator's multi-level cache, too. In either case, Web query caching is likely to be attached to a wrapper interrogating a particular Web repository.

In the following sections, we will consider the Web query caching mainly in the context of meta-searching as it brings an immediate performance gain. At the same time, we point out that the mechanisms proposed can be adapted to the class of mediator systems.

Related work. Intelligent query caching has been used in the SIMS project [5], where some important principles for any intelligent caching mechanism have been developed. These principles are as follows: (i) a query cache should process both containment and intersection cases; (ii) a cache item should not be large; (iii) a cache item should have a simple formula to avoid too complex reasoning on the remainder queries. These principles have been validated in various studies and our approach conforms to them, too.

Given a limited cache size, there is a strong tradeoff between the number of items in the cache and their size: larger cache items reduce the overall number of cache items, and thus, the cache re-use ratio. A semantic model for the query caching in a client-server architecture is discussed in [12]. It introduces the semantic query framework and all major features, such as semantic regions as well as probe and remainder queries. However, it applies the semantic caching mainly to storing range SQL queries posed against numerical attributes in a relational database. In this context, a cache region corresponds to a multidimensional cube. This reduces the semantic intersection or containment problem to the comparison of the attribute ranges. As a consequence, standard indexing techniques like B-trees can be used to maintain cache regions that provide an efficient detection of region containment and region intersection. In [13], multidimensional queries are divided in chunks before caching, this permits to lower the query granularity and to improve the cache performance.

Unfortunately, keyword-based Web queries are quite different from range SQL queries, and Web query caching requires a proper treatment, as discussed in [18] where the caching of Web queries is reduced to a Datalog query evaluation. Since this Datalog query evaluation allows full Boolean queries with operators AND, OR and NOT, the detection of semantic containment/intersection becomes computationally hard, thus considerably restricting the efficiency of the approach.

Query caching is also implemented in the Hermes distributed mediator system [1]. Caching SQL queries is based on the invariant mechanism where query rewriting techniques are used. Moreover, Hermes exploits semantic information about the repositories to collect some
source statistics and to build optimal query plans. However, the mechanism assumes the equivalence of subqueries and thus it does not consider the semantic containment/intersection cases.

The problem of re-using cached entries in the LDAP (Lightweighted Directory Access Protocol) network directories have been recently studied in [11]. The LDAP entries are organized into a hierarchical name-space and are accessed using positive conjunctive queries. As a first result, a sound and complete algorithm is proposed that determines whether such a conjunctive LDAP query can be satisfied using cached queries.

Our contribution. In this paper, we develop a mechanism for caching conjunctive Web queries. A conjunctive query allows the Boolean operators AND and NOT between query terms. Without the operator OR, the cache management becomes algorithmically tractable, i.e., we avoid the exponential complexity of the semantic containment and intersection problem for the full Boolean expressions. We employ the signature file method to efficiently detect and process all cases of semantic containment and intersection. Although not allowing the operator OR seems a strong limitation, the end user does not feel it that way. S/he is obviously allowed to formulate queries containing ORs. However, before such a user query is processed, in a pre-processing phase this query is split into several independent conjunctive queries that are handled separately [4]. The predominance of conjunctive queries on the Web prompts that such splits will not be overwhelming for the cache.

The cache contains semantic regions. Each 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 signatures of cached regions. The proposed caching mechanism includes a procedure that (i) detects all cache items qualified for the query, i.e., items that can be re-used immediately, and that (ii) identifies missing data that must be fetched from the servers.

Our proposal has the following advantages. First, it processes in a uniform way both cases where a query is either contained in the cache or where it intersects some regions. It supports the efficient reporting of partial answers and the direct construction of remainder queries. Second, our proposal solves the region coalescing problem. We adopt the standard LRU strategy to cope with regions of different sizes and involvement in probe queries. Third, we also consider the semantic caching for Web repositories with a particular behavior as far as the answer set generation is concerned, namely repositories that only provide incomplete answers (e.g., the n top-ranked documents relevant to the query). We study how the incompleteness of the answer set influences the cache management policy and propose an efficient solution. Finally, we propose a signature hashing method to detect regions qualified for a query in a sub-linear time. To validate our algorithms, we have tested the semantic caching mechanism based on signature files with real Web repositories. Although the main motivation and the targeted use of our proposed algorithms are Web-based meta-searching, we point out that the presented approach can be adapted to full-fledged distributed database environments.

The work reported in this paper extends [9]. New is the treatment of incomplete answer sets that are frequent in real Web engines. Additionally, we also conducted a much larger set of experiments to evaluate the proposed caching mechanisms more thoroughly.

The remainder of the paper is organized as follows. Section 2 introduces the Web queries. It discusses the semantic containment as well as the intersection cases between a query and the semantic regions. Moreover, we discuss the underlying cache architecture based on semantic regions and their signatures. Section 3 studies the construction of the probe and the remainder queries. In Sect. 4, we discuss some cache replacement strategies together with the region coalescing. A first caching algorithm is presented. In addition, we briefly discuss the case of incomplete answers and present an adapted caching algorithm. Experimental results of the caching algorithms are reported in Sect. 5. Section 6 discusses some open issues and concludes the paper.

2 Semantic Caching in Meta-searchers

In this section, we describe the main components of semantic caching. We introduce Web queries, the cache architecture and a method for the signature generation.

2.1 Web queries

To find relevant information on the Web, users query different search engines. In the following we look at user queries formulated as extended Boolean formulas, where a formula is a conjunction of terms.

Each query term \( t \) has the form \( \text{Attr Op Value} \), where \( \text{Attr} \) is an attribute name specific for a particular Web source (like Title, Author and Abstract for a bibliographic site or Patent Number for a patent database). \( \text{Op} \) is an operator, such as Contains or Equals, and \( \text{Value} \) is a keyword. In a conjunctive query formula \( Q \), at least one term is not negated. Without loss of generality, we assume that the first term is not negated and, therefore, a query \( Q \) has the following form: \( Q = t_1 \land t_2 \land \ldots \land t_I \), where \( t_i = t_i \lor t_i = \neg t_i, i = 2, \ldots, I \).

Hereafter, we call such formulas Web queries. A query example is \( Q = \text{Title Contains 'Cache' \land \neg Author Equals 'Dar'} \). Two terms \( t_1 \) and \( t_2 \) are equivalent if their attributes, operators and values coincide. For the sake of simplicity, we will often omit attributes and operators and present queries as simple conjunctions of keywords, for example, \( \text{('Cache \land \neg Dar')}. \)
Note however that Web queries refer to one search engine only and, therefore, no query can contain any sort of predicates or variables. Although this makes our Web queries a proper subset of a more general set of queries, like Database queries [4,18], it brings in two essential benefits. First, the heterogeneity of Web search engines makes the caching adapted to a single search engine more natural. Since Web engines differ in their search interface and functionality, caching often requires query translation that makes cross-engine caching rather cumbersome. Second, reducing Web queries to conjunctive Boolean formulas only makes all cache algorithms treatable in linear or even sub-linear time.

