1 Problem addressed and its significance

The convergence of the rapidly progressing fields of wireless communication and lightweight hardware coupled with technologies like Geographic Information Systems (GIS) and Global Positioning System (GPS) has led to the emergence of a new field called Mobile GIS. Mobile GIS is bringing fundamental changes to the way geography is utilized and data is handled in mobile environments. It is extending the enterprise GIS by providing its users the ability to bring work with them, off the office environment It is bridging the gap between working in the office and working on the move. Users of Mobile GIS can now retrieve, manipulate and update enterprise data right from their PDA's anytime, from anywhere in the world. The resulting benefit is consistently improved productivity. Examples include real time access and tracking of shipments, emergency services, car navigation services, real time stock quotes, field services etc. With the proliferation of Mobile GIS applications, the emphasis is now laid on spatial data; data that has a characteristic location attribute in addition to any other attributes it may have. Issues of storage, retrieval and efficient management of spatial data in these applications are gaining prominence.

Caching is a technique where frequently accessed data is generally stored closer to the requester than the original source of the information. This technique is employed to enhance the system performance by improving the query response time. Queries can be answered faster if the requested information is in the cache rather than sending the request to the original source, which may be remotely located. Other advantages of caching include alleviating the load at the server and reducing network usage. Caching is especially important in todays mobile environments where limited bandwidth and errant wireless connections make disconnected mode of operation both pertinent and indispensable.

Conventional page-server[1] architecture benefits most from clustering of data and its relative performance is extremely sensitive to the degree of database clustering [DFMV90]. In a mobile environment the data requested by mobile clients may differ much depending on individual settings and preferences. A physical organization that favors spatial locality exhibited by one client will lead to poor locality for other mobile clients. Hence it is safe to assume that data within a page at a database server will barely exhibit any degree of spatial locality. Also, page and tuple caching techniques use page and tuple references to identify the corresponding data present in the cache. These references do not capture the semantics of the data stored in the cache. We are unable to take advantage of schema knowledge with the help of these references alone and hence it is impossible to support content based reasoning just on the basis of these techniques. We need an altogether different approach to manage and efficiently utilize the cache while dealing with the spatial data in mobile environments. Spatial data has a unique characteristic location attribute. Since Mobile GIS applications deal with spatial data, the spatial queries in these applications tend to exhibit semantic locality due to their inherent location attribute. This fact can be exploited for our caching purpose and this is where semantic cache comes into picture. The idea of a semantic cache is very simple. Along with the data, the description of the query that generated that data is also stored in the cache [DFJ+96], [KB96]. These descriptors also known as semantic descriptors are then used in determining and manipulating the contents of the cache. Semantic caching can be particularly effective in mobile GIS environments where applications frequently need semantically related data and in many cases the results of semantically associated queries are likely to overlap each other.

2 Contribution

The recent progress in the field of mobile GIS and the promise it holds to completely revolutionize the way we work and interact directly with the world around us makes it imperative to develop an infrastructure capable of efficiently managing spatial data. Such a system has to work within the confines of the mobile environment and within the bounds set by the inherent limitations of the mobile devices like memory, computing power, battery life, etc. The success of such a system is a lot dependent on the efficacy of the cache management scheme it adopts. Cache management encompasses policies for admission into the cache and eviction from the cache, broadly known as policies for cache replacement. A good cache replacement strategy will not only improve the query response time but also drastically reduce the communication costs between the client and the server. This thesis is an endeavor in this direction. Our contribution to this end is twofold. First, we present a new mechanism for cache replacement. Our replacement algorithm is based on semantic locality of reference and it takes advantage of schema knowledge to efficiently manage

[1] An architecture in which a disk-page is the unit of transport between the server and the client.

the cache. It takes into account three parameters (cache hits, communication costs and optimal usage of cache) and tries to optimize along these dimensions. We formulate the problem as an integer linear program and then solve a relaxation to obtain a fractional optimal solution. We then propose a system architecture that implements semantic caching in mobile environment. Our system uses the existing relational database framework to store and manipulate the semantic descriptions of the queries and in turn use this information to efficiently manage data in the cache. We build a prototype of the system as a "proof-of-concept" that captures the essential elements of semantic caching and how it is performed. We implement a variation of the algorithm proposed in [RH80] that determines if and how a query can be answered from local cache. We also suggest other alternative and innovative ways of efficiently answering queries from local cache with the help of previously answered queries.

3 Semantic Caching

3.1 Concept

To understand the concept of semantic caching, let us start by briefly going through the mechanism employed in a conventional page caching scheme. Let us assume a client-server model, where the client application continuously sends data requests to the server which is remotely located. These requests are in the form SQL queries on relational database. To enhance the query response time, the client application uses a local cache where the results from the previous queries are stored with the anticipation that the cached data will be accessed again in the near future. The user issues a query which the application is supposed to provide an answer for. Since the local cache is initially empty, the application encounters a cache miss and it forwards this request to the server. The query is executed at the server and the results are send back to the client. More often than not the client receives data as a set of pages from the server. These pages contain some data in addition to the data requested by the user query. The idea behind sending data in pages is based on the assumption that spatial locality of reference holds true which implies that it is quite likely that some other data present in the same page will also be accessed in the near future. Other benefits of transferring entire pages to the client include amortizing the communication costs and the overheads incurred in communicating with the remote server on every single query request. The pages fetched from the server are stored in the local cache and page references are created to help locate these pages in the cache. These references form the metadata for the data stored in the cache. Whenever a new query enters the system, a check is first made to see whether its page reference matches with any of the references already present in the system. If the answer is affirmative, then this query can be answered from the cache. Else, like any other request, this request is also forwarded to the server and the data is fetched into the local memory. Important point to note is that the page references simply act as pointers to the pages in the cache. They do not contain any information regarding the data contained in those pages. A new query is answered from the cache only if it was one of the queries previously stored in the cache or its data happened to be located in the same page as that of a previously requested query. Since the semantics of the data are not known, it is impossible to support content based reasoning on the basis of these page references. The presence or absence of data in a cache is simply determined by checking whether its page reference is present in the cache table entry.
Now in case of semantic caching, instead of the page references, we have query descriptors that not only function as a reference to the data, but also contains some meaningful information about the data they refer to. This added functionality of the query descriptors is used to determine whether new query can be answered from the existing data in the cache. Semantic Caching can be thought of as a special case of general page caching scheme where the descriptor of the contents of the cache is not just a simple page reference but in fact an intelligent reference which conveys some important information regarding the data it refers to. Along with the query results the query descriptors are also stored in the cache. Now, when a new query enters the system an attempt is first made to see if the answer to the query can be obtained from the results already present in the cache. This check is made by evaluating the new query against the query descriptors already present in the cache. If the new query can be mapped either partially or totally to any of the of query descriptors previously stored in the cache, then it is certain that at least a partial result to the query is present in the cache and there is a likelihood of the query being satisfied form the local cache at least partially.

