# Satisfiability and Implication Evaluation of Conjunctive Queries in Semantic Caching

Muhammad Azeem Abbas, Abdul Qadir, Munir Ahmad, Tariq Ali
Center for Distributed and Semantic Computing
Mohammad Ali Jinnah University, Islamabad. Pakistan
azeem.abbas@uaar.edu.pk, aqadir@jinnah.edu.pk, {munirahmad83, tariqali.1982}@gmail.com

***ABSTRACT:*** *Finding satisfiability and implication results among queries is fundamental to several problems in databases especially in distributed databases. The known complexity of finding satisfiability of term S is O ( $|S|^3$). Similarly complexity of finding "Is S implies T ($S \rightarrow T$)" is O($S^3$+K) Where | S | is the number of distinct predicate attributes in S, and K is the number of predicate terms in T (S and T are conjunctive select-project-join(PSJ) queries). We show that with the induction of Cross Attribute Knowledge ($C_{RA}$ ) the above complexity is reduced to O ($|S - C_{RA}|^3$) and O ( $|S - C_{RA}|^3$ + K) for satisfiability and implication respectively.*

**Keywords:** Distributed databases, Querying, Semantic Caching, Conjuctive queries

**Received:** 11 May 2012, Revised 29 June 2012, Accepted 8 July 2012

## 1. Introduction

Satisfiability and implication results are fundamental to several problems in databases especially in distributed databases. It is widely used as a key to find equivalences among queries, query optimization, query rewriting and semantic cache query processing. When optimizing, an equivalent of the original query that is cheaper in execution is found through satisfiability and implication relation. Similarly a query is rewritten that either integrate multiple resources or used in dataware house design[1].

The process of computing available (probe query) and unavailable (remainder query) parts from prestored data fragments (cache) is highly effective in latency of data retrieval from distributed resources where communication cost is a major concern [2]. Since cached and incoming queries are formulas of conjunctive or disjunctive inequalities. So finding Probe (pq) and remainder (rq) queries is a problem of finding implication or satisfiability between cached query (QS) and user query (QU) formulas. The following definitions formally define the problems and notation to be used in the rest of this paper.

**Definition 1:** A select-project-join (PSJ) query Q is a tuple < $\pi_Q$,$\sigma_Q$,operand$_Q$ >, where $\pi_Q$ is Select Clause of query which contains projected attributes. operandQ is the From Clause which contains relation of a database D, from which data is to be retrieved. $\sigma_Q$ is Where Clause which contains conjunctive ($\wedge$) or disjunctive ($\vee$) compare predicates, a compare predicate is of the form P = (X op C) , P = (X op Y) or P = (X op Y + C), where X,Y $\in$ A {Attributes Set}, op $\in$ {<, ≤, >, ≥, ≠ }, C is a constant in a specific domain [1].

**Definition 2:** Implication is defined as *"S implies T, denoted as S $\rightarrow$ T, if and only if every assignment that satisfies S also satisfies T"*.

**Definition 3:** Satisfiability is defined as *"S is satisfiable if and only if there exists at least one assignment for S that satisfies T."*. Let us have a formula (Salary < 20K AND Salary > 8K AND Department = 'CS') is satisfiable, because the assignment 12K / Salary , CS/Department satisfies the formula. Similarly a formula (Salary >10K OR Salary < 12K) is a tautology, because every assignment under this formula is satisfiable. We will use the following semantic cache application in our examples through the rest of the paper.

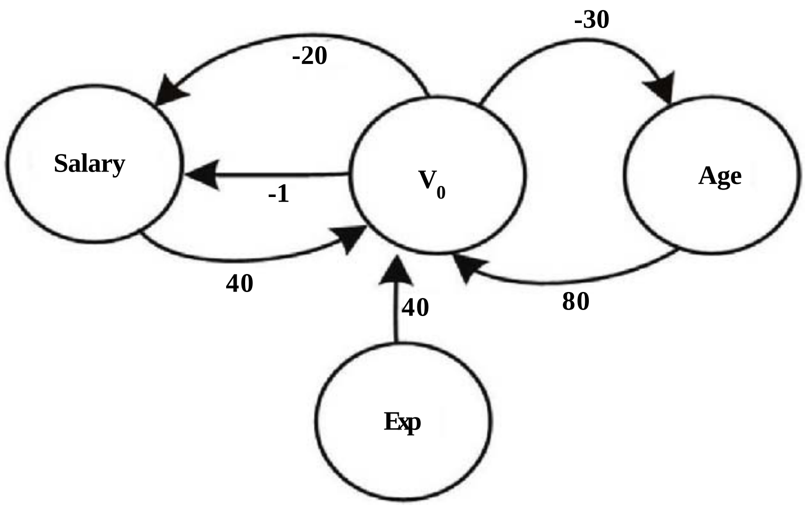
A cached query with associated semantics stored in semantic cache along with resultant data is called a semantic region [3] or semantic segment [2]. If a user query ($Q_U$) posed over n semantic segments ($Q_{S1}$, $Q_{S2}$,..,$Q_{Sn}$) then this user query can either be totally answered (implies) or partially answered (satisfiable) or cannot be answered (unsatisfiable) from underlying segments.

So for above scenario, implication ($Q_U \rightarrow Q_S$) means every tuple retrieved by evaluating $Q_U$ can be obtained by evaluating $Q_S$ i.e. the whole answer to $Q_U$ is available locally in the cache. For satisfiability there are some tuples that can be retrieved by both $Q_U$ and $Q_S$.

In case of implication user query ($Q_U$) shall be posed to the cache i.e. $< \pi_{QU}, \sigma_{QU}, operand_{QS} >$.if satisfiability holds then user query ($Q_U$) is split into two parts i) pq = $< \pi_{QU} \wedge Q_S, \sigma_{QU}, operand_{QS} >$ and ii) rq = $<\pi_{QU}, \sigma_{QU} \wedge \neg_{QS}$, operand$_{QU}$ >. This splitting process is known as query trimming [4]. Query trimming is based on satisfiability and implication results. Let us assume that query trimming take O(QueryTrim) time. Then total (worst case) time taken by semantic cache query processing is $O(|Q_U|^3 + K)$ + $O(|Q_U \wedge Q_S|^3)$ + O (QueryTrim) where K is number of terms in $Q_S$. In section III we show that with the induction of Cross Attribute Knowledge (CRA) [5] this complexity is reduced to $O(|Q_U - C_{RA}|^3 + K) + O(|Q_U \wedge Q_S - C_{RA}|^3)$ + O(QueryTrim).