**Proposition 1** Let W denote the set of Web queries defined above. Then, W is closed under intersection, and not closed under union and difference.

### 2.2 Semantic cache architecture

The query cache manages a collection of semantic regions grouping together semantically related data, such as the answers to a user query. In any semantic region, we distinguish between the region descriptor and the region contents (see Fig. 1). The region descriptors occupy a small portion of the cache. They are stored in a cache dictionary, separately from region contents. The descriptors are used to detect regions relevant to the query. Then, the contents of those regions will be accessed to retrieve answer tuples. As tuples in a region content can be filtered with a user query, they have the same format as tuples in the entire system, for instance, the Object Exchange Model format [26].

A region descriptor includes the following elements:

- **Region signature**: A binary code assigned to the region formula which allows for a fast comparison among conjunction formulas.
- **Region size**: The size of a region that can vary over time.
- **Replacement value**: This value is used by the replacement strategy to free cache space for a new query. In the simplest case, this value is a counter that reflects the access times for the cached items; see Sect. 4.2 for more details.
- **Pointer to the region content**: A pointer that links the region descriptors with the data structure where the content of the regions is stored.

When a user formulates a query \(Q\), the meta-searcher checks it first against the content of the semantic cache. The cache splits the user query into two portions, a probe query and a remainder query.

The probe query \(Probe(Q)\) represents a partial answer to the query provided by the cache. It addresses those cache regions that contain tuples satisfying the query and thus contribute to the answer.

The remainder query \(Rem(Q)\) refers to data not contained in the cache (but still relevant to the query) that must be fetched from the remote Web servers over the network. If the remainder query is empty, the probe query completely satisfies the user query. The remote Web servers are not contacted at all. If the remainder query is not empty, its evaluation proceeds in a regular way. The answer to the user query is built up from the answers to the probe and the remainder queries, i.e., \(Q = Probe(Q) \cup Rem(Q)\).

For example, if the cache contains the region \(R = a \land b\) and the query is \(Q = a\), the probe query intersects with \(R\). Hence, we construct a probe query \(Probe(Q) = a \land \neg b\) and a remainder query \(Rem(Q) = a \land b\).

If the probe and the remainder queries are disjoint, the remainder query can simply be constructed from the query and the probe query as \(Rem(Q) = Q \land \neg Probe(Q)\).
In a more general case, however, this is not possible. The probe query \( \text{Probe}(Q) \) does not necessarily require to be a conjunction. If some regions \( R_1, \ldots, R_m \) qualify for a query (we will formally define this below), the probe query is constructed as \( \text{Probe}(Q) = Q \land R_1 \lor \ldots \lor Q \land R_m = Q \land (\bigvee_{i=1}^m R_i) \). Instead, any remainder query should be a conjunction since it may be stored, upon its arrival from the Web server, as an independent cache region. As Web queries are not closed under union and difference (see Proposition 1), a direct application of \( \text{Rem}(Q) = Q \land \neg \text{Probe}(Q) \) for the remainder queries becomes problematic. In Sect. 3, as a first solution we properly choose regions qualified for a query in order to keep probe and remainder queries disjoint. In Sect. 4.4 we study the more complicated case where Web repositories return the \( n \) top-ranked answers only, i.e., not the full answer set. Here, we allow an intersection of the probe and the remainder queries when the remainder query fetches missing tuples (already reported by the probe query) from the remote Web servers.

Beyond the processing of regions against a query, another important issue is the management of semantic regions involving coalescing and replacement strategies. The coalescing strategy determines how to merge/split the regions in order to provide the optimal granularity of the cached items. The replacement strategy (see Sect. 4) specifies a policy how to discard some cached regions when cache space is needed for a new query.

2.3 Semantic containment and intersection

Our caching algorithms cope with the semantic containment and intersection between the user query and cache regions. The semantic containment holds between the query and a cache region when one contains the other. In the semantic intersection, neither the region contains the query nor vice versa.

Because of the conjunctive form, we often consider any query or region formula as a set of terms and apply standard set operations to define and process the containment and intersection cases. In this context, \( Q \supset R \) means that query \( Q \) will always produce a superset of the answers w.r.t. region \( R \), i.e., query \( Q \) has a smaller term set (is less specific) than region \( R \). As an example, consider the query \( Q \) with formula ‘‘query \land caching’’ and a region \( R \) with formula ‘‘optimal \land query \land caching’’.

Analogously, \( Q \subset R \) means that query \( Q \) will always produce a subset of the answers w.r.t. region \( R \), i.e., query \( Q \) has a larger term set (is more specific and prunes the search space better) than region \( R \).

Using this notation, we distinguish three cases.

1. As depicted in Fig. 2.a, a query \( Q \) is equivalent to a region \( R \), \( Q = R \), if their Boolean formulas are equivalent.
2. As depicted in Fig. 2.b (single region containment), a query \( Q \) contains a (single) region \( R \), \( Q \supset R \), if the set of \( Q \)'s terms are properly contained in the set of \( R \)'s terms. Figure 2.c reflects the more general case of multiple region containment. As both query and region are conjunctions of terms, the query formula can be obtained from the region formula by dropping one or more terms. Recall that, the query formula is less specific than the region formula and always produces a superset of answers w.r.t. the region.
3. Inversely, as depicted in Fig. 2.d (single query containment), a query \( Q \) is contained in a (single) region \( R \), \( Q \subset R \), if the region formula can be obtained from the query by dropping one or more query terms (i.e., the query formula is more specific than the region formula). Figure 2.e reflects the more general case of multiple query containment.

In any of the above cases, the region \( R \) is qualified for query \( Q \).

![Fig. 2 Semantic containment cases: a) equivalence; b) single region containment; c) multiple region containment; d) single query containment; e) multiple query containment](image)

In Web searching, the semantic containment is less frequent than the semantic intersection (i.e., a region neither contains nor is contained in the query). However, such a region can contribute to the probe answers as well. Consider the following example.

**Example 1** Let the cache contain a region \( R \) with formula ‘‘query \land caching’’ and let the query \( Q \) be ‘‘optimal \land query’’ (see Fig. 3). The semantic containment does not exploit \( R \) for the partial answer, although the tuples in \( R \)'s contents matching the formula ‘‘optimal \land query \land caching’’ also match the query. Moreover, creating a new semantic region \( R' \) for query \( Q \) results in two semantic regions \( R \) and \( R' \) containing...
tuple duplicates which match their intersection formula \textit{"optimal \land query \land caching"}. In other words, considering the semantic intersection cases only retains a low cache re-use and a high tuple duplication level.