3.2 Benefits

Caching query descriptors along with the query results have some important benefits:
1. adjusts grouping of queries to the requirements of the incoming query so that no irrelevant data is cached along with the relevant ones, thus reducing overhead in managing the cache.
2. minimizes the cost of cache lookup due to the compact representation of the cache content based on semantic query descriptors.
3. adapts dynamically to the patterns of the user queries rather than caching static clusters of tuples [DRWS].
4. It is also possible to partially answer the query from the cache and fetch the remaining part of the result from the server.
5. Approximate answers to the queries may be provided quickly, where applications have some tolerance to the accuracy of the results provided.

3.3 Issues

The research issues in caching have traditionally been selecting the cache granularity (page level, tuple, hybrid, object and attribute caching), cache coherency and cache replacement strategy. But while dealing with semantic caches some other issues come to the forefront. The first and the foremost issue is to determine whether a new query can be satisfied from the existing data present in the local cache. This has been a widely researched topic in academia, and has similarities with problems of query folding [Qia96], query satisfiability problem and answering queries from views. Next task is to process the query and extract the results from the cache. And the third issue is to manage the cache by selecting and implementing suitable policies and mechanisms for cache replacement.
We survey the previous work done in these areas and compare and contrast the various techniques found therein.

the issue of semantic query processing has been a widely researched topic in the academic world. although the concept of semantic caches is pretty straightforward, that is , storing the results of a query along with the query itself, the use of semantic caches to answer queries is a difficult problem and a rather daunting task. in general there can be 4 cases to be dealt with

- deciding when answers are in cache,
- extracting answers from cache,
- semantic overlap and semantic independence, and
- semantic remainder.

there are cases when the result of a query is present in the cache in its entirety, but we do not have sufficient information to extract the results from the cache. hence the distinction between the first and the second.

---

explain these in brief

Other issues not handled in this work
Data security and replication . Computing model Push / pull * semantic caching and containment (special case) : contribution of this paper data validation and transfer : contribution of this paper

4 Literature Survey

in this section we start with the concept of query folding . then we go on to address the satisfiability and the containment problem involved with the query processing . we then briefly explain the query trimming technique and finally end this section by detailing the remainder query operation.

5 Query Satisfiability Problem

Terminology used:
Query: A query corresponds to an expression in a query language. An example of a query in SQL (cite it) query language could be "SELECT name, age FROM table1 WHERE age > 40".
Answerset: Answerset A_Q is the interpretation of a query Q corresponding to a database instance. Put plainly, answerset is the result obtained after executing a query against a database.
Temporary: A temporary T is a set of queries $\{T_1, T_2, \ldots, T_n\}$ stored in the local cache. The answerset corresponding to temporary T is given by $A_T = A_{T_1} \cup A_{T_2} \ldots, \cup A_{T_n}$
In our discussion we use T interchangeably to be either the set of queries or the corresponding answerset that is stored in local cache and trust that the context will disambiguate the meaning.

The first and the foremost task in semantic caching is to determine whether the answer to a new query is present in the cache. In other words we need to find whether the implication $Q \longrightarrow T$ is true or equivalently to test whether the boolean expression $\neg Q \vee T$ is satisfiable. The satisfiability and implication problems have been studies widely in the literature [GSW96, SK89, GG99, Lev00, LY85, LY87, YL87, Qia96, LMSS95, RH80].

Rosenkrantz and Hunt in [RH80] show that the general satisfiability and implication problems in the integer domain are NP-hard. In that paper they discuss the satisfiability and equivalence of conjunctive mixed predicates containing inequalities of the form $x\theta c$, $x\theta y$ and $x\theta y + c$ where x,y are variables, c is a constant and $\theta \in \{=, <, \leq, >, \geq, \neq\}$. they give a $\emptyset(n^3)$[2] algorithm for the restrictive satisfiability and implication problems where $\neq$ in not allowed and the domain is of integers. Guo at. el. in [GSW96] present a $O(n)$ algorithm for the same restrictive problem. In their thorough treatment of the topic, they study the satisfiability and implication problems under the integer domain and the real domain as well as under two different operator sets which differ in the inclusion of $\neq$ operator.They also show that the algorithm that solves a satisfiability problem can be used to solve the equivalent implication problem and vice versa.

A related problem called query folding or query rewriting is also researched extensively in the literature [Qia96].
Qian in [Qia96] describe Query folding as "the activity of determining if and how a query can be answered using a given set of resources, which might be materialized views, cached results of previous queries, or queries answerable by other databases". They develop an exponential time algorithm that finds all complete or partial folding when both the query as well as the resources are conjunctive queries where the selection conditions are restricted to equality. They also give a polynomial-time algorithm that tests the existence of complete or partial folding acyclic conjunctive queries.
We can regard query containment as a special case of query folding. If a query can be completely folded using a the set of queries present in the temporary T, it can be implied that the answer to the query in contained is the answerset of T.