**Example 1:** Consider an employee database with a relation name Emp (Empid,Department, Age, Salary,Exp). The domain of the Age, Salary, Department and Exp attributes of Emp are {20,...,100},{0.1K,...,1K,15K},{CS, EE, BI, BA},{1,..,50} respectively. Also suppose that the cache already has following cached segment.

$Q_S = < \pi_{\text{Age , Salary, Exp}}, \sigma_{Salary\ \le 1\text{K} \wedge \text{Salary} \ge 40\text{K},} \text{Emp} >$



| | Salary | $V_0$ | Age | Exp |
|---|---|---|---|---|
| **Salary** | 20 | 40 | 10 | ∞ |
| **$V_0$** | -20 | 20 | -30 | ∞ |
| **Age** | 60 | 80 | 50 | ∞ |
| **Exp** | 20 | 40 | 10 | 0 |

Figure 1. (a) [$Q_{U1} \wedge Q_S$] and $G_{QU1} \wedge Q_S$ (b) Shortest Path Table

## 2. Related Work

Satisfiability and implication results in databases [6], [7], [8], [9], [10] are relevant to the computation of probe and remainder query in semantic cache query processing for a class of queries that involve inequalities of integer and real domain. Previous work models the problem into graph structure. Hunt [9] provided an algorithm of complexity $O(|Q|^3)$ for solving satisfiability problem; the expression S to be tested for satisfiability is the conjunction of terms of the form X op C, X op Y, and X op Y + C. Guo et al. [6] provided an efficient algorithm (GSW) for computing satisfiability with complexity $O(|Q|^3)$ involving complete operator set and predicate type X op C, X op Y and X op Y + C. Here we demonstrate GSW algorithm [6] for finding implication and satisfiability between two queries.

The GSW algorithm starts with transforming all inequalities into normalized form through given rules. It was proved by llman [7] that these transformations still holds equality. After these transformation remaining operator set become $\{\leq, \neq, \geq\}$.

- $(X \geq Y + C)$ $(Y — X \leq C)$
- $(X < Y + C)$ $(X \leq Y + C) \wedge (X \neq Y + C)$
- $(X > Y + C)$ $(Y \leq X \geq C) \wedge (X \neq Y + C)$
- $(X = Y + C)$ $(Y \leq X \geq C) \wedge (X \leq Y + C)$
- $(X < C)$ $(X \leq C) \wedge (X \neq C)$
- $(X > C)$ $(X \geq C) \wedge (X \neq C)$
- $(X = C)$ $(X \leq C) \wedge (X \geq C)$

Satisfiability of a conjunctive query Q is computed by constructing a connected weighted-directed graph GQ = $(V_Q, E_Q)$ of Q after above transformation. Where $V_Q$ are the nodes representing predicate attributes of an inequality and $E_Q$ represent an edge between two nodes. An inequality of the form X op Y + C has X and Y nodes and an edge between them with C weight. The inequality X op C is transformed to X op $V_0$ + C by introducing a dummy node $V_0$.

According to GSW [6] algorithm, for any query Q if a negative-weighted cycle (a cycle whose sum of edges weight is negative) found in GQ then Q is unsatisfiable. Otherwise Q is satisfiable. Testing satisfiability among user query $Q_U$ and cached segment $Q_S$ require us to construct a graph $(G_{QU} \wedge Q_S)$ of $(Q_U \wedge Q_S)$ and check $G_{QU} \wedge Q_S$ for any negative weighted cycle. Negative weighted cycle is found through Floyd-Warshall algorithm [11]. Complexity of Floyd-Warshall algorithm is $O(|V|^3)$, so finding satisfiability become $O(|G_{QU} \wedge Q_S|^3)$.

An algorithm with $O(|S|^3 + K)$ complexity for solving the implication problem between two conjunctive inequalities S and T was presented by Ullman [7] and Sun [10]. Conjunctive queries of the form X op Y were studied by [8] and [9]. Implication between conjunctive queries of the form X op Y + C was addressed by GSW algorithm [6] with complexity $O(|Q_U|^2 + |Q_S|)$. GSW Implication [6] requires that $Q_U$ is satisfiable. Satisfiability of $Q_U$ can be checked by above mentioned steps. At first the implication algorithm constructs the closure of $Q_U$ i.e., a universal set that contains all those inequalities that are implied by $Q_U$. Then, $Q_U \wedge Q_S$ if $Q_S$ is a subset of the $Q_U$ closure.

Ahmad et al.[16] proposed graph based indexing scheme for management of cached segments. This indexing scheme is then applied to matching process for finding implication & satisfiability results. Satisfiability and implication evaluation is done through string matching and applied only to select-project queries. Later incoming query is trimmed on the basis of these results. Similarly a graph based query trimming approach was proposed by Abbas et al [18].

**Example 2:** Let us have a user query
$Q_{U1} = < \pi_{Age, Salary, Exp}, \sigma_{Salary \geq 20K \wedge Age \geq 30 \wedge Age \geq 80 \wedge Exp \leq 40}, Emp >$ over cached segment $Q_S$ of Example 1. The directed weighted graph $G_{QU1 \wedge QS}$ of $Q_{U1} \wedge Q_S$ is shown in Figure 1(a). $Q_{U1}$ is satisfiable with respect to $Q_S$, as there is no negative weighted cycle in $G_{QU \wedge QS}$.