Hence, processing the semantic intersection in the cache – beyond the semantic containment – improves the cache utilization and reduces tuple duplications. The intersection of a semantic region \( R \) and a query \( Q \), i.e., \( R \cap Q \), is given by the intersection of their formulas. We have to distinguish two cases:

\textit{Complemend:} \( R \cap Q = \emptyset \), i.e., the formula intersection is empty. As an example, query \( Q = a \land b \) is the complement of the region \( R = a \land \neg b \). Consequently, the region contains no tuples to answer the query. Instead, the region coalescing is possible. For the query and region above, the coalescing will result in one region with formula \( a \).

\textit{Intersection:} \( R \cap Q \neq \emptyset \), i.e., the formula intersection is not empty and, therefore, \( R \)'s content can contribute to the answer of \( Q \). Here, there are two possibilities:

- Query and region formulas have some common terms appearing in the intersection (for example, query \( Q = a \land b \) and region \( R = a \land c \) have the common term \( a \)).
- Query and region formulas have no common terms. For instance, region \( b \) and query \( c \) have no common terms, but their intersection \( b \land c \) is not empty, and, therefore, the region can contribute to the partial answer.

Unlike semantic containment, a user query can intersect nearly all regions in the cache while not all of them are worthwhile to consider, because of their negligible contribution to the probe query. The situation is complicated by the existence, like in semantic containment, of certain limitations in generating remainder queries. In Sect. 3, we will study this issue in detail and exploit a particular case of semantic intersection, namely the so-called one-term difference regions that are beneficial in both contributing to probe queries and constraining remainder queries.

\subsection{2.4 Query signatures}

User queries have different lengths, so do the cache regions’ formulas. To quickly process a query against the cache regions, we assign binary signatures to both query and regions and determine qualified regions by signature comparisons.

Signature methods were originally designed for the retrieval of full-text fragments containing query words \([14, 15]\) and re-used later in other applications \([16, 21]\). The conjunctive form of query and region formulas helps us to use the \textit{superimposed coding} technique to assign signatures to formulas. Each term in a region formula is assigned with 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 4.a shows the signature generation for the semantic region \textit{"query \land 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 – again in a bit-wise mode – matched against each region signature in the signature file to provide a partial answer and construct the remainder query.

Let \( S_Q \) and \( S_R \) denote a query signature and a region signature, constructed as described above. With the bit-wise comparison of the signatures, the semantic containment and intersection cases are detected as follows:

\textit{Region containment:} For each bit in the query signature \( S_Q \) set to one, the corresponding bit in the region signature \( S_R \) is also set to one (see Query 2 in Fig. 4.b). Recall again that, in this case, the query \( Q \) has a \textit{smaller term set} (less bits are set to one) than region \( R \), i.e., query \( Q \) will always produce a superset of the answers w.r.t. region \( R \).

\textit{Equivalence:} The region and the query signatures have the same bits set to one, i.e., \( S_Q = S_R \) (see Query 3 in Fig. 4.b).

\textit{Query containment:} For each bit in the region signature \( S_R \) set to one, the corresponding bit in the query signature \( S_Q \) is also set to one (see Query 4 in Fig. 4.b).

\textit{Intersection:} Given the region signature \( S_R \) and the query signature \( S_Q \), we will use their signature intersection \( S_Q \land S_R \), which is obtained by bit-wise AND-ing of \( S_Q \) and \( S_R \), and its cardinality. Here, for a signature \( S \), the signature \textit{cardinality} \(| S |\) denotes the number of bits set to one in the signature.

A signature method eliminates most, but unfortunately not all of the regions which are not qualified for the query. Query 6 in Fig. 4.b shows a \textit{false drop}, that is a non-qualified semantic region whose signature is qualified for the query. False drops can occur when different term signatures superimposed with bit-wise AND produce the same signature. False drops are eliminated by further comparing the query formula with the region formulas. If false drops are frequent, the performance degrades dramatically. To minimize the false drop probability, ones should be uniformly distributed in a term
signature [16]. When the length of the signatures increases – for the same number of distinct keywords in a region or a query formula – the density of ones in the signatures decreases. The chance of getting false drops will decrease accordingly. However, the storage overhead increases at the same time. For false drops in the semantic containment, [20] establishes the optimal tradeoff between signature length $F$ and the number $k$ of bits set to one in a term signature. If a query contains $t$ terms and the signature length is $F$, the lowest false drop ratio is achieved when $k$ is given by $k_{opt} = F^{\log_2 t}$.

Unfortunately, in caching Web queries, the above formula is not immediately applicable. First, the number of terms in queries can vary. Second, the cache copes with both semantic containment and intersection. In Sect. 5, we will show how these differences influence the value of $k$ in providing a low false drop ratio.

**Negated terms.** Any region formula contains terms as well as their negations. When generating signatures for negated terms, one may want to extend the relationship of complement between a term and its negation to their signatures. In this case, a negated term’s signature might be built up as a bit-wise negation of the basic term signature. This solution, however, is unsatisfactory. As the number $k$ of bits set to one in a term signature is much smaller than the signature length $F$, $k << F$, the solution would result in $F-k$ bits set to one in the negated term signature and, therefore, a high false drop probability for any region’s formula containing the negated term. To keep the false drop probability low, we assign two independent signatures to a term and its negation, with $k$ bits set to one in both signatures.

### 3 Probe and Remainder Queries

In the following section, we develop replacement and coalescing strategies for the semantic cache. The study of the generation of the probe and the remainder queries starts with the semantic containment case.

#### 3.1 Semantic containment

In the cases of equivalence as well as query containment, the construction of the probe and the remainder queries is straightforward. The probe query $\Probe(Q)$ addresses a cache region $R$ containing the query result $(\Probe(Q) = Q \land R)$; the remainder query $\Rem(Q)$ is empty. In the case of region containment, however, the construction is not trivial and requires more analysis.

Assume that $m$ semantic regions, $R_1, \ldots, R_m, m \geq 1$, are contained in the query (see Fig. 2.c). Although they all can be used in the probe query, only some will be taken into account when constructing the remainder query. As we have seen before, if the probe query $\Probe(Q)$ is set to $Q \land (\bigvee_{i=1}^m R_i)$, the remainder query $\Rem(Q)$ constructed as $\Rem(Q) = Q \land \neg \Probe(Q) = Q \land \neg R_1 \land \ldots \land \neg R_m$, after simplification, can contain conjunctions! To distinguish the regions which preserve the conjunctive form of the remainder query from those which do not, we introduce a difference measure between the query and the region formulas.