Larson and Yang in [LY85] give the necessary and sufficient conditions for deciding whether a query can be computed from a single derived relation. They lay the restriction that both the query and the derived relation has to be defined by PSJ-expressions, that is, relational algebra expressions involving only projections, selections and joins, in any combination.
In [YL87] they address the problem of computing a query from a set of derived relations. They compute only complete folding, which are foldings that depend solely on the resources. They solve the query transformation problem for conjunctive queries when both the query and the derived relations are defined by PSJ-expressions. Their approach considers what amounts to one-to-one mappings from the derived relations

[2] n stands for the number of inequalities present in the predicates.

to the query, and does not search the entire space of rewriting, therefore may not always find all the possible rewritings of the query.

Levy et.al. in [LMSS95] consider the problem of computing answers to queries by materialized views. They consider the problem of finding a rewriting of a query that uses materialized views, the problem of finding a minimal rewritings and finding complete rewritings. they express the original query in terms of materialized views . minimal –¿ minimum number of views required to express the query. Complete implies the entire query can be wholly expressed in terms of views only.

they show that : 1. all possible rewritings can be obtained by considering containment mappings from the views to the query. 2. when both the query and the views are conjunctive and don't involve built in comparison predicates, the problem is NP-complete. 3. the problem has 2 independent sources of complexity. a. number of possible containment mappings . b. the complexity of deciding which literals from the original query can be deleted.

they describe a polynomial time algorithms for finding literal of the query that can be removed (guaranteeing removal of redundant literals only) and in some case, all redundant literals.

Redundancy removal algorithms + containment mapping enumeration algorithms –¿ algorithm for finding rewriting of a query. analysis of complexity with 1. disjunction 2. built-in predicates

my analysis —– this paper is not exactly the solution to our problem .
we need to find if q1 contains q2.
this paper answers the question " q2 can be expressed in terms of q1"

[GG99] present a general logical framework for semantic caches where they consider the use of all relational operations across the caches for answering queries, the various ways to answer, and to partially answer, a query by cache. They address when answers are in cache, when answers in cache can be recovered, and the notions of semantic overlaps, semantic independence, and semantic query remainder.

[Lev00] surveys the state of art on the problem of answering queries using views, and synthesizes the disparate work into a coherent framework. They also describe various applications of the problem and the algorithms proposed to solve them with the relevant theoretical results.

[Fin82] describe how processing can be improved using intermediate results and answers produced from earlier queries. They come up with an intuitive model for queries called query graph, that supports common expression detection for optimization of a stream of requests. Based on the query graph model they present a methodology for inter-query optimization. Their algorithm is correct but not complete, in the sense it will answer affirmatively only when the implication is valid, however not all valid implication will be identified by the algorithm. They implement a Pascal program COMMON which implements a variation of the algorithm that they describe. The program can determine temporaries that can be useful in answering new queries .

my analysis : ———

They however, do not assume any semantic knowledge of the contents or sequencing of the queries . theirs is a generalized scheme for query processing

we extend their query model and algorithm and incorporate the semantic knowledge for processing of the queries. [read the other paper —] [i think, using the information in dunham of rectangles... in an example of using semantic knowledge for query processing]

in multiple query answering, the results of a query can be used to at least partially answer another query if the second query can be folded using the first one. in client server environment where views and queries might be cached at the client side, client queries can be answered more efficiently if they can be folded using the cached data [KB96]

checking the satisfiability , equivalence or implication relationship between the query and the set of derived relations is the first step for query folding . Comparing a derived relation predicate P and a query predicate Q, there are three scenarios :

- P and Q is unsatisfiable, implying the derived relation will not contribute to the answer of the query Q;
- $Q \longrightarrow P$ i.e. Q implies P, implying a the whole answer of Q is contained in the derived relation

- Neither P contradicts Q nor Q implies P, implying that there may exist tuples in the derived relation that contribute to the answer of the query Q.

the satisfiability and implication problems have been studies intensively in the literature cite . when the $\neq$ comparison between variables is allowed , the problem becomes more difficult , or even NP-hard in the integer domain.

Query Trimming if a query Q can be partially answered by a semantic region S, it must be divided into two parts : one that is satisfied by S and the other part that cannot be satisfied. the unanswered part of the query is sent to the server for processing. this entire operation is known as query trimming. Basically query trimming is the process of dividing Q into two sub queries: Probe Query : which retrieves the portion of Q satisfied by S. Remainder Query : which is sent to the server of r execution, and it retrieves the portion of Q not found in S

———————————— Coalesce and Decomposition

once the remainder query is sent to the server and the answer is obtained, we have three disjoint parts at our disposal.

- the intersection of Q and S
- the difference of S with respect to Q
- the result Q‘ obtained from the execution of the remainder query at the server

we have a number of options to choose from.

- Complete coalesce : it reduces the number of semantic segments in cache, thus cache maintenance is decreased . in addition it aids in queries processing . but the price to be paid it that it may lead to the bigger and bigger segments leading to poor cache utilization
- no coalesce : keeps all the three segments separate. but it results in large number of semantic segments.
- partial coalesce : to merge the new segment with either of the segments. this seems to me the most viable solution of the three.

6 Satisfiability Algorithm

An atom A is a predicate of the form $(x\ \theta\ y)$ or $(x\ \theta\ c)$ where
x, y are variables (attributes),
c is a constant, and
θ is a comparison operator.
Clause $C = A_1 \vee A_2 \vee \ldots A_n$, a disjunction of atoms.
Formula $f = C_1 \wedge C_2 \wedge \ldots C_m$, a conjunction of clauses.

An SQL query predicate can be represented as a formula f consisting of conjunction of clauses. In order to show that the results of query Q are present in the answer set of locally cached queries, we need to show that $Q \longrightarrow T$. where $T = \cup\, T_i$ and $T_i \in \{Q_1, Q_2, \ldots, Q_n\}$, a set of previously stored queries in the cache.

