1 Introduction

1.1 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 have a characteristic location attribute in addition to other attributes they 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 are generally stored closer to the requester than to 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. Caching is especially important in todays mobile environments where limited bandwidth and errant wireless connections make disconnected mode of operation both pertinent and indispensable. Other important advantages of caching at the client side include alleviating the load at the server and reducing network usage by reducing the amount of data transferred between the client and the server.

Traditional cache replacement schemes like page caching or tuple caching assume and derive benefits from spatial locality of reference[1]. DeWitt et. al. in [DFMV90] show that the conventional page-server[2] architecture benefits most from clustering of data and its relative performance is extremely sensitive to the degree of database clustering. In a typical mobile GIS 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 may lead to poor locality for other mobile clients. Hence the assumption that spatial locality of reference benefits caching is questionable while dealing with spatial data in mobile GIS applications. Also, Mobile GIS applications frequently deal with semantically related queries whose results, in many cases, are likely to overlap. Conventional page and tuple caching schemes, however, fail to capitalize on these overlapping results. They use pointer references to identify the corresponding data present in the cache. Since these references do not capture the semantics of the data they refer to, they are unable to support content based reasoning. Also, it becomes impossible to take advantage of the schema knowledge just on the basis of these references alone.

We need an altogether different approach to manage and efficiently utilize the cache while dealing with the spatial data in mobile GIS applications. Spatial queries in these applications tend to exhibit semantic locality due to the inherent location attribute associated with spatial data. We can exploit this fact for caching purpose and this is where semantic caching comes into picture. The idea of a semantic cache is very simple. Along with the data, the query description that generated the data is also stored in the cache [DFJ+96], [KB96]. These query descriptors are then used in determining and manipulating the contents of the cache.

1.2 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

[1] A concept, meaning, related data tend to be clustered together at the source.
[2] An architecture in which a disk-page is the unit of transport between the server and the client.

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 among other things, 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 as follows. First, we device a new replacement policy for semantic caching. Our replacement algorithm is based on semantic locality of reference and it takes advantage of schema knowledge to efficiently manage the cache. It takes into account three parameters (cache hits, communication costs and cache usage) and tries to optimize along these dimensions. As a replacement mechanism, we formulate the problem as an integer linear program and then solve a relaxation to obtain a fractional optimal solution. We then propose an infrastructure that implements semantic caching in mobile environments. Our system uses the existing relational database framework to manage semantic queries. We develop a prototype of the system as a "proof of concept" that captures the essential elements of semantic caching and demonstrates its working. Our simulation study shows that our model performs much better than the contemporary replacement algorithms proposed in the literature. We also suggest innovative ways of efficiently answering queries from local cache with the help of previously stored queries.
We can summarize the contribution of this thesis as follows:

- System architecture for semantic caching in mobile environments.
- Techniques for semantic query processing.
- Cache management strategies.
- Cache replacement algorithm.

2 Semantic Caching

2.1 Concept

The idea of semantic caching is very simple. Along with the data, the queries that generate those data are also stored in cache. These queries, also known as semantic descriptors, are then used to determine whether a new query can be answered from the existing data present in the cache. To answer a new query, we compare its semantics with the set of previously stored query descriptors. The result of these comparisons will determine whether the answer to the new query is already present in the cache. The essential difference between traditional page caching and semantic caching is that in the former, pointer references are used to identify the data present in the cache, while the latter makes use of query descriptors to identify the data.

To understand the concept of semantic caching in greater details, let us briefly walk through the mechanism employed in a conventional page caching scheme and then compare it with semantic caching. Our model is 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 databases. To enhance the query response time, the client application uses a local cache where the data from the previously issued queries are stored with the anticipation that the cached data will be accessed frequently in the near future. The query issued by the user is executed at the server and the results are sent back to the client. The client receives data as a set of pages from the server. Apart from the requested data, the pages may also contain some additional data. The idea behind sending additional data is based on the assumption that spatial locality of reference holds true and that it is quite likely that this additional data clustered 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 reducing the overheads incurred by the server on every single client request. The pages fetched from the server are stored in the local cache and pointer references are created in order to quickly locate and access these pages. Whenever a new query is issued, a check is first made to see if the reference to the requested data matches with any of the references already present in the cache. 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. Important point to note is that the page references simply act as pointers to the pages in cache. They do not implicitly 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 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 pointer references, but also contains some meaningful information about the data they refer to. This added functionality of the query descriptors is used to determine whether a 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 query descriptor is not just a simple pointer reference but in fact, an intelligent reference which conveys some important information regarding the data it refers to. For the purpose of semantic caching the query descriptors are also stored in cache. If a new query can be mapped to a set of query descriptors previously stored in cache, then it is certain that the results of this new query are present in the cache. The exact mapping procedure is a complicated task and is explained in great details in section 3.