**Definition 1** Let $Q$ be a user query and $R$ be a cache region contained in the query. The difference of a region $R$ from a query $Q$ is the set of terms in $R$’s formula not present in $Q$’s formula.

As an example, the region formula $R = a \land b \land c$ has a two-term difference from the query $Q = a$.

The difference measure is not symmetric. Further, it partitions the set of regions $R_1, \ldots, R_m$ yielding the semantic containment into disjoint groups, where all regions in one group have $l$-term difference from the query, where $l = 0, 1, 2, \ldots$. The case $l = 0$ comprises the equivalence and query containment cases, while cases $l = 1, 2, \ldots$ represent the region containment. Among
the latters, the regions with one-term difference play the key role. As the following theorem proves, they preserve the conjunctive form of the remainder query.

**Theorem 1** Let regions \( R_1, \ldots, R_m, m \geq 1 \) in the semantic cache have one-term differences from a query formula \( Q \), with the difference terms being \( a_i, i = 1, \ldots, m \). Then the remainder query is \( \text{Rem}(Q) = Q \land \neg a_1 \land \neg a_2 \land \ldots \land \neg a_m \).

*Proof.* As the query contains all \( m \) regions (multiple region containment), we can denote the region formulas, without loss of generality, as \( Q \land a_1, Q \land a_2, \ldots, Q \land a_m \). Then, by applying \( m \) times the rule \( a \land \neg(a \land b) = a \land \neg b \), we obtain:

\[
\text{Rem}(Q) = Q \land \neg \left( \bigvee_{i=1}^{m} R_i \right) = Q \land \neg (Q \land a_1) \land \ldots \land
\neg (Q \land a_m) = Q \land \neg a_1 \land \ldots \land \neg a_m. \]

Beyond preserving the conjunctive form of the remainder query, regions with one-term difference have another advantage over regions with two-or-more-term difference. Let \( Q \) be a query and let \( R_1 \) and \( R_2 \) be regions having one and two-term difference from \( Q \), respectively. Whatever the difference terms are, region \( R_1 \) contributes on average more to the probe answer than \( R_2 \).

Indeed, any term \( t \) in a conjunctive formula \( F = F' \land t \) has a selectivity ratio \( S_F(t) \) evaluated as \( |F'| / |F| \), where \( |F| \) and \( |F'| \) are cardinalities of answers to \( F \) and \( F' \), \( 0 \leq S_F(t) \leq 1 \). On average, a positive term is more selective than a negated term, and two terms have a higher total selectivity than one term. Therefore, the region \( R_2 \) with two terms of difference has on average a smaller cardinality than region \( R_1 \) and contributes less to the partial answer of \( Q \). Because of this double advantage, we will exclude regions with two-or-more-term difference and use only regions with one-term difference in the construction of both probe and remainder queries. In Sect. 5, we report a set of experimental results that confirm our decision.

### 3.2 Semantic intersection

Processing the semantic intersection in a semantic cache raises two issues concerning the construction of the probe and the remainder queries. First, too many regions intersect a user query but very few of them really contribute to the probe query \( \text{Probe}(Q) \). Second, as with semantic containment, not all regions intersecting the query cache can constrain the remainder query while keeping it conjunctive at the same time. Similar to the case of region containment, we re-use the term differences as introduced in Sect. 3.1 to group regions and take into consideration only the regions having a one-term difference from the query. Below, we extend Theorem 1 to the case of semantic intersection.

**Theorem 2** Let the regions \( R_1, \ldots, R_m, m \geq 1 \) in the semantic cache be not contained in the query \( Q \) but have one-term differences from \( Q \) and let the term difference of \( R_i \) from \( Q \) be \( a_i, i = 1, \ldots, m \). Then the remainder query is \( \text{Rem}(Q) = Q \land \neg a_1 \land \neg a_2 \land \ldots \land \neg a_m \).

*Proof.* We re-use Theorem 1 and revise a step in the proof that differentiates the semantic intersection from the semantic containment. With the semantic intersection, no region is contained in the query, and, therefore, no region formula can be presented as \( Q \land a_1 \). For the case \( m = 1 \), we denote with \( Q_{R_1} \) the intersection of \( Q \) and \( R_1 \):

\[
Q_{R_1} = Q \land R_1.
\]

Then, we can represent the region formula as \( R_1 = Q_{R_1} \land a_1 \) and the query as \( Q = Q_{R_1} \land Q_{\text{tail}} \). Then the remainder query is \( \text{Rem}(Q) = Q \land \neg R_1 = Q_{R_1} \land Q_{\text{tail}} \land \neg (Q_{R_1} \land a_1) \). As \( Q_{\text{tail}} \) does not contain \( a_1 \), we immediately obtain:

\[
\text{Rem}(Q) = Q_{R_1} \land (Q_{\text{tail}} \land \neg a_1) = Q \land \neg a_1.
\]

The case \( m > 1 \) of the proof is derived in a similar way. \( \square \)

**Example 2** As the region \( \{\text{query} \land \text{caching}\} \) in Example 1 has a one-term difference from query \( \{\text{optimal} \land \text{query}\} \), the region can report the portion \( \text{Probe}(Q) = \{\text{optimal} \land \text{query} \land \neg \text{caching}\} \) to the user and construct the remainder query

\[
\text{Rem}(Q) = \{\text{optimal} \land \text{query} \land \neg \text{caching}\}.
\]

Similarly, the region with formula \( b \) has a one-term difference from query \( c \). Therefore, the portion \( b \land c \) is reported, and the remainder query is set to \( c \land \neg b \).

**One-term difference in signatures.** The way we designed region and query signatures in Sect. 2.4 permits to determine the semantic containment from the signature containment. It is different, however, from the case of semantic intersection. If a 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 \( |S_Q \land S_R| \) of corresponding bits set to one in both signatures. Unfortunately, this is not always true. For example, 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 \( b \land a \).

The following theorem proves an important inequality for the signatures of regions having one-term difference from a query formula. As before, let \( k \) denote the number of ones in a term signature.

**Theorem 3** Let region \( R \) have a one-term difference from query \( Q \). Then \( |S_R \land S_Q| \geq |S_R| - k \).

*Proof.* The region signature has \( |S_R| \) bits set to one, and its intersection with the query signature has \( |S_R \land S_Q| \) such bits. The region signature \( S_R \) is obtained by bit-wise AND-ing. Therefore, the difference between the two numbers should be at most \( k \) bits. \( \square \)
Note that the above theorem can be used for both cases of one-term difference, namely, when a region \( R \) is contained in and when it intersects a query \( Q \).