From elementary logic we know that $(Q \longrightarrow T) \wedge \neg(Q \longrightarrow T) =$ FALSE
Also, we have $\neg(Q \longrightarrow T)$
$\equiv \neg(\neg Q \vee T)$
$\equiv Q \wedge \neg T$
$\equiv Q \wedge \neg(T_1 \vee T_2 \vee \ldots T_n)$
$\equiv Q \wedge (\neg T_1) \wedge (\neg T_2) \wedge \ldots (\neg T_n)$

Hence by testing the above conjunctive boolean expression, we can show whether the query Q can be satisfied by queries in T or not. If the above expression evaluates to TRUE, it means that there is a predicate in Q that does not imply T and hence the implication $(Q \longrightarrow T)$ is FALSE.

The Algorithm for testing conjunctive boolean expressions in presented below. It shows whether an expression is satisfiable or not. This algorithm is adopted from [LY87] which is a more restricted version of the algorithm presented by Rosenkrantz and Hunt in [RH80].

Algorithm SATISFIABILITY
Input: A boolean expression B consisting of conjunctive clauses, of the form $B = C_1 \wedge C_2 \wedge \ldots C_m$ where, each clause C is given by $C = A_1 \vee A_2 \vee \ldots A_n$, a disjunction of atoms and,
an atom A is a predicate of the form $x\ \theta\ c$ (simple condition) or $x\ \theta\ y$ (connective condition) where,
x, y are variables (attributes),
c is a constant, and
θ is a comparison operator, $(\theta \in \{=, <, <=, >, >=\})$.
Output: Boolean value TRUE if the expression B evaluates to true, FALSE otherwise.

Each variable x_i is associated with a domain having a lower and an upper bound denoted by $r_{x_i} = (L_{x_i}, R_{x_i})$. The domain of integers is assumed for all variables.

- The first step of the algorithm is to find a permissible range for all variables involved in all the simple conditions B_k . Starting with a predetermined values for lower and upper bound $(-\infty, \infty)$, adjust the bounds according to the following rule.

  if $B_k = (x_i > c)$ then $r_{x_i} = (max(a_{x_i}, c+1), b_{x_i})$
  if $B_k = (x_i >= c)$ then $r_{x_i} = (max(a_{x_i}, c), b_{x_i})$
  if $B_k = (x_i < c)$ then $r_{x_i} = (a_{x_i}, min(b_{x_i}, c-1))$
  if $B_k = (x_i <= c)$ then $r_{x_i} = (a_{x_i}, min(b_{x_i}, c))$
  if $B_k = (x_i = c)$ then $r_{x_i} = (c, c)$
  At the end of this step we have a defined range for all variables involved in simple conditions. If for any range $L_x > R_x$ (lower bound greater than the higher bound), it means that there is no single value in the domain that will satisfy the range and the range is empty. In that case the boolean expression B is unsatisfiable and the algorithm terminates.
  If none of the clauses are unsatisfiable and if only atoms of form $(x\ \theta\ c)$ are present, then the boolean expression is satisfiable. The respective value of the variables for which the expression becomes true is

the lower bound for each variable x_i given by the term L_{x_i}.

If atoms of the form $(x\ \theta\ y)$ are also present, then we proceed to the next phase called the graph creation phase. In this phase atoms of the form $(x\ \theta\ y)$ are handled where $(\theta \in \{=, <, >\})$.

A directed graph is constructed with the nodes representing the variables and the edges represent the connective condition between the variables.

- Let a node be denoted as $N(v;(a,b))$, where v is a set of variables associated with the node, and (a,b) is their permissible range. For each variable x_i in B, create a node $N(x_i;(a_{x_i},b_{x_i}))$ where $(a_{x_i},b_{x_i}))$ is the initial permissible range obtained from the previous step.

- Merge nodes whose edges are of the form $(x_i = x_j), i \neq j$. Let the two nodes be denoted as $N_{x_i}(u;(a,b))$ and $N_{x_j}(v;(c,d))$. Modify node N_{x_i} to be $N_{x_i}(u \cup v;(max(a,c),min(b,d)))$ and delete node N_{x_j}.

- For each connective condition of the form $(x_i > x_j)$ add an edge from the node N_{x_j} to node N_{x_i} .

  For a variable to satisfy all conditions, its value must be consistent with the values of all its predecessors in the graph, and all paths to each predecessors .
  (example diagram)
  Since no integer value can satisfy the inequality of the form $x_i < x_j < \ldots < x_n < x_i$, a cycle in the graph will always equate to false, and the boolean expression is unsatisfiable.
  Assuming the graph contain no cycle, do the following.

- Start with the node that does not have any predecessor in the graph. Mark it and assign its variable the lowest bound . Then repeatedly select any node having only marked immediate predecessors, adjust the lower bound of its range and mark it. Whenever all nodes have been processed and all permissible ranges are non-empty, then the boolean expression B is TRUE. There exists at lease one combination of values that satisfies all the conditions, the one obtained by setting each variable equal to the minimum value in its permissible range.

This algorithm, adopted from [LY87] is a specialized algorithm given by Rosenkrantz and Hunt in [RH80] where they prove its correctness. The worst case running time of the algorithm is $O(n^2)$ where n is the number of variables present in B. The graph G built from B, in worst case may contain n^2 edges. By choosing appropriate representation of the graph, operation on each edge can be performed in constant time.

7 Tackling the Problem of Answering Queries from Cache

Let Q be the query and let A_Q be its answer set.
$A_Q = \{t|Q(t)\}$
Let T be the set of queries in cache such that $T = \cup T_i$, $T_i \in \{Q_1, Q_2, \ldots, Q_n\}$.
$A_T = \{t|T(t)\}$

Our final goal is to find A_Q and if possible try to find it from A_T
We need to check whether $A_Q \subseteq A_T$
i.e. to checke whether $\forall t, t \in A_Q \longrightarrow t \in A_T$
i.e. to checke whether $\forall t, Q(t) \longrightarrow T(t)$
i.e. to test whether $Q \longrightarrow T$

Techniques:

- Evaluate the boolean expression $B = Q \wedge (\neg T_1) \wedge (\neg T_2) \wedge \ldots \wedge (\neg T_n)$. If this expression evaluates to TRUE, it means that there exists a predicate in Q that does not imply T and hence $Q \longrightarrow T$ is FALSE. We now send the expression B to the remote database for further processing. The answer set A_Q can then be obtained from the union of B and $(Q \wedge T)$.
  i.e. $A_Q = B \cup (Q \wedge T)$
  Issues: For arbitrary large $|T|$, the expression B becomes too complex to evaluate and the computations involved in testing its satisfiability increase exponentially.