2.2 Benefits

Caching query descriptors along with the query results have some important benefits:

- It adjusts grouping of queries to the requirements of the incoming query so that no irrelevant data is cached along with the relevant ones. This greatly reduces the overhead in managing the cache.
- Minimizes the cost of cache lookup due to the compact representation of the cache content based on semantic query descriptors.

- Adapts dynamically to the patterns of the user queries rather than caching static clusters of tuples [DRWS].
- Can provides partial and/or approximate answers to the queries quickly. This can be very useful in applications that have some tolerance to the accuracy of the results obtained.

2.3 Issues

The research issues in caching have traditionally been selecting the cache granularity (page level, tuple, hybrid, object and attribute caching), cache coherency, cache replacement and cache management. But while dealing with semantic caches two other issues come to the forefront. The first 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 and answering queries from views. Once it has been determined that the results of the new query are in cache, the next task is to extract the results from the cache. Larson and Yang in [LY87] give the theoretical foundation to address this issue. In this section we briefly touch upon the issues of cache granularity and cache coherency while the issues of query containment, result extraction, cache replacement and cache management are thoroughly dealt with in the later sections.

2.3.1 Cache Granularity

Dewitt et. al. in [DFMV90] compare and contrast page-level caching with single object granularity caching and their experiments show that page-based caching mechanisms require a high degree of clustering among the items present in the page to be effective. Chan at. el. in [CSL98] investigate three different levels of granularity of caching a database item in an object-oriented database (OODB), namely, attribute caching, object caching. and hybrid caching. Attribute caching involves caching of frequently accessed attributes of database objects, entire objects are cached in object caching while in hybrid caching only frequently accessed attributes of frequently accessed objects are cached. Their simulation study shows that page-based caching is not suitable in mobile environments. [DFJ$^+$96, KB96] use semantic regions as the granularity of cache, that are nothing but the results of previously executed queries. They also maintain the query descriptions in the cache. In our discussion we using a semantic segment as the granularity of cache. At the logical level, the semantic cache consists of distinct semantic regions, i.e. query results, that may vary in size. Physically, we could represent each semantic region by one or more pages. Notice the difference between a semantic cache and a traditional page cache where in the former, each page consists of answers to queries while the latter consists of pages that contain data from the base relations.

2.3.2 Cache Coherency

Data stored in the local cache can become stale or outdated, if the corresponding data present at the remote server is updated. Similarly, data modified locally has to be reflected back to its base database at the remote location and all other clients using that data have to be notified of the updates. In order to maintain consistency, a cache coherency scheme has to be devised that will ensure that cached data are consistent with those stored at the server. Barbara and Imielinski in [BI94] present a complete taxonomy of different cache invalidation strategies. Cai et. al. in [CTO97] re-examine the cache coherency problem in a mobile computing environment in the context of relational operations and also propose a taxonomy of cache coherency schemes. In our presentation, we assume that the data is read only or is updated rarely and hence cache coherency issues are not considered.

3 Semantic Query Processing

The issue of semantic query processing has been a widely researched topic in the academic world. Although the concept of semantic caching is very simple, its actual implementation is a rather challenging task. Unfortunately, most of the work in the literature is focused on the concept with either no implementation details or with over simplifying assumptions. The important issues in semantic query processing are:

1. Deciding whether the result of a query is present in cache (Query Containment Problem),
2. Extracting results of a new query from the cache (Result Extraction Problem),
3. Query trimming and remainder query calculation (Partial Query Match Problem).

There are cases when the result of a query is present in the cache in its entirety, but it may be impossible to extract the results without any additional information and hence the distinction between the first and the second case. Also, sometimes partial query results may be present in cache. In order to use these results, we may need to modify the query so that we can extract the partial results from the cache and at the same time calculate the remainder query to fetch the remaining part of the resultset from the remote location.

In this section we start by introducing the terminology and the various definitions we use. Then we move on to the literature survey where we review query containment problem and the related topics of query folding, query satisfiability and implication. Then we explain how we apply the techniques of query containment to our problem domain. And finally we conclude this section by illustrating techniques for answering queries from cache.