4 Cache Management

Residing regions in a semantic cache are managed by the coalescing and replacement strategies. The region coalescing strategy controls the region granularity while the replacement strategy decides which regions to discard from the cache to accommodate new queries.

4.1 Region coalescing

Consider again the region containment case (see Fig. 2.b) and assume that region \( R \) has a one-term difference from query \( Q \). The remainder query \( Rem(Q) \) is a conjunction, and upon reception of an answer from the remote Web server with regard to \( Rem(Q) \), there are two options of how to incorporate the answer into the cache structure. With the non-coalescing strategy, \( Rem(Q) \) gets stored in a new region. This strategy implies that also smaller regions appear in the cache, thus increasing the overall number of regions in the average. With the coalescing strategy, no new cache region is added. Instead, region \( R \)'s content is extended with the answer of \( Rem(Q) \), i.e., the region formula \( R \) is substituted (relaxed) with \( Q \).

None of the two strategies appears to be obviously preferable. The non-coalescing strategy improves the region granularity by keeping (a high number of) small regions. However, since negated and positive terms have different selectivity, two regions \( a \land b \) (as the original region) and \( a \land \neg b \) (as the remainder query) often split the content of \( a \) (as a query) in unequal parts, with the bigger one \((a \land \neg b)\) being on average more than 90% of \( a \)'s size thus making problematic the advantage of the region split.

Moreover, if \( m \) semantic regions, \( R_1, \ldots, R_m, m > 1 \), are contained in a query (i.e., multiple region containment; see Fig. 2.c), the coalescing strategy is obviously advantageous. With the remainder query \( Rem(Q) = Q \land \neg R_1 \land \ldots \land \neg R_m \), this strategy creates a single region (with formula \( Q \)) instead of \( m \) individual regions \( R_1, \ldots, R_m \), and the remainder \( Rem(Q) \).

As regions \( R_1, \ldots, R_m \) may contain tuple duplications, the coalescing strategy results in a better storage utilization. In sum, the lower region granularity when applying the non-coalescing strategy is often compensated by a better storage utilization and a faster processing when compared with the coalescing strategy.

The semantic intersection does not change the non-coalescing strategy. It continues to create a new region for \( Rem(Q) = Q \land \neg R_1 \land \ldots \land \neg R_m \), where \( R_0 \) are regions with one-term differences. Instead, it gives a new extension to the coalescing strategy. The strategy can coalesce the query and a region if their unified formula is a conjunction. For instance, it can coalesce query \( a \land b \) and the region \( a \land \neg b \) in one region. Three conditions are sufficient for the coalescing: (i) the region has a one-term difference, say \( a_1 \), from the query; (ii) symmetrically, the query has a one-term difference, say \( a_2 \), from the region; (iii) \( a_1 \) is a negation of \( a_2 \). Note however that \( a_1 \) and \( a_2 \) have independent signatures due to the cache architecture (see Sect. 2.4). Thus this coalescing case is determined by a direct formula comparison once Theorem 3 for signatures has been verified.

4.2 Replacement strategy

As the cache space is a scarce and limited resource, the cache may discard some regions to free space for new regions. The standard LRU strategy designed for the replacement of objects in the data cache acts upon objects of the same size, that is, the replacement unit is one page or one tuple [3, 7].

The semantic cache with Web queries is different from the data cache in two aspects. First, cached queries are not independent like objects in a data cache. One region can provide a full answer, a partial answer, or nothing. Second, regions have different size. In the following subsections, we will modify the basic LRU strategy in order to accommodate for the new aspects.

Region involvement in answers. When a region \( R \) qualifies for a query, the region contribution to the answer can be different. If the query contains (or is equivalent to) a region (see Fig. 2.a-2.c), the region contents is completely involved in responding as all tuples from the region contents appear in the answer. In contrast to that, if a region contains the query (see Fig. 2.d-2.e) or intersects it (see Fig. 3), the region involvement is only partial, as only some of the tuples in the region content match the query.

To cope with these varying contributions, 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 the query at all, the change of the replacement value (i.e., its upgrade) toward “the most recent one” depends on the size of the portion of matching tuples.

We measure the region \( R \) involvement as \( p = T_Q / T \), where \( T_Q \) is the number of tuples matching 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”, \( V_{\text{op}} \), is incremented by one, each time a new query is launched. If the current replacement value of region \( R \) is \( V_R \), \( V_R < V_{\text{op}} \), and the region involvement in a new query \( Q \) is \( p \), we calculate a new replacement function as \( V'_R = V_R + (V_{\text{op}} - V_R) \cdot p \). If \( p = 1 \),
then $V_R' = V_{top}$. If $p = 1/2$, then $V_R' = (V_{top} + V_R)/2$. Note, that this replacement (upgrade) function can be implemented for any region in the cache, regardless of 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 simply kept unchanged.

**Example 3** The cache contains three regions with formulas $a \land b \land c$, $b \land d$ and $d \land \neg a$. Figure 5.a1 shows the regions with their replacement values (assuming $V_{top} = 6$). Let a new query be $d$. The second and third regions are contained in the query. As both region formulas differ from the query formula in one term only ($b$ for the second region and $\neg a$ for the third one), the generated remainder query is given by $d \land a \land \neg b$. Once the complete answer is produced, the coalescing strategy will substitute the second and third region, as well as the remainder query with one region with formula $d$ (Fig. 5.a2). Its replacement value is $V_{top} = 7$. With the non-coalescing strategy, a new region will be created for $d \land a \land \neg b$ with replacement values of regions $b \land d$ and $d \land \neg a$ upgraded (Fig. 5.a3).

Now let the query be $b \land d \land f$ (see Fig. 5.b1). Two regions, $b \land d$, and $d \land \neg a$, 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 upgraded (from 3 to 4) accordingly to the portion of tuples matching the query in the region contents.

The replacement policy, designed for the semantic containment above, does not change for the semantic intersection. When a new query is launched, any semantic region in the cache has its replacement value upgraded towards $V_{top}$, proportionally to the region involvement in the answer.

**Example 4** Assume the cache contains the regions with formulas $a \land b$, $c \land d \land \neg e$, and the user query be $b \land c$. Figure 6.a shows the regions with their replacement values (assuming $V_{top} = 7$) and the query. The cache detects that the region $a \land b$ has a one-term difference from the query and constrains the remainder query $Rem(Q)$ which is $b \land c \land \neg e$. All tuples from the region contents matching the query report their partial answer. Upon reception of the answer to the remainder query $Rem(Q)$, a new region with the formula $b \land c \land \neg a$ is created. The replacement value is set to $V_{top} = 8$. At the same time, the replacement value of region $a \land b$ is upgraded (from 2 to 4), in proportion to its contribution to the answer set (see Fig. 6.b).
Algorithm 1

\textbf{Input:} cache with semantic regions and query \(Q\).
\textbf{Output:} answer to \(Q\) and the cache updated.