- Send the query Q and the list of cached queries T to the remote database. The database will create and execute the expression $Q' = Q \wedge \neg T$ and return the results.
  The database only needs to create a complex query string for Q'. Then the database can compare the cost of processing Q' versus Q and execute the query and send back the results accordingly.

- If query $Q' = (Q \wedge \neg T)$ is difficult to evaluate locally, let Q_s be a query such that $Q_s \supseteq Q'$. Now if Q_s is simpler to evaluate, we can send this query to the remote database and fetch the results. A_Q can then be obtained from the resultset of $Q_s \cup T$.
  Issues: The answerset of Q_s can be much larger that Q'. So it may be worthwhile analyzing the cost and the resultset obtained from both Q_s and Q before executing one of them.
  Example: Query predicates of the form $(x + y)\ \theta\ c$ can be reduced to the form $x\ \theta\ c'$ where c' can be calculated as MAX$(y) + c$. Now it is much faster to solve simple predicates of the type $x\ \theta\ c'$ than the original ones.
  $\theta \in \{<, <=, >, >=\}$

8 Procedure

ProcMain

- Accept SQL query from the user and check for valid syntax
- call ProcQuerySatisfaction to check if the answer to the query is present in the local cache
- if query answer present in cache, do the following
  - rewrite the query so that the modified query can obtain its answer from the local cache
  - compute remainder query to be executed remotely
  - execute both modified query and remainder query against respective databases and send the results to the client

  else
  - execute the original query against remote database(s), fetch the results and send them to the user
- call ProcCacheManagement for further processing of the queries and their results

ProcCacheManagement

- Run query admissibility test to determine if the user query is worth caching
- if worthwhile,
  - Run replacement algorithm to make room for the new query results.
  - Store the query results in the local cache
  - update auxiliary data regarding the query and the cache.

  else discard the query and its results

ProcQuerySatisfaction

- create the query signature sign(Q) using the relation names and attributes present in the query's predicates.
- for all queries T_i present in the cache set, do the following:
  - compare sign(T) with sign(Q)
  - if sign(T) is not a subset of sign(Q) discard T from further consideration
- create a set I of all queries in the cache set that satisfy the subset condition. These queries are potential candidates whose answers may contain the answer to the user query Q
- for each query T_i present in set I do the following
  - check if all predicates of T_i are implied by predicates of Q
  - if true, the answer to the query Q is present in the answer to the query T_i, that is stored in cache. return true
- if no T is found such that all the predicates of T_i are implied by those of Q, return false

9 Architecture

Our architecture is based on the object relational data model in a client server environment. since the use of object relational database technology is widespread, we believe that it is advisable to use such a technique to manage cache. Also the availability of light weight small footprint databases in the markets make it easier for users to develop and deploy applications using such databases.

in our architecture, we store the data in the cache and along with the data, the queries that resulted in that data are also stored in the cache. storing queries in database offers considerable benefits in terms of ease of access and manipulation as compared to storing them in flat files. And since we are essentially going to manipulate these queries in order for caching purpose, it makes more sense to store them in such a manner that its retrieval and manipulation becomes extremely efficient and convenient. a relational table called the query table is used to store the queries, and its associated meta-data like count, query string query identifier. another table called cache table is the table where the cached data is stored. in essence this table is the real cache of the system. these two tables share a many to many relationship and hence to express this, another table called association is stored that acts as a link between these two primary table.

10 Query Graph model

A query node consists of nodes corresponding to instances of relations present in the query. the node also consists of local predicates of the form x theta c and x theta y whose attributes belong to that relation only.

Edges : Connecting two relations, and corresponds to the join predicates between them.

A query graph is a 4 tuple ¡¿, G : an undirected graph N : nodes present in the graph E edges present in the graph. γ : global predicates

this definition is a not exhaustive, but a simplified version since it does not include aggregate functions like MIN, Max, and the quantifiers All , None, some .

11 Cache Management

The importance of efficient cache management in the overall success of the system cannot be overemphasized. This is especially true in the case of Mobile GIS applications where queries tend to be semantically related and the efficacy of the system is directly dependent on how efficiently a cache management strategy finds and utilizes the semantic locality present in the data. Due to the limited size of the cache there is an upper bound to the amount of data that can be stored in it. Also, the data present in the cache at some point in time may become useless either due to the changing requirements of the user or due to its staleness. Hence it needs to be flushed out or updated with the latest information from the source. Cache management scheme encompasses the important policies of admission into the cache and eviction from the cache collectively known as cache replacement policy. Replacement strategy has to be devised that not only determines what data is to be replaced in the cache by new data, but also makes these computations quickly and efficiently. It should strive to minimize the overhead in examining and maintaining the contents of the cache. The aim of the cache replacement strategy should be to optimize the performance of the cache and at the same time make the entire mechanism completely transparent to its users. It should make no difference to the users whether the query is satisfied from the actual source of data or from the local cache. A good cache replacement policy will try to maximize the probability of cache hits by keeping data in the cache that has a higher probability of access in the near future and flushing out data that is unlikely to be accessed in the near future. Cache management activity also include book keeping activities such as updating cache table entries, updating their time-stamps, recalculating matrices, checking the freshness of the data etc.