3.1 Definitions and Terminology

The following terminology will be used throughout our presentation:

Query Q : A query Q corresponds to an expression in a query language. An example of a query in SQL could be "SELECT name, age FROM table1 WHERE age > 40".
Answerset A : 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 the database.
Temporary T : 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 $A_T = A_{T_1} \cup A_{T_2}, \ldots, A_{T_n}$. In our discussion we use T interchangeably to mean either the set of queries or its corresponding answerset and trust that the context will disambiguate the meaning.
Location Aware Query (LAQ) : A query that contains location as one of the attributes in its predicate.
Location Dependent Query (LDQ) : A Location Aware Query whose result is dependent on the current location of moving object that issued the query.
Non Location Related Query (NRL) : A query that is niether location aware nor location dependent.
Variables in lower cases like x, y, z are boolean variables while constants are represented by a, b, c .
θ belongs to the set of operators given by $\theta \in \{=, <, \leq, >, \geq, \neq\}$.

3.2 Literature Survey

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 out whether $A_Q \subset A_T$. In order to show this containment, we can test the implication $Q \longrightarrow T$ or equivalently, 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. They discuss the satisfiability and equivalence of conjunctive mixed predicates containing inequalities of the form $(x\ \theta\ c)$, $(x\ \theta\ y)$ and $(x\ \theta\ (y + c))$. They give a $O(n^3)$ 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, where n stands for the number of inequalities present in the predicates. 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 by $\neq$ 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. 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 of acyclic conjunctive queries. Query containment can be viewed 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 is 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 foldings, 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 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 minimal rewritings and finding complete rewritings. They express the original query in terms of materialized views; "minimal" implies minimum number of views required to express the query while "complete" implies the entire query can be wholly expressed in terms of views only. They show that all possible rewritings can be obtained by considering containment mappings from the views to the query. Secondly, when both the query and the views are conjunctive and don't involve built in comparison predicates, the problem is NP-complete. And finally, the problem has two independent sources of complexity; first, the number of possible containment mappings and second, 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. Using redundancy removal algorithms and containment mapping enumeration algorithms, they give the algorithm for finding rewriting of a query. They also analyze their complexity with disjunction and built-in predicates.

Godfrey and Gryz in [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.

Levy in [Lev00] survey the state of art on the problem of answering queries using views, and synthesize 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.

Finkelstein [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. They however, do not assume any semantic knowledge of the contents or sequencing of the queries. Theirs is a generalized scheme for query processing. In our work, we extend their query model and algorithm and also incorporate the semantic knowledge for processing queries.

3.3 Addressing the Query Containment Problem

In our application scenario, we assume the SQL queries to be simple PSJ (Project-Select-Join) queries and restrict the domain of numbers to integers. Since we are dealing with mobile navigation application, we implicitly assume that most of the queries issued by the user are either location aware or location dependent. These simplifications do not alter the nature of the problem at hand, but help us in simplifying the techniques and algorithms we apply for query processing.

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

An SQL query predicate can be represented as a formula consisting of disjunction of conjunctive clauses. It is easy to reduce simple SQL predicates to disjunctive normal form. 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
Hence in order to prove the implication $(Q \longrightarrow T)$ we try to show its negation $\neg(Q \longrightarrow T)$ to be FALSE. We use the following simplification to achieve our objective.

$$
\begin{aligned}
&\neg(Q \longrightarrow T)\\
&\equiv \neg(\neg Q \vee T)\\
&\equiv Q \wedge \neg T\\
&\equiv Q \wedge \neg(T_1 \vee T_2 \vee \ldots T_n)\\
&\equiv Q \wedge (\neg T_1) \wedge (\neg T_2) \wedge \ldots (\neg T_n) \qquad (1)
\end{aligned}
$$

Hence by testing the satisfiability of Eq. (1), we can find out if the query Q can be satisfied by queries in T. If the expression evaluates to FALSE, it implies that the implication $(Q \longrightarrow T)$ is TRUE, which means that the answer to the query is present in cache.

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].