1. Check the query signature against the region signatures in the cache.
2. Case \(S_Q = S_R\): /- equivalence /-
   return the region contents as the answer;
   set \(R\)'s replacement value to the "most recently used" (MRU);
Case \(S_Q \land S_R = S_R\): /- single or multiple query containment /-
   if there is a multiple query containment then choose the region with the minimal cardinality;
   scan the (chosen) region contents and return the tuples matching the query;
   upgrade the replacement value of the region;
Case \(|S_Q \land S_R| = k\): /- semantic intersection /-
   identify all regions, say \(R_1, \ldots, R_m, m \geq 1\), with one-term difference from the query;
   return the tuples matching the query in the semantic regions of \(R_1, \ldots, R_m\), discarding duplications;
   \(\text{Probe}(Q) := Q \land \biglor_{i=1}^{m} R_i\);
   construct the remainder query \(\text{Rem}(Q)\) as follows:
   - set the remainder query to query \(Q\);
   - for each region \(R_i, i = 1, \ldots, m\), calculate the difference term \(a_i\), and constrain the remainder query with \(\neg a_i\);
   - send the remainder query to the server;
   - upgrade the replacement values for all regions contributing to \(\text{Probe}(Q)\);
   when an answer to the remainder query \(\text{Rem}(Q)\) is received, update the cache as follows:
   \textbf{Coalescing:}
   - if regions \(R_{i1}, \ldots, R_{i_j}\) contain the query, replace them with a new region with formula \(Q\);
   - if a region \(R\) is complement to query \(Q\) and formula \(R \lor Q\) is a disjunction, substitute \(R\) and \(Q\) with a new region;
   - otherwise, add a new region to the cache with the formula \(\text{Rem}(Q)\);
   \textbf{Non-coalescing:} add a new region to the cache with the formula \(\text{Rem}(Q)\).

Fig. 7 Query processing algorithm for complete answer sets

4.3 Size-adjusted replacement strategy

The standard LRU strategy handles nicely objects of uniform size. As Web queries can be of different size, a rule of thumb says that purging a region from the cache should not only depend on its age (i.e., the duration the region already resides in the cache) but also on its size. We therefore tend to remove larger regions in order to make room for multiple small regions.

Therefore, we adopt a size-adjusted LRU strategy (SLRU) recently developed for caching Web data of heterogeneous size [2]. This strategy proposes the replacement of regions based on their cost-to-size ratio. The region \(c/s\) is given by its replacement value introduced in the previous section where a new region gets the higher cost. If a region has size \(s\), the ratio \(c/s\) gives the region's cost-to-size ratio. The cache should accommodate a new region, the strategy purges from the cache the regions with the minimal cost-to-size ratio. Moreover, the admission policy does not accommodate a new region in the cache at all, if its cost-to-size ratio is lower than the similar ratios of regions it could substitute.

Although regions in the query cache, unlike objects in a data cache, can change their size and replacement value during residing in the cache, this does not change the SLRU strategy. Like the standard LRU, it processes probe and remainder queries separately. Regions contributing to a probe query are upgraded once their involvement has been detected.

Remainder queries are processed upon reception of answers from the server. Because of the admission policy, not every remainder query is retained in the cache. Generally, the non-coalescing strategy has more chances to succeed with the task of retaining \(\text{Rem}(Q)\) in the cache than the coalescing strategy with the task of merging \(\text{Rem}(Q)\) with one or more regions (because of a large total size).

4.4 The algorithms

This section presents two semantic caching algorithms. Algorithm 1 for the situation where Web repositories answer with all tuples matching the query (i.e., complete answer sets), and Algorithm 2 for the situation where Web repositories only provide a subset of the tuples matching the query (incomplete answer sets).

Complete answer sets. Algorithm 1 given in Fig. 7 summarizes the processing of a user query against the cache regions. The algorithm scans the regions with the query signature. Beyond the optimal cases of equivalence and query containment, it identifies all regions with one-term difference in order to construct the remainder query and to produce the partial answer. To detect regions with one-term difference, the algorithm verifies Theorem 3 on the region signatures. If the condition holds for a region signature, the region formula is checked for a one-term
In complete answer sets. Algorithm 1 implicitly assumes the completeness of the answers, i.e., the answer set provided by the Web repositories contains all relevant tuples with regard to the query. This assumption helped us to manipulate the received answers as sets. However, not all Web repositories provide complete answer sets. If a source detects too many relevant tuples, it normally ranks them and returns the n top-ranked hits only. This has some consequences for the proposed caching mechanisms. To properly deal with complete and incomplete answer sets at the same time, we assume that the source wrappers are capable to detect whether the answer set is complete or not. As discussed next, the cache is informed accordingly.

The incompleteness of the answer set does not change the equivalence and region containment cases [10]. However, it changes the cache mechanism in cases of query containment and intersection (see Fig. 8). Consider the query containment case given in Fig. 8.b. The incomplete region R contains the answer set $R^a$, which is a proper subset of the tuples at the source relevant to the query. To answer the query $Q$ contained in $R$, the cache can re-use the tuples of $R^a$ in the probe query. However, those tuples do not form an complete answer set. In other words, the query containment $Q \subseteq R$ does not lead to $Q^a \subseteq R^a$, and, therefore, the initial query should be submitted to the source to retrieve all relevant tuples. In the one-term difference case, the situation is similar (see Fig. 8.d). The content of region $R$ can be re-used for the partial answer, but it cannot constrain the remainder query.

The incompleteness of the answer set also affects the coalescing strategy. In the query containment case, region $R$ and the new region for $Q$ can contain different tuples in their contents, as $Q \subseteq R$ does not necessarily lead to $Q^a \subseteq R^a$ (see Fig. 8.b) and $Q^a$ can contain tuples not present in $R^a$. Keeping one region $R$ by extending its content with $Q^a$ disables the re-use of $Q^a$ and we have to create a new region $Q$.

Rather, in the region containment case we still coalesce the remainder query $Rem(Q)$ and the existing regions $R_1, \ldots, R_m$. It could happen that the number of tuples in the content of the new region $Q$ will be larger than the number of tuples the source returns. Fortunately, such a behavior only improves the cache performance and relaxes the limit imposed by the source.

One important change concerns the priority of the cases processed. With complete answers, the query containment was more preferable than the region containment because it allows for the local query processing. Instead, in the case of incomplete answers, the region containment is more preferable, as it allows to constrain the remainder query.

Algorithm 2 in Fig. 9 summarizes the differences from Algorithm 1. Under the additional condition that the cache may contain complete as well as incomplete answer sets from the remote Web sources, this algorithm shows all cases of semantic containment/intersection.