11.1 Cache Replacement Policy

When a new query comes in, a decision has to be made whether to admit that query in the cache[3]. If there is enough room in cache, the query result is directly stored in the cache. If the cache is already full, some of the existing data will have to be purged out of the cache. Making the right selection of target regions to be removed is pertinent to efficient management of cache. The cache replacement strategy employed can be based on a cost model that incorporates information about the time taken to locate the source of data, execute the query at the remote source, fetch the data into the cache and also the frequency of access of the data. Let each rectangle be assigned a metric, a positive number indicating the relative importance of that rectangle in the cache. This metric could be based on the cost model described above or could be based on other factors like Manhattan distance LRU, etc. The larger the value of the metric, the greater is the likelihood of that rectangle being accessed in the near future.

[CSL98] describe a mobile caching mechanism in a mobile client server environment. They develop a spectrum of cache replacement policies for attribute caching, object caching and hybrid caching. These policies adopt the access probabilities of database items as an indicator for the necessity of replacing a cached item. These probabilities are denoted by a replacement score. In the first case they use the mean inter arrival operation duration as the replacement score. They better the approach by using a window, the cache item with the highest arrival duration within the window is replaced. Their third scheme assigns weights to each arrival duration such that the recent duration have higher weights.

The cache replacement strategy adopted by [RD98] is based on the LRU (Least Recently Used) approach. The granularity of cache replacement is a semantic region (a query result) . Since the size of the regions in the cache can change with time due to coalescence and decomposition, the freshness of each region also changes. Based on its freshness, each region is assigned a time-stamp. The replacement strategy works by removing the regions with older time-stamps.

[RD99] propose a cluster based approach to manage a semantic cache. If a query Q_1 can be partially or totally answered by query Q_2, then both the queries are placed in the same cluster. They then use a modified LRU (two-level LRU) cache replacement strategy to determine the candidate targets for replacement. First the LRU is used to select the oldest cluster based on their time stamps. Then the oldest segment in that cluster is selected as the potential candidate for replacement. The rational behind this scheme is that if a part of the cluster had been recently visited, the other parts are also likely to be accessed soon. Similarly if a cluster hasn't been visited for a long time, it may not be referenced again in a while. This scheme exploits temporal locality (using LRU scheme) as well as Spatial locality (using the clustering approach) .

* what do they lack *

[3]Selective admission of partial results is also possible.

12 Problem Framework and Problem Formulation

Although considerable research has been done in devising new replacement strategies for semantic caches in mobile environments, most of it is directed towards maximizing the cache hits and/or reducing the communication costs between the client and the server. Little effort is expended at optimizing the usage of the cache. We believe that optimal utilization of cache in addition to the above factors will lead to the overall success of the semantic cache management system. We device a heuristic that takes all the three dimensions (cache hits, communication costs and optimal usage of cache) into account for efficient and effective cache replacement. In our cache replacement policy the granularity is a semantic region, i.e. a query result generated due to the execution of a user query. Since any query executed on a relational database yields a set of tuples having a set of attributes, a query result can be visualized as a rectangular region. The size of the rectangle indicates the size of the query result, with the length of the rectangle indicating the number of tuples and width indicating the number of attributes present in the query result. For our discussion we shall use the terms region and rectangle interchangeably both indicating the query result generated by executing a query. In our treatment of the topic we assume that a suitable cost function is already determined based on one or more of the above factors and that every rectangle in the cache is assigned a value based on the cost function. Let v_i be the value of rectangle i, indicating its relative importance amongst other rectangles in the cache and let w_i be its weight, indicating the size of the query result. Our aim is to find a set of rectangles with the largest possible sum of their values such that their total size does not exceed the maximum capacity W of the cache. Since we also store the query descriptors along with the query results, there is a limit to the number of descriptors and hence the number of rectangular regions that can be accommodated in the cache. Let this limit be denoted by a positive integer k. Our problem can now be stated as:

Given a set of rectangles with a weight and a value function defined on it, maximize the sum of values of the rectangles subject to the constraints that the sum of their weights do not exceed the fixed size W of the cache, and the number of such rectangles do not exceed a predetermined value k.

This problem can be modeled as the classical 0-1 Knapsack Problem with an additional cardinality constraint as shown by [CKPP97].
The classical Knapsack Problem (KP) is defined by a set $N := \{1, \ldots, n\}$ of items, each having a positive integer profit p_j and a positive integer weight w_j , and by a positive integer knapsack capacity c. The problem requires the selection of a set of items which will maximize the overall profit but entails a condition that the overall weight of those selected items do not exceed the knapsack capacity. This problem can be expressed as the following Integer Linear Programming (ILP) formulation:

$$\text{maximize} \quad \sum_{j \in N} p_j x_j, \tag{1}$$

$$\text{subject to} \quad \sum_{j \in N} w_i x_i \leq c, \tag{2}$$

$$x_i \in \{0, 1\},\ j \in N, \tag{3}$$

where the binary value $x_j, j \in N$, is equal to 1 if and only if the item j is selected. With the additional cardinality constraint on the number of items that can be selected, we have one more constraint that can be formulated as:

$$\sum_{j \in N} x_j \leq k,\ 1 \leq k \leq N, \tag{4}$$

where k is the maximum number of items that can be packed in the knapsack. [CKPP97] give an exact solution to this problem by formulating it as a Dynamic Programming Problem (DPP).
Since the algorithmic complexity of this problem is pseudo polynomial in W, for arbitrarily large values of W the solution becomes unwieldy. To alleviate this problem we can either scale the weights in such a way so a to impose an upper bound on W or we can find an approximate solution to this problem. Our approach is to find an approximate solution to this problem.