3.3.1 Algorithm IMPLICATION

Input: A query formula Q consisting of disjunctive clauses represented as $Q = C_1 \vee C_2 \vee \ldots C_m$, and a set of queries T that is previously stored in cache. $T = \{T_1 \cup T_2 \ldots T_n\}$
Output: Boolean value TRUE if the answer to query Q given by $A_Q \subset A_T$, FALSE otherwise.
Let $T' = \neg T_1 \wedge \neg T_2 \ldots T_n$.
In order to test the implication $Q \longrightarrow T$ to be true we test its negation $\neg(Q \longrightarrow T)$.
for all $C_i \in Q$ do
    if (SATISFIABLE(C_i, T') == TRUE)
        continue;
    else
        return FALSE
return TRUE

3.3.2 Algorithm SATISFIABLE

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 an atom. An atom A is a predicate of the form $(x\ \theta\ c)$, a simple condition or $(x\ \theta\ y)$, a

connective condition where,
x, y are variables, 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 .
  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.

4 Answering Queries from Cache: Issues and Techniques

Applying this algorithm, we test whether the answer to a query is already present in cache. Although this technique is fairly simple and straight forward, it is difficult to implement it for the following reasons.

- With the increase in the number of queries stored in cache, the complexity of the algorithm increases exponentially. This is due to the fact that the algorithm SATISFIABLE takes as one of its inputs, disjunction of queries stored in cache. Individual query itself could consist of conjunctions of numerous predicates. This leads to an explosion of boolean expressions that need to be tested for the satisfiability condition.

- Though the answer to a new query may be completely present in cache, it may be impossible to extract the answer without additional information. Consider the following example:

  Query Q_1 = SELECT name, age FROM table WHERE (age > 10)
  Query Q_2 = SELECT name, age FROM table WHERE (age > 10 AND height < 6')

  It can be shown that $Q_1 \longrightarrow Q_2$. But since we do not have information regarding the "height" attribute in query Q_1, it is impossible to retrieve the answerset to query Q_2 without any additional information.

Larson and Yang in [LY87] define coverage and derivability and give the theoretical foundation to address the above problem. Let Q be the query and let A_Q be its answer set.

$$A_Q = \{t|Q(t)\} \tag{2}$$

Let T be the set of queries in cache such that $T = \cup T_i,\ T_i \in \{Q_1, Q_2, \ldots, Q_n\}$.

$$A_T = \{t|T(t)\} \tag{3}$$

They define Coverage as:

If for all $t \in A_Q$, $A_Q \subseteq A_T$, then T is said to cover Q.

And Derivability is defined as:

A function F such that $A_Q \equiv F(T)$ where $F(T)$ is the Cartesian product of all $T_i \in T$.

Simply put, for tuple coverage all tuples in A_Q must be present in the Cartesian product of some subset of the queries in T. And derivability means that it should be possible to extract the exact results from T such that they are equivalent to the results had would have been obtained directly by executing the query Q against the remote database. It can be shown that derivability implies coverage and that coverage is a necessary but not sufficient condition for derivability. We use the following technique to answer queries from cache.

Evaluate the boolean expression in Eq. (1). 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.

$$A_Q = B \cup (Q \wedge T) \tag{4}$$

Issues and Solution: For arbitrary large $|T|$, the expression B becomes too complex to evaluate and the computations involved in testing its satisfiability increase exponentially.

We tackle this problem in the following manner. We limit the number of queries in cache that are used in testing the satisfiability of a given query by grouping them based on their attributes. A signature is created for each query based on the attributes present in its predicates. The signature of the query in question is checked against those of all queries in the cache. Queries whose attributes do not subsume the attributes of the new query are eliminated from further consideration. In this way we can limit the number of queries that participate in the algorithm. This step, also known as the filtering step is just a table lookup and is performed very quickly and without much overhead. This is a conservative approach in that, we may never find all the mappings present in the cache, but it is guaranteed never to find a false mapping. It might so happen that a query can be satisfied from the cache but the algorithm will not detect this mapping. This is the primary technique we adopt in our experiments to make the satisfiability procedure more manageable.

Also, there are other techniques that can be possibly employed depending on the type of mobile application. These include:

- 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 $Q_s \cup T$. 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. As an example, predicates including numbers of the $(x + y)\ \theta\ c$ can be reduced to the form $x\ \theta\ c'$ where c' can be calculated as $\text{MAX}(y) + c$. Now it is much faster to solve simple predicates of the type $x\ \theta\ c'$ than the original ones.

Note that these techniques can also be used to answer queries partially from the local cache.

5 System Architecture

Our semantic caching architecture consists of the following modules.

- Semantic caching module
- Execution module
- Replacement module

The first three modules heavily interact with the local cache that essentially stores the frequently required data by the user of the system, i.e., the client.