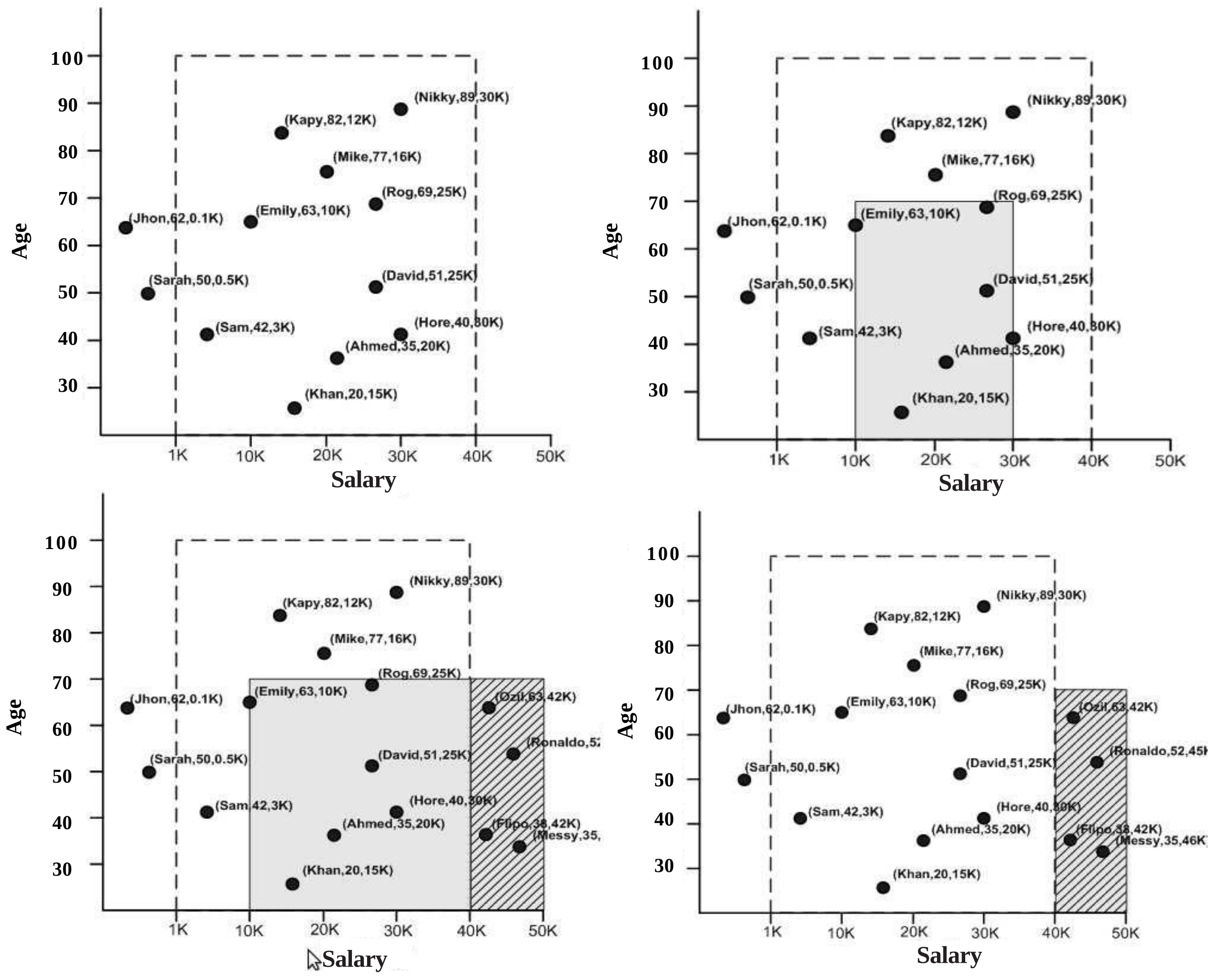


Figure 2. (a top left), (b top right), (c bottom left), (d bottom right) Query Q over Cached Segment S

## 3. Cross Attribute Knowledge

In all previous work satisfiability and implication was only addressed for horizontal partitioned fragments. Where it was assumed that projected attributes of incoming user query and cached segment are same ($\pi_Q = \pi_S$). So implication and satisfiability was only applied to predicate part of the queries. Cross Attribute Knowledge ($C_{RA}$) consider both projected attributes of the queries and their predicates. The following definitions formally defines the $C_{RA}$. Later we provide the effects of $C_{RA}$ in satisfiability and implication problem.

**Definition 4:** Common aligned comparison ($C_{AC}$) is defined as similar clauses of user and cache queries are compared with each other i.e. project clause with project, from clause with from and similarly select clause with select clause.

**Definition 5:** Consider a cached query $Q_S =< \pi_{QS}, \sigma_{QS}, operand_{QS} >$, and a user posed query $Q_U =< \pi_{QU}, \sigma_{QU}, operand_{QU} >$ over the cached query ($Q_S$). Then a predicate attribute of user query (U) is said to be cross attribute knowledge ($C_{RA}$) [5] if

- it is present in the cached query attribute ($\pi_S$) and
- it is not in cached query predicate ($\sigma_S$)

From definition it is clear that $C_{RA}$ itself is not $C_{AC}$, but it actually extends $C_{AC}$ by comparing cache attributes with query predicates. The $C_{RA}$ contributes towards evaluation of Satisfiability and Implication testing as an implicit knowledge.

The $C_{RA}$ is a predicate attribute and it behaves differently when appears in conjunction with other predicate attributes. So here we classify query predicate attributes into three categories: 1) The common predicate attributes ($C_{PA}$), 2) Cross Attribute Knowledge ($C_{RA}$) and 3) Non-Common predicate attributes ($N\text{-}C_{PA}$). This classification is formulated from conjunctive appearance of these categories. Every conjunctive appearance contributes to the answer differently. Their contribution can be shown with the examples given below (consider relation definition given in Example 1 for all examples given below):

**Example 3:** The data set of cached query ($Q_S$) is shown in Figure 2 (a, b, c, d) as white dotted line boxes. A user query ($Q_{U1} = \pi_{\text{Age, Salary, Exp,}}\ \sigma_{\text{Salary} \geq 10\text{K} \wedge \text{Salary} \leq 30\text{K} \wedge \text{Age} \leq 70}$, Emp >) is shown as gray boxes over cached query ($Q_S$) in Figure 2 (b). According to $C_{RA}$ definition, in this case the predicate attribute Age in user query ($\sigma_{QU1}$) is a $C_{RA}$. The CRA is in conjunction with a predicate attribute ($C_{PA}$) Salary, which is common between cached query predicate ($\sigma_S$) and user query predicate ($\sigma_{QU1}$). This