13 Finding Approximate Solution

The existing $2D$ problem can be converted into a simpler $1D$ problem by the following transformation. Let all rectangles be transformed into line segments by projecting them on X-axis. Each rectangle (now represented by a line in $1D$) will still have the same weight and value function associated with it. We now have a set of intervals of at most one dimension. Any overlapping intervals are converted into non-overlapping intervals by splitting the interval into parts: the overlapped part and the non-overlapped part. The weight of these two new intervals change according their individual sizes. However their values are assigned as following. The value of the non-overlapping interval remains the same as that of the original interval, while the value of the overlapping interval is the sum of the values of the individual intervals that caused the overlapping. This idea can be illustrated with a following example. Let I_1 and I_2 be two intervals that overlap each other with respective weights w_1 and w_2 and values v_1 and v_2. Let the common overlapping interval be represented as I_{12} with its weight w_{12} and value v_{12}. The new intervals created are as follows:
$I_1^{'} = I_1 - I_{12}$, $w_1^{'} = w_1 - w_{12}$, and $v_1^{'} = v_1$;
Similarly, $I_2^{'} = I_2 - I_{12}$, $w_2^{'} = w_2 - w_{12}$, and $v_2^{'} = v_2$;
and the newly created interval $I_{12} = I_1 \cap I_2$, with weight w_{12} and $v_{12} = v_1 + v_2$.
We now have a set of three non-overlapping intervals. After all the intervals are transformed into non-overlapping intervals by the above procedure, we have a sequence of intervals. Intuitively, the overlapping intervals denote common attributes between queries. If an attribute is common among queries, there is a higher probability of it being accessed in near future and hence is a probable candidate to be cached. Higher probability of access causes its value function to increase and is equal to the sum of values of the overlapping intervals.

If we consider each interval consisting of discrete elements[4], our new problem can be stated as follows:
Input : A sequence S of N elements, each having a non-negative value v.
Output : Set A of at-most k intervals, such that the sum of values of all elements in all the intervals present in A is maximized.
Constraints : Total numbers of elements present in all intervals in A does not exceed the maximum capacity W and the total number of intervals present in A do not exceed k.
Formally,
Sequence $S = (s_1, s_2, s_3, \ldots, s_n)$.
Interval $I_i = [b_i..e_i]$, where $1 \leq b_i \leq e_i \leq N$.
Also $1 \leq e_{i-1} < b_i \leq e_i < b_{i+1} \leq N$. (condition for non-overlapping intervals)
Set $A = \{I_i\}$, where $|A| \leq k$ and $\sum_{i=1}^{|A|} (e_i - b_i + 1) \leq W$.
Value of interval I_i is $val(I_i) = \sum_{x=b_i}^{e_i} s_x$.

$$\text{maximize} \sum \text{val}(I_i), \tag{5}$$

where $I_i \in A$.

Recursion formula :
$B(i, j, W, k)$: A range from i to j containing at most k intervals and the total number of elements in all the intervals do not exceed W.

$B(i, j, W, k) = max[B(i, l, w', 1) + B(l + 1, j, W - w', k - 1)]$,
where $i \leq l < l + 1 \leq N$, and $1 \leq w' \leq W$.

Also, $B(j, k, W, 1) = max\{I_i\}$ or $B(j, k, W, 1) = max\{[b_i..e_i]\}$,
where $j \leq b_i \leq e_i \leq k$ and $e_i - b_i + 1 = W$.

For ease of simplification this problem can be further transformed into an equivalent problem in the graph domain by the following mapping. Let each original interval be represented by a node. The weight and the value of the interval now represents the weight and the value of the node respectively. The edge between

[4]Each element can be a tuple of fixed size. i.e. constant weight and its value is the value of the interval divided by the number of elements present in that interval.

two nodes denotes the overlapping of the corresponding intervals. Two nodes are said to be independent of each other, if they do not share an edge. Our aim is to find a set of mutually independent nodes with the objective of maximizing their total value, with the constraint that their total weight does not exceed W. In effect we seek the maximum weighted independent set for this graph.
The new problem can be described as follows.

Given an undirected graph with a weight and a value function defined on nodes such that both the functions map nodes to real-valued positive numbers; weight $w : n \to \Re$ and value $v : n \to \Re$, find a set S of at most k mutually independent nodes such that the sum of their values is maximized, subject to the constraint that the sum of their weights do not exceed a predefined weight constant W.

Input Graph $G = (V, E)$.
Let e_{ij} denote an edge from node i to node j.

$$e_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Let x denote the indicator variable for node i such that

$$x_i = \begin{cases} 1 & \text{if node } i \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The LPP formulation of this problem is as follows:

$$\text{maximize} \quad \sum_{i=1}^{n} v_i x_i, \tag{8}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq W, \tag{9}$$

$$x_i + x_j \leq 2 - e_{ij}, \tag{10}$$

$$x_i \in \{0, 1\} \tag{11}$$

Analysis:
For n nodes, the number of equations formed are $\binom{n}{2} + 1$.
The problem is an linear integer programming problem with $O(n^2)$ equations.
If the constraint for indicator variable x is relaxed so that it can take all real values from 0 to 1, we have a mixed integer program.