5.1 Semantic Caching Module

The main purpose of this module is to determine if a new query can be answered locally from the cache. The main component of this module is the solver that implements the algorithm to test query satisfiability. The solver also determines if and how a query can be mapped to the existing queries present in the cache. When a new SQL query enters the system, the SQL parser [Gib] validates the SQL query for its syntax. If the syntax is valid, the query request is passed on to the solver. The solver then applies the satisfiability algorithm. If it is determined that the query can be completely answered locally, the query is passed on to the executor for further processing. If the query can only be partially answered from the cache, the solver massages the query and splits it into two parts, one that can be answered locally and the other that cannot, called the remainder. Partially answerable query will be executed locally while the remainder query will be sent to the remote server for further processing.

5.2 Execution Module

The function of the executor present in this module is fairly straightforward. It receives queries and the information of what action has to be taken on those queries from the semantic caching module. For queries that cannot be answered locally, the executor sends them to the remote server. For split queries, the partially answerable part is executed locally while the remainder part is sent to the server. The results of both parts are then combined and sent to the client. In cases where approximate results are tolerable, locally stored results can be sent to the client till the remainder part of the answer set is fetched from the remote location. In this way a better response time can be achieved by answerset pipelining.

5.3 Replacement Module

The function of the replacement module is to efficiently manage cache. It determines if there is enough room in cache to fit in the next query result. If not, then it determines the relative worth of this new query as compared to those already present in cache. The determination of relative merit of a query is based on cost benefit analysis. The specifies of this analysis are explained in the section 8.1. Once relative merits have been established, a replacement algorithm is applied on the cache. The algorithm chooses a victim to be evicted from the cache whose relative merit is perceived to be the lowest. The candidate query is purged out of the cache. The process is repeated till either there is enough space in cache to accommodate the new query or the new query's relative merit becomes the lowest in the group, in which case it is not cached at all.

6 Algorithm (Pseudocode)

This algorithm describes how the processing of a query takes place in the system.

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

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

ProcCacheManagement

- Run query admissibility test to determine if the user query is worth caching
- if the query is worth caching,
  - 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 along with its results

7 Cache: Structure, Operations and 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. In the context of semantic caching, the queries present in the cache may have to be deleted, trimmed, coalesced or decomposed. In this section we start with a general description of the structure and operations performed on cache. Then we move on to the strategies for cache management and then go on to elaborate on the issue of cache replacement.

7.1 Semantic Cache : Structure

A semantic cache is a structure where queries along with their results are stored. Each resultset can be viewed as a rectangular semantic region described by its corresponding query. Internally, the cache is implemented as a two level relational database structure. At the first level, the frequently required data is stored in a relational table. At the second level, the query descriptors that generate those data are stored in a separate relational table. The descriptor table is used by the semantic caching module to determine if a new query can be answered locally. And this is the beauty of the entire structure. Instead of querying the large data table, we always query the relatively smaller descriptor table to determine if and how a query can be answered from the cache. Storing query descriptors in database offers considerable advantages in terms of ease of access and manipulation. Since semantic caching is essentially manipulation of query descriptions, it makes perfect sense to store them in such a manner that its retrieval and manipulation becomes extremely efficient and convenient. Besides, as compared to the data table, the descriptor table is much smaller in size since it contains only the description of the query issued by the user. The format of the query descriptors stored in the descriptor table is not always the same as that of the original query. The is due to the fact that queries are massaged and converted in a format that can be easily accessed and manipulated by the solver in the semantic caching module. Also, the queries are modified over time to accurately reflect the mapping to the data present in the data table. The data regions may be coalesced, trimmed, added or deleted over time. However, queries in cache are always rectangular regions. If not, they are first converted into a rectangular form. For example, a query of the form $Q = (a < 10)(b < 5) \vee (a < 5)(b > 0)$ can be converted into rectangular form by decomposing the query into two separate queries. i.e., $Q : Q_1 = (a < 10)(b < 5), Q_2 = (a < 5)(b > 0)$. The rule is, disjunctive queries are always decomposed to form rectangular regions. Conjunctive queries are replaced by its Minimum Bounding Box (MBB). At the time of merging, the queries to be merged are replaced by a single rectangular query.