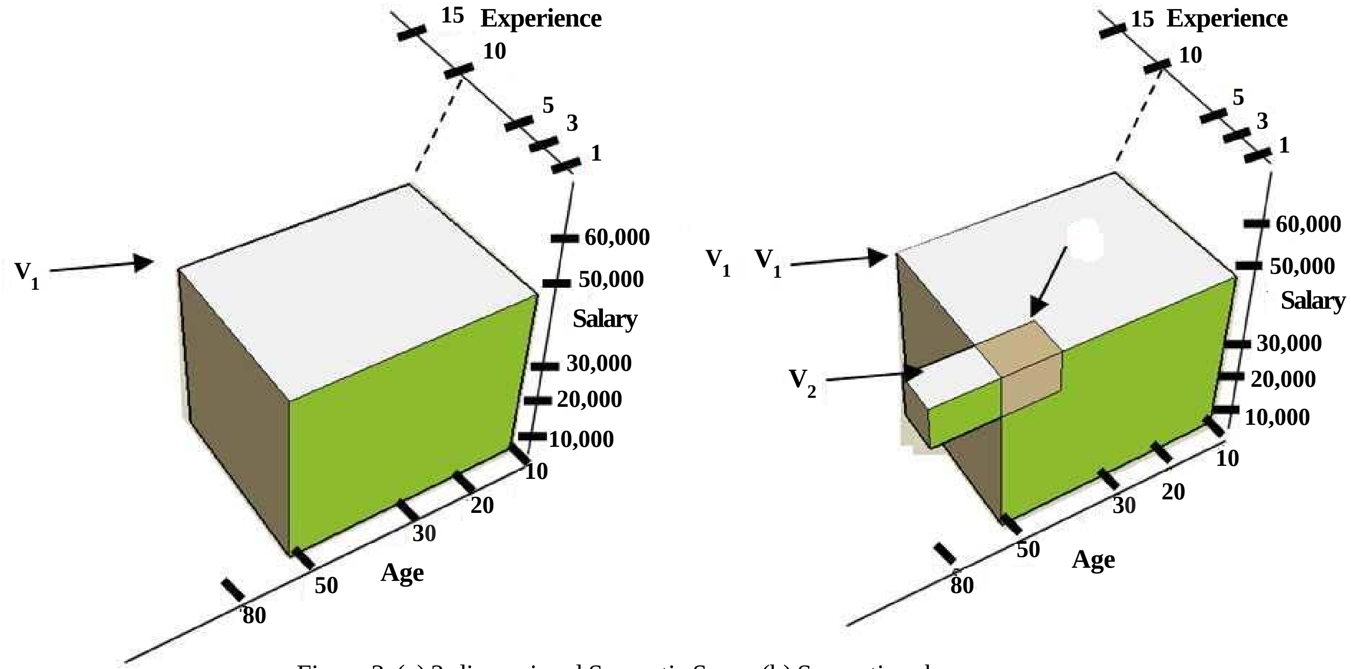


Figure 3. (a) 3-dimensional Semantic Space (b) Semantic sub-spaces

$C_{PA}$ (Salary) implies cached query predicate i.e. ($\sigma_{QU1 \wedge CRA}$). So it can be concluded from the data set as shown in Figure 2 (b) that if $C_{RA}$ appears in conjunction with CPA and this $C_{PA}$ implies cached query predicate attribute, then the whole answer of user query is present in semantic cache. This conclusion can be derived from lemma given below.

**Lemma 1:** Consider a cached query $Q_S = < \pi_S, \sigma_S, operand_S >$, and a user posed query with $C_{RA}$ is $Q_U = < \pi_U, \sigma_U, operand_U >$. We define a new predicate attribute set $\sigma_{U0} = \{\sigma_U - C_{RA}\}$, (this can also be written as $\sigma_U = \{\sigma_{U0} \wedge C_{RA}\}$), Then we have a statement (which needs a proof) as:

- If ($operand_U = operand_S$, and $\pi_U \subseteq \pi_S$ and user query without $C_{RA}$, $< \pi_U, \sigma_U, operand_U >$, is fully answerable from the cache,) then $Q_U = < \pi_U, \sigma_U, operand_U >$ is fully answerable from $Q_S$.

**Proof:** Suppose we have a user query $Q_{U1} = < \pi_{U1}, \sigma_{U1}, operand_{U1} >$ that is fully answerable from $Q_S = < \pi_S, \sigma_S, operand_S >$ i.e. pq = $< \pi_{U1}, \sigma_{U1 \rightarrow S}, operand_{U1} >$ and rq = null. Then let us consider another user query with new predicate set, $Q_{U2} = < \pi_{U1}, \sigma_{U2}$, $operand_{U2} >$ where the new predicate set is $\sigma_{U2} = \sigma_{U1 \wedge CRA}$. So $Q_{U2}$ become

$$Q_{U2} =< \pi_{U2}, \sigma_{U2 \wedge CRA}, operand_{QU2} >$$

According to relational algebra splitting law [12],

$$Q_{U2} =< \pi_{U2}, \sigma_{CRA}(\sigma_{U1}), operand_{QU2} >$$

As per our assumption $\sigma_{U1 \to S}$ then

$$Q_{U2} =< \pi_{U2}, \sigma_{CRA}(\sigma_{U1 \to S}), operand_{QU2} > Q_{U2} =< \pi_{U2}, \sigma_{CRA}(pq), operand_{S} >$$