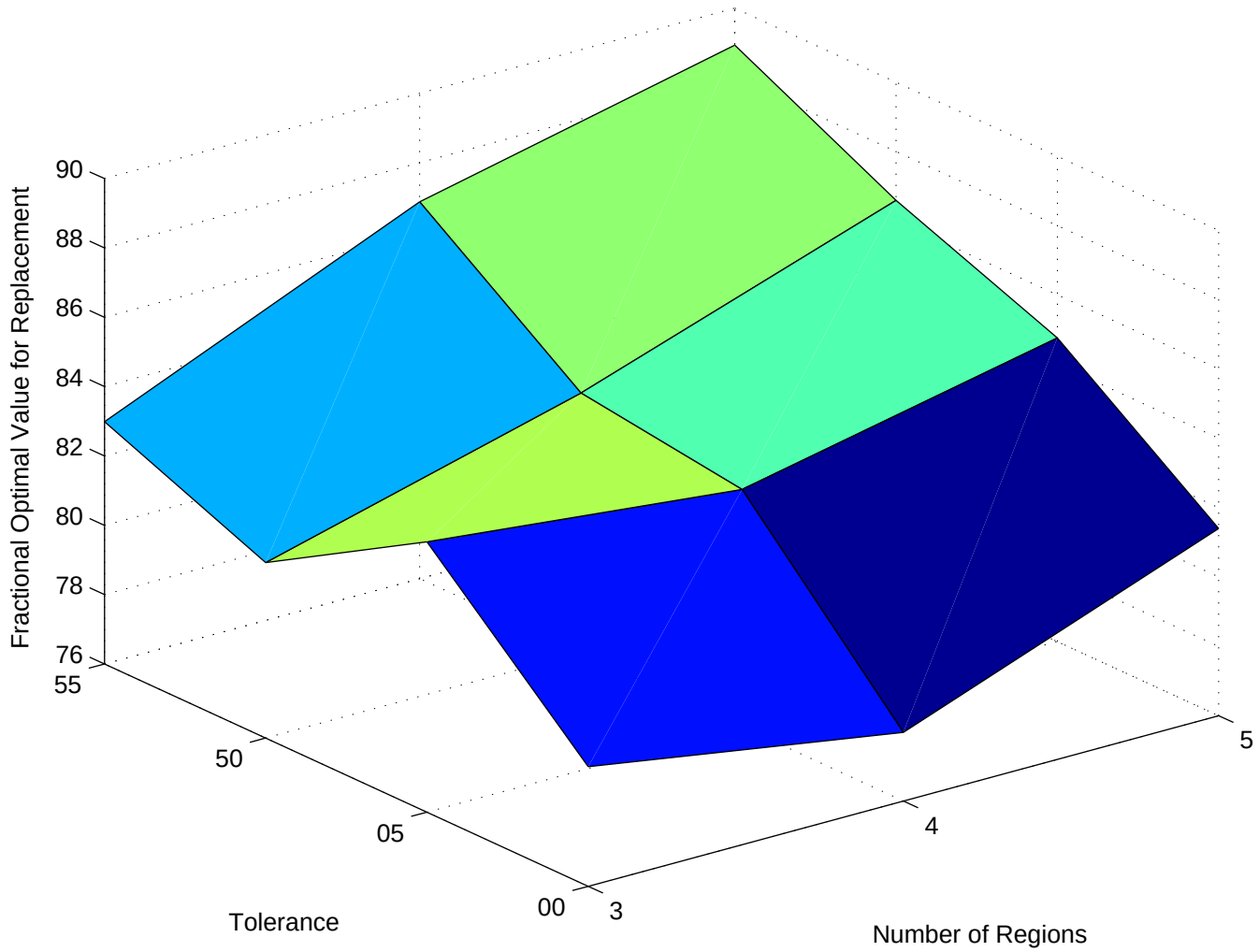

Figure 1: Fractional optimal value function

14 Experiments

A sample experiment is performed to find a solution to the mixed-integer problem. Figure **??** gives the fractional optimal usage of the cache, while Figure 1 give the fractional optimal cost value achieved using the mixed integer program.

References

[CKPP97] A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. Technical report, 1997.

[CSL98] Boris Y. L. Chan, Antonio Si, and Hong Va Leong. Cache management for mobile databases: Design and evaluation. In *Proceedings of the Fourteenth International Conference on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*, pages 54–63. IEEE Computer Society, 1998.

[DFJ+96] Shaul Dar, Michael J. Franklin, Björn ór Jónsson, Divesh Srivastava, and Michael Tan. Semantic data caching and replacement. pages 330–341, 1996.

[DFMV90] David J. DeWitt, Philippe Futtersack, David Maier, and Fernando Velez. A study of three alternative workstation-server architectures for object oriented database systems. In *The VLDB Journal*, pages 107–121, 1990.

[DRWS] Punit R. Doshi, Elke A. Rundensteiner, Matthew O. Ward, and Daniel Stroe. Prefetching for visual data exploration.

[Fin82] Sheldon Finkelstein. Common expression analysis in database applications. In *ACM-SIGMOD*, pages 235–245, 1982.

[GG99] Parke Godfrey and Jarek Gryz. Answering queries by semantic caches. In *Database and Expert Systems Applications*, pages 485–498, 1999.

[GSW96] Sha Guo, Wei Sun, and Mark A. Weiss. Solving satisfiability and implication problems in database systems. *ACM Trans. Database Syst.*, 21(2):270–293, 1996.

[KB96] Arthur M. Keller and Julie Basu. A predicate-based caching scheme for client-server database architectures. *VLDB Journal: Very Large Data Bases*, 5(1):35–47, 1996.

[Lev00] A. Levy. Answering queries using views: A survey. Technical report, 2000.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 95–104, San Jose, Calif., 1995.

[LY85] Per-Åke Larson and H. Z. Yang. Computing queries from derived relations. In Alain Pirotte and Yannis Vassiliou, editors, *VLDB'85, Proceedings of 11th International Conference on Very Large Data Bases, August 21-23, 1985, Stockholm, Sweden*, pages 259–269. Morgan Kaufmann, 1985.

[LY87] Per-Åke Larson and H. Z. Yang. Computing queries from derived relations: Theoretical foundation. Technical report, 1987.

[Qia96] Xiaolei Qian. Query folding. In Stanley Y. Su, editor, *12th Int. Conference on Data Engineering*, pages 48–55, New Orleans, Louisiana, 1996.

[RD98] Q. Ren and M. Dunham. Semantic caching and query processing. Technical report, 1998.

[RD99] Qun Ren and Margaret H. Dunham. Using clustering for effective management of a semantic cache in mobile computing. In *MobiDE*, pages 94–101, 1999.

[RH80] D. Rosenkrantz and H. B. I. Hunt. Processing conjunctive predicates and queries. In *VLDB'85, Proceedings of 6th International Conference on Very Large Data Bases, 1980*, pages 64–72, 1980.

[SK89] Xian-He Sun and Nabil N. Kamel. Processing implication on queries. *IEEE Trans. Softw. Eng.*, 15(10):1168–1175, 1989.

[YL87] H. Z. Yang and Per-Åke Larson. Query transformation for psj-queries. In Peter M. Stocker, William Kent, and Peter Hammersley, editors, *VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, England*, pages 245–254. Morgan Kaufmann, 1987.