7.2 Semantic Cache : Operations

The cache can be thought of as a black box that exposes certain methods to the outside world, that assist in accessing and manipulating the internal state of the cache. A semantic cache is a structure where queries along with their results are stored. Each resultset can be viewed as a semantic region described by its corresponding query. Operations on cache are essentially the operations to query and manipulate the semantic regions present therein. We use the generic term query to mean either the query descriptor or its corresponding semantic region. The following operations are allowed on cache.

- Addition: A new query can be added to the cache. The added query need not necessarily be a user query. It could be a modified query so as to assist in easy manipulation later on.

- Deletion: A query can be deleted in totality. Meaning, the granularity of deletion is a semantic segment. This is a two step process. In the first step, the query descriptor is removed from the descriptor table. In the next step the reference count of all associated tuples in the data table is decremented by one. Now the entire data table is scanned on the reference count column. Tuples whose counts are zero, indicate that they are no longer related to any of the query descriptors and hence are purged out of the cache. The change in the size of the data table is the number of tuples actually purged out of the cache and not the number of tuples associated with the query descriptor. It is important to note that though a data tuple may be related to numerous semantic regions, a single invocation of the deletion operation, results in purging of a single semantic segment.

- Merge (Coalesce): Merge is a specialized operation. It is used primarily to reduce the number of semantic segments present in the cache, thus facilitating cache maintenance. More importantly, it is used as a technique to reduce the total number of query predicates stored int the cache, whose benefits are explained later in this section.
- Decomposition: In some cases where a semantic segment is very large, it may be advisable to break it into smaller segments for ease of processing. It may also happen that a part of the query is utilized heavily and it needs to be cache, while its remaining part is not used much. In that case we decompose the query into two parts and subsequently treat these two parts as two different queries.

7.3 Cache Management Strategies

- Reduction in the number of queries to be processed:
  A large number of queries in the cache causes an exponential growth in the number of equations to be processed by the solver for testing the satisfiability. We restrict the number of queries so that they can be easily handled. If however queries in cache are unrelated, i.e., coming from different relations or having independent attribute sets, we can relax on the number of queries to be stored, since these unrelated query sets do not contribute to the exponential growth in the number of equations to be processed. This is achieved by clustering the queries based on their attributes and creating a signature for each cluster.
- Merging queries:
  One technique of reducing the number of queries in the cache is by merging related queries into a single query. Assuming each attribute takes values in the integer domain and has a range, we can club the ranges together. Real ranges can be converted into integer ranges and non-rectangular queries can be enclosed in a rectangular minimum bounding box. We can also process ranges. For example to merge ranges, instead of sending, say, 100 predicates to the server, only the ranges of all the variables involved can be sent. Since the predicates involved are restricted to be of the form $(x\ \theta\ c)$ and $(x\ \theta\ y)$, the number of ranges to be sent to the server is $\mathrm{O}(n^2)$ where n is the number of variables. Thus we can reduce the amount of data transferred between the source and the client.
- Handling different types of queries:
  In our presentation we handle predicates of the form $(x\ \theta\ c)$ or $(x\ \theta\ y)$. But the following types can also be handled without much complication. Predicates of the type $(a.x\ \theta\ c)$ where $a, c \in I$. i.e Linear predicates with integer coefficients. We can view them as a system of equations with linear constraints and they can be formulated as a linear programing problem (LPP) . There are a couple of solvers in the market like lpsolve that can solve these type of equations. Alternatively, we can formulate the problem as finding the interior (feasible region) of a polygon, if it exists. We could simplify the problem by creating a minimum bounding box for the polygon and then work with the MBB. Quadratic and other polynomial constraints can be tackled using the theory of Groebner bases.

8 Cache Replacement

8.1 Theory and Assumptions

When a new query comes in, a decision has to be made whether to admit that query in the cache. 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 to make room for the new query. Making the right selection of target regions to be removed is pertinent to efficient management of cache.

A 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 variety of factors need to be considered while designing an effective replacement strategy. Prominent among them are, hit rate, response time, query execution time and data transmission time. We need to decide what our optimization strategy is going to be based on the priorities we set for each of these parameters. In the literature different replacement schemes have been proposed that optimize one or more of these parameters, but no consensus has been reached as to which parameter is the best indicator of the performance of the system. We define these parameters as follows:

Response Time r_i : Time interval between the query request sent by the client and the entire resultset sent back to the client.
Query Execution Time e_i : Time required to executed the query against the database.
Data Transmission Time t_i : Time incurred in transmitting the data from the source to the destination.
Cost of Query c_i : Cost of execution of query Q_i.
Size of Query s_i : Size of resultset of query Q_i.