5 Experimental Results

Over a period of seven months, we have conducted a series of experiments to check the correctness of our findings. This section reports some of the most significant results.

Workloads. In our experiments, we employed two different workloads. For the first workload, we used the log of real user queries submitted to the Knowledge Broker meta-searcher [4] as it is installed at the Xerox Research Center Europe in Grenoble, France. The user community using this meta-searcher is relatively small (60 to 80 users), and most of the workload queries fit into a few domains of common interest, including computers, printers, business and leisure. Consequently, this workload is characterized by a high weight of frequently used keywords: 1% of the most frequent keywords occur in 20% of the queries and 9.3% of these keywords occur in 50% of the queries.

The second workload is completely different from the first: it contains user queries submitted to the Metacrawler search engine (http://www.go2net.com) and collected by the WebSpy server (http://www.webspy.com). User queries composing this workload arrive from the entire world and expose a wide diversity of user interests. The keyword dictionary of this workload is large, and the keyword distribution is less skewed than the Knowledge
Algorithm 2

Input: cache with semantic regions and query Q.
Output: answer to Q and the cache updated.
1. \(\text{Probe}(Q) := \text{null};\)
\(\text{Rem}(Q) := Q;\)
2. for a region \(R\) in the cache, compare its signature \(S_R\) to the query signature \(Q_R\) and check the following cases:
   - \(S_Q = S_R\): /\ equivalence /
      - return the region contents as the answer;
      \(\text{Probe}(Q) := R;\)
      \(\text{Rem}(Q) := \text{null};\)
      - set \(R\)'s replacement value to \(V_{sp};\)
   - \(S_Q \land S_R = S_Q\): /\ query containment /
      - if \(R\) is complete, scan the region contents and return the tuples matching the query;
      \(\text{Probe}(Q) := R \land Q;\)
      \(\text{Rem}(Q) := \text{null};\)
      - upgrade \(R\)'s replacement value;
      - if \(R\) is incomplete, scan the region content and add the tuples matching the query to the probe query;
      \(\text{Probe}(Q) := \text{Probe}(Q) \lor (Q \land R);\)
      /\- the remainder query does not change /
   - \(S_Q \land S_R = S_R\): /\ region containment /
      - add the tuples from region \(R\)'s content to the probe function;
      \(\text{Probe}(Q) := \text{Probe}(Q) \lor (Q \land R);\)
      - if \(R\) differs in one term, say \(a\), from \(Q\), constrain the remainder query with \(\neg a;\)
      \(\text{Rem}(Q) := \text{Rem}(Q) \land \neg a;\)
   - \(S_Q \land S_R \geq |S_Q| - k\): /\ semantic intersection /
      - if \(R\) differs from \(Q\) in one term, say \(a\), scan the region content and add the tuples matching the query to the probe function:
      \(\text{Probe}(Q) := \text{Probe}(Q) \lor (Q \land R);\)
      - if \(R\) is complete, constrain the remainder query with \(\neg a;\)
      \(\text{Rem}(Q) := \text{Rem}(Q) \land \neg a;\)
3. if \(\text{Rem}(Q) \neq \text{null}\), send \(\text{Rem}(Q)\) to the server;
   - upgrade the replacement values for all regions contributing to the probe query;
4. when an answer to \(\text{Rem}(Q)\) is received, insert a new region \(\text{Rem}(Q)\) to the cache;
   - if the coalescing is chosen and the answer set is complete, coalesce \(\text{Rem}(Q)\) with the regions contained in \(Q\), if any.

Fig. 9 Query processing algorithm for incomplete answer sets

Broker’s workload: as many as 6% of the most frequent keywords occur in 20% of the queries and 27.3% of the most frequent keywords occur in 50% of the queries.

The two workloads (called KB and WebSpy) do not differ much in the query length distribution (see Fig. 10.a), with the average query length being 2.2 keywords. Instead, as shown before, they essentially differ in frequency distributions of query keywords.

For comparison, we have approximated the keyword frequencies with a Zipf-like distributions. In a Zipf distribution with parameter \(\theta\), the frequency of the \(i\)-th keyword is \(c/i^\theta\), where \(0 \geq \theta \geq 1\) and where \(c\) is a normalization coefficient.

Figure 10.b shows the approximation of both workloads with Zipf distributions with parameters \(\theta = 0.75\) and \(\theta = 0.30\), respectively.

With the KB workload, the probability that a new query term is already present in some cache regions is higher than with the WebSpy workload. Consequently, the probability of any semantic containment or intersection and, as we will see later, the cache efficiency is higher with the KB workload than with the WebSpy workload.

Measurements. For both workloads, we tested the query caching mechanism and compared different replacement strategies. More precisely, the three following parameters have been measured in the experiments:

1. Cache efficiency: The average portion of a query answer provided from the cache. For one query, the efficiency is evaluated as \(r_c/r_t\), where \(r_t\) is the total number of answer tuples, and \(r_c\) is the number of the answer tuples retrieved from the cache. For a series of queries, the cache efficiency is the mean of individual query efficiencies.

2. Qualified regions ratio: The average number of regions qualified for a query. It explicitly includes contribution by each operational case, namely equivalence (EQU), region containment (RCO), query containment (QCO), and one-term difference (OTD). It is preferably to gain a higher cache performance with less qualified regions, as scanning too many regions lengthens the cache response time. We compare the coalescing and the non-coalescing policy as well as the caching of complete and incomplete answer sets.
3. False drop frequency: The average number of false drops per query. We experimentally study the optimal tradeoff between parameters $k$ and $F$ that drive the signature generation and influence the false drop frequency.

Both workloads are normalized with respect to the size of the query answers. The cache size varies in the $[10 \cdot s; 200 \cdot s]$ range, where $s$ is the average size of a query answer. Then, KB queries are tested against complete sources (including the Library of Congress) while WebSpy queries are posed against the Metacrawler engine that returns top-ranked and, therefore, incomplete answer sets.

**Replacement and coalescing.** Three replacement strategies have been tested for each workload: standard LRU, LRU with contribution update (LRU+) and size-adjusted LRU (SLRU) with its contribution updated (see Sect. 4.3). For the KB workload, each replacement strategy has been additionally tested with and without the region coalescing.\(^2\) Figure 11 shows the plots for both workloads. All strategies performed better with the KB workload because this workload has a higher probability of finding a hit in the cache than the WebSpy workload. When comparing the two improvements of the LRU strategy (contribution-based update and size-adjustment), the latter appears to be more powerful in improving the cache efficiency. All strategies saturate for larger cache sizes, with the SLRU being the overall winner.