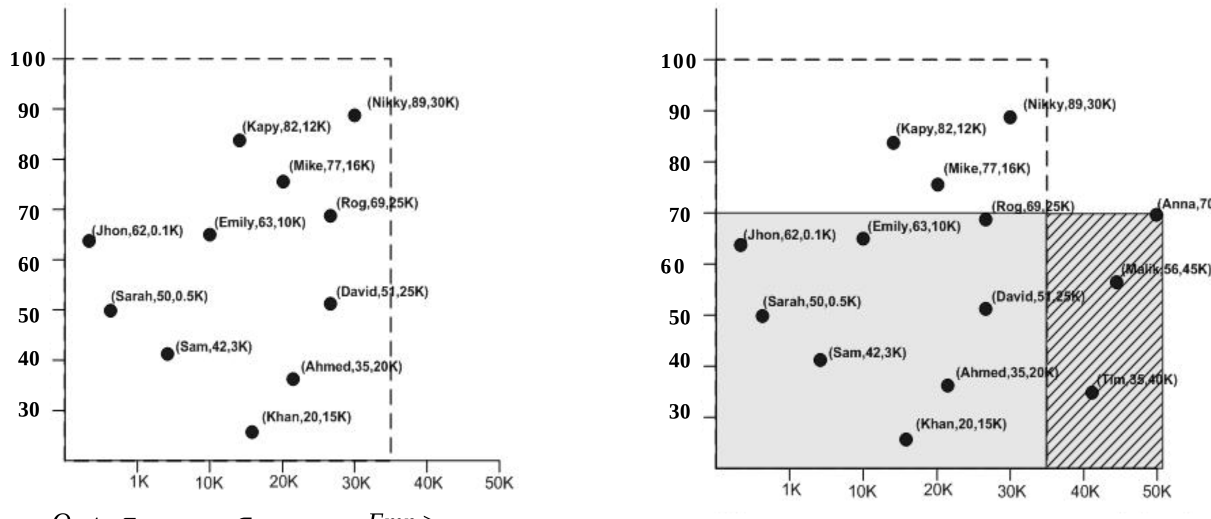
Hence proved that any query that is answerable from semantic cache is also answerable while containing $C_{RA}$ in conjunction.

**Example 4:** Figure 2 (c) shows an incoming user query ($Q_{U2} =< \pi_{\text{Age, Salary, Exp,}}\ \sigma_{\text{Salary} \geq 10K \wedge \text{Age} \leq 70}$, Emp >) over the cached query $Q_S$ shown in Figure 2 (a). The gray color filled box shows the available part (probe query), where lined patterned area represents unavailable data or the remainder query. In this example the $C_{PA}$ (Salary) satisfies the cached query predicate. So any conjunction of CPA and $C_{RA}$, where $C_{PA}$ satisfies cached query predicate will be partially answered from semantic cache. Lemma 1 can be easily extended for common predicate attribute satisfy case.

**Example 5:** A user query ($Q_{U3} =< \pi_{\text{Age, Salary, Exp,}}\ \sigma_{\text{Salary} \geq 10K \wedge \text{Age} \leq 70}$, Emp >) posed over the cached query $Q_S$ is shown in Figure 2 (d). The $C_{PA}$ (Salary) is unsatisfiable in this case. So the cached query $Q_S$ does not contribute any answer to this query ($Q_{U3}$). CRA behave according to $C_{PA}$ when it appears in conjunction with it. That is; ($C_{PA} \wedge C_{RA}$) implies when $C_{PA}$ implies, similarly ($C_{PA} \wedge C_{RA}$) satisfies when $C_{PA}$ satisfies and ($C_{PA} \wedge C_{RA}$) does not contribute if $C_{PA}$ does not satisfies. The other remaining predicate attribute i.e. non-common predicate attribute ($N\text{-}C_{PA}$) need special handling when it appears in conjunction with $C_{PA}$ or $C_{RA}$. All previous semantic cache query processing techniques treats $N\text{-}C_{PA}$ as non-satisfying condition. But our analysis and experimentation shows that $N\text{-}C_{PA}$ can be treated as satisfying in many cases.

### 3.1 Semantic Space

In relational model every relation has semantically correlated attributes. This means that every attribute is related to each other, for example salary and experience of an employee within a tuple are correlated because they belong to a single person. So queries can be visualized as *n*-dimensional spaces. In figure 2 (a, b, c, d) queries are shown in 2-dimensional space, where if a query with more than two predicate or select attributes can be visualized as n-dimensional space where n is the number of predicate or select



$Q_C \leftarrow \pi_{Ename,\ Age}\ \sigma_{Salary \leq 35K}, Emp >$ $\qquad$ $Q_U \leftarrow \pi_{Ename,\ Age}\ \sigma_{Age \leq 70K}, Emp >$

Figure 4. (a) Cached Query ($Q_C$) (b)User Query ($Q_U$) over Cached Query ($Q_C$)

attributes. Below figure 3 (a) shows a query that has three predicate attribute, in 3-dimensional space.

When ever a new query is cached, it forms a semantic sub-space. These sub-spaces combined up to form a semantic space. In above figure 3 (b) a new query creates sub-spaces $V_2$ and $V_3$ when it overlaps previously cached semantic space $V_1$. Any semantic cache query processing should find answer to an incoming query from this semantic space. As discussed earlier in section 3, all previous semantic cache query processing techniques are based on common aligned comparison ($C_{AC}$). In $C_{AC}$, those techniques just compare descriptions of user and cached queries. So they neglect answers that are available in semantic cache. But with $C_{RA}$ and handling $N\text{-}C_{PA}$ properly, maximum data can be retrieved from semantic cache. Our proposed algorithm also consider semantic relevance of data, so a user query always falls in semantic space what ever its entrance dimensions are. With this, user and cached queries with entirely different predicates can result to each other. The example below shows semantically data extraction.

**Example 6:** Let us have a cached query $Q_C$ shown in figure 4 (a,b) as white dotted line box. Results of an incoming user query $Q_U$ over the cached query $Q_C$ is shown in figure 4 (b). Since there is nothing common between cached and user query predicates but still $Q_U$ can be partially answered from $Q_C$. Because data requested in user query is the same as cached query but their predicate dimensions are different from each other. Also the predicate attribute Age of user query is $C_{RA}$. But even without $C_{RA}$, results can be extracted from semantic space.

$Q_C =< \pi_{Age,\, Salary,}\ \sigma_{Salary \leq 10K}, Emp >$

$Q_U =< \pi_{Age,\, Salary,}\ \sigma_{Salary \leq 22}, Emp >$

$Amend_Q =< \pi_{Empid,\, Salary \leq 10K \wedge Exp \leq 22}, Emp >$

$Probe_Q =< \pi_{Empid,\, Salary \leq 10K \wedge Exp \leq 22}, Emp >$

$Rem_Q =< \pi_{Empid,\, Salary \leq 10K \wedge Exp \leq 22}, Emp >$

**Example 7:** A user query $Q_U$ (shown in statement ii above) over cached query $Q_C$ (shown in statement i above) can be answered even if user query ($Q_U$) does not contain $C_{PA}$ or $C_{RA}$. We can not depict these queries in diagrammatical representations here. In this example the predicate attribute Exp of user query $Q_U$ is a $N\text{-}C_{PA}$.