Conventional page caching replacement policies try to maximize the probability of cache hits by keeping data in the cache that have higher probability of being accessed in the near future. They make the assumption that response time of the system is directly related to the cache hit ratio. But unfortunately, this is true only when the size and cost of the transfered objects are constant. In other words, if the amount of data transferred per query request and the cost of execution of each query are variable, then response time is a function of size, cost and hit ratio. In our application scenario, both size and cost are variables since we are essentially dealing with variable sized semantic segments whose cost of execution can vary drastically. Hence we contend that cache hit ratio will not be a good indicator of the performance of the system. Besides, since it is quite possible for queries to be partially answered from cache, it is hard to precisely define hit ratio. We base our foundation on the assumption that the amount of data transferred will be the most important factor influencing the performance of the system. We can drastically improve the response time of the system by reducing the amount of data transferred between the source and the client. In a wireless environment, the data transmission time is dependent on a multitude of factors, most prominent among them being latency, bandwidth and the size of data. High bandwidth and low latency will imply low data transmission time. Response time depends on query execution time and data transmission time. Complex queries involving multiple joins may take more time to execute than simple select, project queries. We will make the assumption that however computationally expensive, the query execution time is much smaller than the data transmission time. Hence we will ignore the execution time while considering its impact on the response time. Now, the response time will be dependent on the data transmission time and hence on latency, size and bandwidth. Assuming uniform latency and bandwidth across the network, the response time will now be dependent only on the size of the query. Hence we will use the size of data transferred as the primary metric to judge the overall performance of the system. Our goal will be to minimize the amount of data transferred from the remote source to the client.

8.2 Literature Survey

In the literature, the cache replacement strategies employed are based on a variety of cost models that incorporate 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.

In WATCHMAN [SSV96], they consider profit based replacement strategy in which each item in cache is assigned a metric which considers its average rate of reference, its size and execution cost of its associated query. The performance metric P is defined as

$$P_i = \frac{\lambda_i c_i}{s_i} \text{ where,} \tag{5}$$

- c_i is the cost of execution of the query Q_i
- s_i is the size of the retrieved set by query Q_i
- λ_i is the average rate of reference to query Q_i

In order to deal with the possibility of workload variations, WATCHMAN uses the last K inter-arrival times of requests to query Q_i to estimate λ_i

$$\lambda_i = \frac{K}{t - t_k} \text{ where,} \tag{6}$$

t is the current time and t_k is the time of the K-th reference. In the case of a new query where the average rate of reference in not available, the performance metric takes the form of

$$P_i = \frac{c_i}{s_i} \tag{7}$$

They then try to find the solution to the optimization problem of

$$\text{maximize} \quad \sum_{i \in N} p_i c_i, \tag{8}$$

$$\text{subject to} \quad \sum_{i \in N} s_i \leq S, \tag{9}$$

where S is the total cache size, and $p_i = \frac{\lambda_i}{\sum \lambda}$
With the assumption that it is always possible to fully utilize the space S, they relax the constraint to $\sum s_i = S$. Thus the problem becomes the fractional Knapsack Problem which has an optimal greedy solution.

Chan at. el. in [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.

Ren and Dunham in [RD98] suggests a dynamic LRU algorithm called the *d-LRU*. Since semantic regions grow or shrink in size dynamically, each of them is time-stamped according to its most recent access or modification. The cache replacement strategy adopted 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.

In [RD99], Ren and Dunham 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) .

Dar et. al. in [DFJ+96] suggest a replacement strategy based on the semantic locality. For each semantic region in the cache, its distance to the most recently accessed region is calculated. The region with the largest distance is the candidate segment for replacement. The semantic regions are assumed to be rectangular in shape and the distance is measured from the center of the rectangle. The problem is, every time the replacement algorithm is run, the distances of all segments in cache have to be recalculated.

Keller and Basu in [KB96] use the frequency of access of the semantic regions to determine the replacement victim.

8.3 Our Replacement Algorithm

Goal:

> **To minimize the cost of servicing the requests that cannot be completely answered from the local cache.**

Cost is measured in terms of time and time is directly proportional to the amount of data transferred from the source to the client. We propose a new function called the **guiding action selector (GAS)** that will assign a score to each query in the cache. The general form of the function is:

$$\text{GAS} = \alpha + s * f * \beta \tag{10}$$

where,

- s is the size of the query result transferred from the remote source,
- f is the frequency of access of the query,
- α and β are domain specific parameters.