**Qualified regions.** The analysis of qualified regions helps us understand the difference between caching of complete as well as of incomplete answer sets. With the WebSpy workload, answer sets are incomplete and any region coincides with a query. Figure 12 analyses the occurrences for all four operational cases and their contributions to partial answers for the WebSpy workload.\(^3\)

These plots unveil that only three cases contribute sufficiently much to the cache efficiency, with the contribution of region containment being rather marginal.

\(^2\) Because of the incompleteness of answer sets, only the non-coalescing strategy has been allowed for the WebSpy workload.

\(^3\) Note the logarithmic scale for all plots.
Indeed, each equivalence or query containment case provides a full answer, whereas any region containment or OTD case brings a tiny portion of the answer set.

This tiny contribution by one OTD region (varying in the 0-20% range with the average being around 1% per occurrence) is largely compensated by their large number. Totally, OTD regions contribute some 30% of answer tuples, on average. Instead, region containments occur approximately as often as query containments and thus the overall contribution of this case is rather negligible.

The same behavior has been analyzed for the KB workload (see Fig. 13 that shows SLRU plots only). It reveals that the region coalescing plays a negligible role with the KB workload because of the rather short query lengths, and, therefore, the low probability of successful region containment. Instead, there is an essential difference from the SpyWeb workload in the number of qualified regions and the weight distribution of operational cases. Because of retaining remainder queries instead of original queries, the OTD cases are rather rare with respect to the WebSpy workload, but each OTD case contributes a rich set of answer tuples.

As a consequence, the number of qualified regions remains within 2% of the cache regions what is considerably less than the 40% with the incomplete answer sets in the WebSpy workload.\footnote{The numbers refer to the 200\cdot s cache size.}

5.1 False drops

Finally, we have studied the optimal tradeoff between the parameters $F$ and $k$ that heavily influence the false drop frequency. For four values of $F$ (128, 192, 256, and 320), we vary $k$ and count the resulting false drops.

As Fig. 14 shows, for each value of $F$ there is a range of $k$ values that provide a sufficiently low false drop level, with this range getting wider for larger $F$’s. Surprisingly, experimental data somewhat differ from the prediction for the optimal $k$ given by $k_{opt} = F^{1/2}$ (see Sect. 2.4); for $F = 320$, the formula gives $k_{opt} = 100.8$ ($t = 2.2$). We recall however that the above formula serves the semantic containment only [20].
5.2 Signature hashing

The cache dictionary occupies a little portion of the cache as compared to the region contents. A linear scan of the dictionary is sufficiently efficient in caches of moderate size, say, up to hundreds of regions. However, to make the cache scalable, the regions qualified for a query should be found in a sub-linear time.

To reduce the number of visited regions, we hash region signatures. We split the cache directory into \( b \) buckets, \( 1 \leq b \leq F \); where \( F/b \) is the bucket length. Each region signature is split in \( b \) segments, too. The \( i \)-th segment in the signature contains signature bits \( [i \cdot F/b : (i + 1) \cdot F/b - 1] \) and is associated with the \( i \)-th bucket of the directory. A signature fragment is non-empty if it contains at least one bit set to one. A region signature is assigned to the \( i \)-th bucket iff its \( i \)-fragment is non-empty. Although a signature can be assigned to multiple buckets, the number of buckets is kept small and the space overhead is held within 1-2% of the cache size (see below).

The cache size certainly limits the number of regions in the cache. This keeps us from seeking for an incremental balanced structure like B-trees or R-trees. Beyond the simplicity which is of primary interest, the random generation of signatures supplies that no particular fragment of either query or region signature will be more frequently used than other fragments. Therefore, the uniform distribution of bits in the signatures ensures that all buckets will get approximately the same number of signatures.

When we search for qualified regions, we determine non-empty fragments in the query signature and visit associated buckets. As those buckets contain all qualified regions, all other buckets are excluded from the search.

We can reduce considerably the number of visited regions by choosing a large value of \( b \). Figure 15 reports the number of visited regions for the KB workload under the signature hashing, when \( b = F, b = F/2 \) and where the cache size is 200 * s. As the figure shows, longer signatures reduce the number of visited region signatures. Indeed, the number of buckets where a signature is assigned to, depends on the parameter \( k \) (see Sect. 2.4).

As small values of \( k \) keep low false drops (see Fig. 14), the number of signatures in a bucket decreases, as the signature length increases. Therefore, the number of visited signatures can be kept really small. As for the space overhead, the signature hashing duplicates the cache dictionary size for large values of \( b \), but keeps it within 4% of the total cache size.

6. Conclusion and Open Issues

We have presented a mechanism for caching conjunctive Web queries as implemented in the Knowledge Broker meta-searcher [4]. The mechanism is based on signature files and allows for an efficient re-use of queries. We have developed caching algorithms that cope with the semantic containment and intersection between a user query and the semantic regions, as well as region coalescing and replacement strategies that are most appropriate for caching Web queries.

Although the semantic cache has been developed primarily for the Web meta-searching, it can be successfully integrated into mediator systems as well. In mediators, join-queries are decomposed into “elementary” sub-queries that address Web repositories and do not contain joins. Hence, query caching in a mediator system can be organized at several levels, with the cache for Web sub-queries at the lowest level.
For the cache performance, the size of the term dictionary is a crucial parameter. Large dictionaries make the matching of a term in the cache rather unlikely, and thus degrade the cache efficiency. Another result is that the large contribution to probe queries is yielded by OTD cases, i.e., by regions having a one-term difference from the query. For the region replacement, two improvements of the standard LRU strategy (see Sect. 4.2) permit to essentially increase the cache efficiency.

As we have seen in our experiments, the contribution of OTD regions varies in a wide range, with some OTD regions giving up to 20% of probe queries, in contrast to the rest of OTD regions whose contribution is marginal. This contribution rate is actually measured by semantic proximity/overlap of query terms. Semantically close terms have high correlation and therefore share multiple common documents in their region contents. This rises the issue of using semantic ontologies to detect semantically close terms and therefore to faster catch the OTD regions that contribute the most to the probe queries. In the future, we plan to extend our scheme to treat ontologies and measure term overlaps, as proposed by Florescu et al. [17].

The linear scan of the cache dictionary can be improved by signature hashing. As we have shown in Sect. 5.2, the hashing method works well with complete answer sets. However, its performance degrades with regions representing incomplete answer sets, where too many regions are qualified for a query. How to make a query cache more efficient under the incompleteness of the answer set remains an open problem.

Moreover, a theoretical interest represents the derivation of the optimal F-k trade-off for holding low false drop ratio for both semantic containment and intersection cases.

Acknowledgements The authors would like to thank the editors and the anonymous referees for suggestions that greatly improved the structure and presentation of this paper.

References