## 4. Satisfiability and Implication Based on Cross Attribute Knowledge

As proved in lemma 1 that if any conjunctive inequalities are satisfiable/implies then conjunction of these inequalities with CRA remain satisfiable/implies. So pruning $C_{RA}$ from original conjunctive inequalities will also give correct results. This pruning of $C_{RA}$ reduces complexity to $O(|Q_U - C_{RA}|^3 + K)$ for finding implication and $O(|Q_U \wedge Q_S - C_{RA}|^3)$ for computing satisfiability between user query $Q_U$ and cached segment $Q_S$.

**Example 8:** The user query $Q_{U1}$ in Example 2 over cached segment $Q_S$ of Example 1 can be evaluated for satisfiability and implication with $\sigma_{QU1 \wedge CRA}$. The directed weighted graph $G_{(QU1 \wedge QS)\text{-} CRA}$ of $Q_{U1} \wedge Q_S$ is shown in Figure 3(a). $Q_{U1}$ is satisfiable with respect to $Q_S$ even with $Q_{U1} - C_{RA}$ as there is no negative weighted cycle.

In previous section, we classified predicates of a user and cached query. There are two types of possible partitioning (horizontal and vertical) among user and cached queries. Our semantic cache query processing algorithm is based on all the combinations of predicates ($C_{PA}$, $C_{RA}$, $N\text{-}C_{PA}$), partitioning types (horizontal, vertical) and $C_{PA}$ implication & Satisfiability results. In this way there are total thirty cases to be addressed.

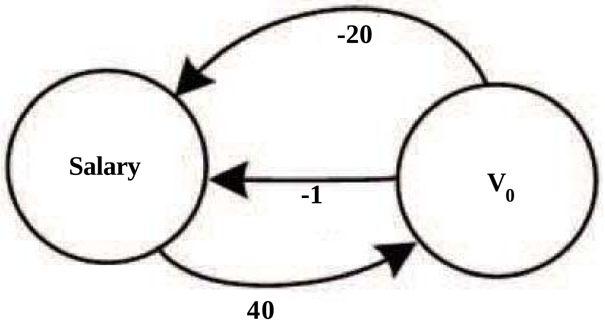
Figure 6 below shows Semantic Cache Query Generator (*SCQG*) algorithm. An incoming user query $Q_U$ and cached query $Q_S$ posed over semantic cache is passed as input to *SCQG* algorithm. These input queries are triplets, as discussed in definition 1. End results of *SCQG* are an amending query, a probe query and a remainder query.

At the beginning of *SCQG* algorithm, the parser function (line 4) separates all information available in semantics of the cached and user query. This information is then stored in global variables for further computation. The parser algorithm is shown in Figure 5.

As in above examples 3, 4 and 5, it is clear that answer to a user query is highly influenced by predicate classification. Our algorithm works accordingly to the appearance of respective predicate type. User posed incoming query and a cached query selected from semantic cache are given as input to *SCQG*. Selection of candidate cached segment is an open question. We select it in linear fashion. Our proposed algorithm generates available part (probe query) and unavailable part (remainder query). At line 4 parser function extract all semantic information present in both user and cached query.

First of all we find either incoming query and cached query are suitable for implication testing or not (line 5). This can be checked by finding a predicate term ($N\text{-}C_{PA}$ (C)) that is in cached query but not in user query. If such term found then these queries are not suitable for implication testing. With this pre-decision we save implication evaluation time.

After that we still check whether these queries are suitable for satisfiability testing or not. If a common predicated attribute ($C_{PA}$) is present then we do satisfiability testing. But still complete user and cached query predicates are not evaluated for satisfiability rather pruned user and cached query predicates are evaluated. If satisfiability holds then incoming query is trimming into probe and remainder queries otherwise in case of unsatisfiability only remainder query is generated. If incoming and cached queries are not suitable for satisfiability testing then incoming query is trimming into probe and remainder query (line 9).



| | Salary | $V_0$ |
|---|---|---|
| Salary | 20 | 40 |
| $V_0$ | -20 | 20 |

Figure 5. (a) [$Q_{U1} \wedge QS$] and $G_{(QU1 \wedge QS) - CRA}$ (b) Shortest Path Table

If user and incoming queries are suitable for implication testing, then we perform implication evaluation but with pruned predicates (line 12). After that if implication does not holds then we perform satisfiability testing.

## 5. Implementation and Results

We implemented a prototype named Semantic Cache Query Generator (*SCQG*) to verify performance and correctness of our proposed algorithm. Experiments are designed to compare the performance of proposed algorithm and Ren et. al. [2] query processing technique. Our primary performance metric is response time while maximum data retrieval is an additional parameter which we observed is highly influential in semantic cache system. Our prototype is implemented in Java and local and remote data is stored in MySql database. Only precise data of cached queries is stored locally. Remote data server that is accessed through internet contains over 3.3 million rows [13]. Local data server contains approximately 30% of the whole data (either horizontally or vertically). Semantics of cached queries are kept in an extensible mark-up language (XML) file; in which each block represent clauses of a conjunctive SQL query. We experimented 1500 queries.The presented results are obtained by averaging the results of three runs of 500 queries over 50 stored queries. All 1500 queries are generated using random attributes and predicates. These random attributes and predicates are generated by strictly following the schema definition and domain values of the underlying database. All generated queries returns non-empty answer which shows their correctness. Ren et. al . [2] computes probe and remainder query by evaluating user and cached query for implication results at first. It then evaluates those queries for satisfiability if implication goes not hold. This technique has higher time complexity as it adds up implication and satisfiability computational time.

Graph above shows time complexity comparison of traditional semantic cache query processing algorithm and *SCQG*. X-axis shows the percentage of similarity between incoming user query and cached segment predicates. Y-axis shows time complexity in milliseconds. Our algorithm compute probe and remainder query within constant time for cached and incoming queries that have similarity below 20%. For exactly equal queries both traditional and *SCQG* takes equal time. But for all queries between 20 and 90 range are computed in polynomial fashion by *SCQG*. But traditional algorithm has high complexity for all queries despite of any similarity concern. Trend-lines clearly show the efficiency of our algorithm.

We grouped our experimented quires accordingly to the branches of our algorithm (*SCQG*). Figure [7] shows computational

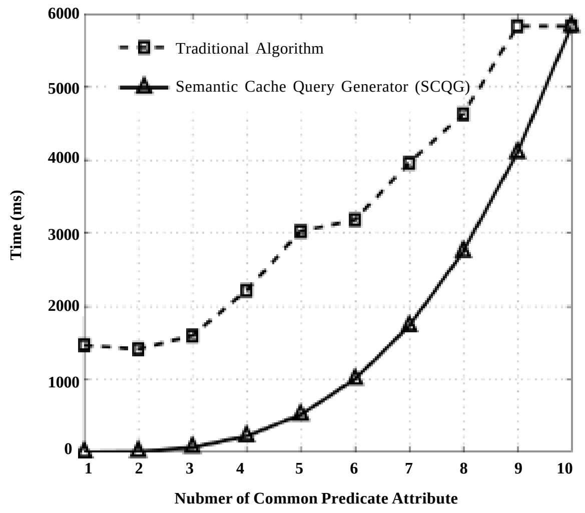
```
SCQG () {
1 Input: $Q_U$ (user query), $Q_C$ (cached query)
2 Output: AmendQ(amending query), ProbeQ(probe query), $RemQ_1$ & $RemQ_2$ (reminder query)
3 Global Variable: $K_A$, $A_1$, $A_2$, $C_{PA}$, $C_{RA}$, $N\text{-}C_{PA(C)}$, $N\text{-}C_{PA(Q)}$, AmendQ, ProbeQ, $RemQ_1$ & $RemQ_2$
4 parser ($Q_U$);
5 if Exists( $N\text{-}C_{PA(C)}$) {
6   if Exists($C_{PA}$ {
7     if Satisfiable($Q_U$, $Q_S$) {
          pq = $< \pi_{Qu} \wedge \sigma_{Qs}$, $operand_{Qs} >$
          rq = $< \pi_{Qu} \wedge \neg \sigma_{Qs}$, $operand_{Qu} >$
          return pq, rq;
8 } else {
      rq = $< \pi_{Qu} \wedge \neg \sigma_{Qs}$, $operand_{Qu}>$
      return rq;
  }
9 } else {
      pq = $< \pi_{Qu} \wedge \sigma_{Qs}$, $operand_{Qs} >$
      rq = $< \pi_{Qu} \wedge \neg \sigma_{Qs}$, $operand_{Qu} >$
      return pq, rq;
  }
10 } else {
11   if Exists($C_{PA}$) {
12     if Implies($Q_U$, $Q_S$) {
          pq = $< \pi_{Qu}, \sigma_{Qu} \wedge \sigma_{Qs}$, $operand_{Qs}>$
          return pq;
13 } else if Satisfiable($Q_U$, $Q_S$) {
      pq = $< \pi_{Qu}, \sigma_{Qu} \wedge \sigma_{Qs}$, $operand_{Qs}>$
      rq = $< \pi_{Qu}, \sigma_{Qu} \wedge \neg \sigma_{Qs}$, $operand_{Qu} >$
      return pq, rq;
14 } else {
    rq = $<\pi_{Qu}, \sigma_{Qu} \wedge \neg \sigma_{Qs}$, $operand_{Qu} >$
    return rq;
  }
15 } else {
      pq = $< \pi_{Qu}, \sigma_{Qu} \wedge \sigma_{Qs}$, $operand_{Qs} >$
      rq = $< \pi_{Qu}, \sigma_{Qu} \wedge \neg \sigma_{Qs}$, $operand_{Qu} >$
      return pq, rq;
 }
}
```

Figure 6. Semantic Cache Query Generator Algorithm

comparison of queries that does not fall in implication relationship. Our algorithm take pre-decision by evaluating predicate classification. The branch at line 10 of SCQG handle all such queries. Dotted line in graph above represent traditional algorithm. *SCQG* is shown in solid line. Efficiency of our algorithm is better in maximum situations.

The branch at line 9 of algorithm SCQG handles queries that does not have a common predicate attribute yet other predicate attributes exists. In this case it is known that both incoming and cached query satisfies each other, so our algorithm perform query trimming in constant time. Figure [7] shows time complexity of traditional algorithm (dotted line) and SCQG (solid line).

Figure [8] shows comparison of traditional (dotted line) and *SCQG* (solid line) algorithms for quires that fall under branch at line 5 of *SCQG* algorithm. Where incoming query have larger answer set than cached query and they both posses some common



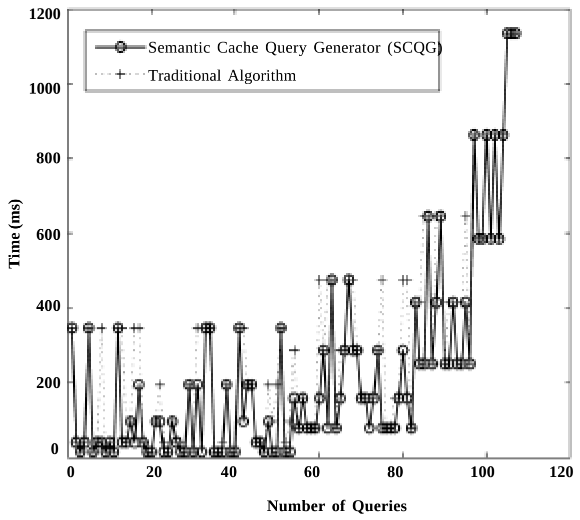
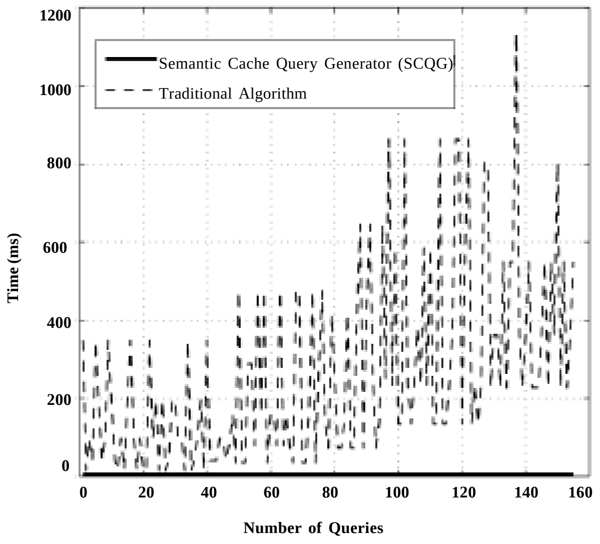


Figure 7. Experimental Results



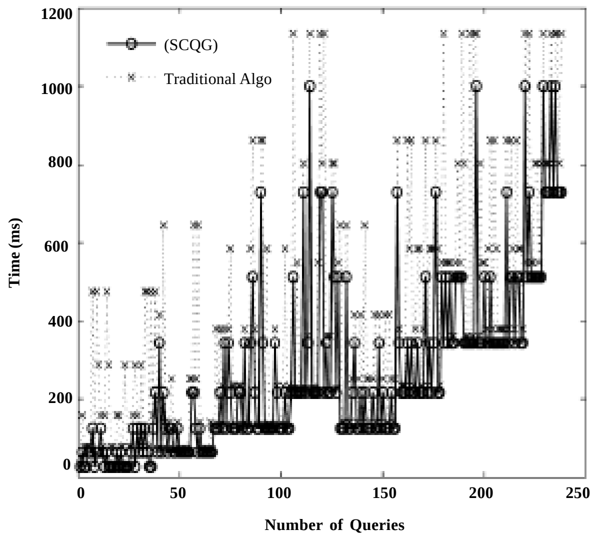


Figure 8. Experimental Results

predicate attributes. Time complexity of *SCQG* remains low in all cases because traditional algorithm sums up implication and satisfiability complexity for satisfiable cases. Also for implication queries we prune incoming and cached query which reduces time complexity require to compute implication relationship.

## 6. Conclusion

Efficient and effective solutions for satisfiability and implication problem are needed as it is important and widely-encountered in database problems such as semantic cache. In this paper query predicates are classified based on their semantic meanings. An algorithm for semantic cache query processing based on satisfiability and implication relationship among cached and user query is proposed. The proposed algorithm computes the available and unavailable part from cache against the incoming user query. Our algorithm process query evaluation while keeping in view the semantic classification of the predicates of the underlying queries. This makes our algorithm time efficient over its predecessors.

## References

[1] Halevy, A.Y. (2001). Answering queries using views, *VLDB J.*, 10, p. 270-294.

[2] Ren, Q., Dunham, M. H., Kumar, V. (2003). Semantic Caching and Query Processing. I*EEE Transactions on Knowledge and Data Engineering*, IEEE Computer Society, p. 192-210.

[3] Dar, S, Franklin, M. J., Jonson, B. T., Srivastava, D., Tan, M. (1996). Semantic Data Caching and Replacement, *In*: Proceedings of 22nd VLDB Conference, Mumbai.

[4] Keller A. M., Basu, J. (1996). A Predicate-Based Caching Scheme for Client-Server Database Architectures, *VLDB J.*, 5 (2), p. 35-47.

[5] Abbas, M. A., Qadir, M. A., Cross Attribute Knowledge: A Missing Concept in Semantic Cache Query Processing, 13th IEEE International Multitopic Conference (INMIC 09), Islamabad Pakistan.

[6] GUO, S., SUN, W.,WEISS, M. (1996). On satisfiability, equivalence, and implication problems involving conjunctive queries in database systems, *IEEE Trans. Knowl. Data Eng*. 8 (4) 604-616.

[7] Ullman, J. D. (1989). Principles of Database and Knowledge-Base Systems, 11. Computer Science Press.

[8] Klug, A. (1988). On Conjunctive Queries Containing Inequalities, *ACM*, 35 (1), p. 146-160.

[9] Rosenkrantz, D. J., Hunt, H. B. (1980). Processing Conjunctive Predicates and Queries, *In*: Proc. Conf. Very Large Databases, p. 64- 71.

[10] Sun, X., Kamell, N. N., Ni, L. M. (1989). Processing implication on queries. *IEEE Trans. Softw. Eng*. 5, 10 (Oct.), 168-175.

[11] Floyd, R.W. (1962). Algorithm 97 Shortest Path, Comm. ACM, 5(6), p. 345, June.

[12] Hector Garca-Molina, Jeffrey, D. Ullman and JenniferWidom., Database Systems: the Complete Book, GOAL Series.

[13] Sample database with test suite, (2011). https://launchpad.net/test-db, Version 1.0.6 dated February.

[14] Tariq Ali, Muhammad Abdul Qadir, Munir Ahmad. (2010). Translation of relational queries into Description Logic for semantic cache query processing, Information and Emerging Technologies (ICIET), International Conference on , 1 (6) 14-16

6, June, ICIET.2010.5625709.

[15] Tariq Ali, Muhammad Abdul Qadir, DL based Subsumption Analysis for Relational Semantic Cache Query Processing and Management,10th International Conference on Knowledge Management and Knowledge Technologies 1â Ø A¸S3 September Messe Congress Graz, Austria.

[16] Munir Ahmed, Sohail Asghar, Muhammad Abdul Qadir, Tariq Ali.(2010). Graph Based Query Trimming Algorithm for Relational Data Semantic Cache, The International Conference on Management of Emergent Digital EcoSystem, MEDES.

[17] Ahmad, M., Qadir, M. A., Ali, T., Abbas, M.A., Afzal, M. T. book chapter: Semantic Cache System, in the book Semantics, eds. (M. T. Afzal).

[18] Abbas, M. A., Qadir, M. A., Ahmad, M., Ali, T., Sajid, N. A. (2011). Graph based query trimming of conjunctive queries in semantic caching, Emerging Technologies (ICET), 7th International Conference on, p. 1-5.