In our case, since we are dealing with semantic distances, we incorporate the notion of proximity in the formula. The closer the query from the current location of the mobile object, the higher the probability of it being accessed in future. Hence in this case β is $\frac{1}{S_d}$ where S_d is the semantic distance between the location of the query and the current location of the moving object. Also, since semantic regions may merge or split over time, we need an accurate measure to determine the freshness of each region. It may so happen that a part of a region is heavily used while the other part is not. To account for this, we use α to indicate the freshness of each semantic segment.

With the help of experiments we show how our empirical guiding action selector function performs as compared to other traditional caching schemes.

8.4 Cache Replacement: Policy and Mechanism

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 application domain we are dealing with two entities, queries and their results. Results contain one or more records. In a single dimension, queries can be viewed as lines and records can be viewed as points on the line. Similarly, in 2 dimensions, queries are rectangular regions while records are points that lie in that region. To simplify the discussion we restrict the space to 2 dimensions only. Each rectangle is a MBB, that encloses all records present in the corresponding resultset. Queries may overlap each other. Their overlapping leads to one or more records lying in 2 or more rectangles. Each rectangle is associated with a "weight" and a "value". Weight can be defined as the sum of the sizes of all the records present in the rectangle while the Value is defined as the relative worth of a rectangle as compared to other rectangles. Any replacement algorithm to work efficiently, it needs to find the weight and the value of all the rectangles present. Once these parameters are known, the problem reduces to an optimization problem involving weight and value parameters with constraints on the sum of weights of all the rectangles. The problem of calculating weights is tricky and complex when overlapping rectangles are considered. The question is how should we calculate the weights of records that are common to more than one rectangle. For 2 dimensions, we could simply decompose the rectangles in such a way that there is no overlapping. Now the weight of each rectangle is simply the sum of weights of all points present in the rectangle. But this solution is too restrictive and it does not scale well in higher dimensions where the decomposition may lead to an exponential increase in the number of rectangles. To determine the Value of each rectangle, we need to know the probability of access and its frequency. If these two variables are known, we can have an algorithm that does cache replacement in an optimal way. Unfortunately for online algorithms, we neither have the probability of access, nor its frequency apriori. To get around this problem, we can use histograms to statistically find the frequency of access and its probability. We need to collect some history in order to generate and use these histograms.

Approximating probability and frequency of access
For approximating probability, we use the distance between the current location of the moving object and the location pertaining to the Location Aware query issued by the moving object. The smaller the distance between the two, the higher is the probability of the Location Aware query (rectangular region) being accessed in near future. Also, the direction of motion can play an important role in this. Mobile object moving towards a rectangular region has higher chance of querying records lying in that rectangle. The access frequency is calculated by recording the frequency patterns of the moving object over a period of time.

8.5 Problem Framework and Problem Formulation

We need an efficient mechanism to implement the cache replacement policy we adopt. From the replacement policy we get a relative merit of each query we execute. Now each query(rectangle) has 2 parameters, the size of the rectangular region and its relative merit. In the subsequent discussion, we will call these parameters "weight" and "value" respectively. Since we are constrained by the size of the cache and also the number of rectangle, 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. Also, let the limit on the number of rectangles 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, choose at most K rectangles that gives maximum value, provided the weight does not exceed W.
This problem can be modeled as the classical 0-1 Knapsack Problem with an additional cardinality constraint as shown by [CKPP97].

8.6 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[3], 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 $val(I_i) = \sum_{x=b_i}^{e_i} s_x$.

$$\text{maximize} \sum \text{val}(I_i), \tag{11}$$

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

[3]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 $w : n \rightarrow \Re$ and value $v : n \rightarrow \Re$, 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.

$$e_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Let x denote the indicator variable for node i such that

$$x_i = \begin{cases} 1 & \text{if node } i \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

The LPP formulation of this problem is as follows:

$$\text{maximize} \quad \sum_{i=1}^{n} v_i x_i, \tag{14}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq W, \tag{15}$$

$$x_i + x_j \leq 2 - e_{ij}, \tag{16}$$

$$x_i \in \{0, 1\} \tag{17}$$

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.

A sample experiment is performed to find a solution to the mixed-integer problem. Figure 1 gives the fractional optimal usage of the cache, while Figure 2 give the fractional optimal cost value achieved using the mixed integer program.

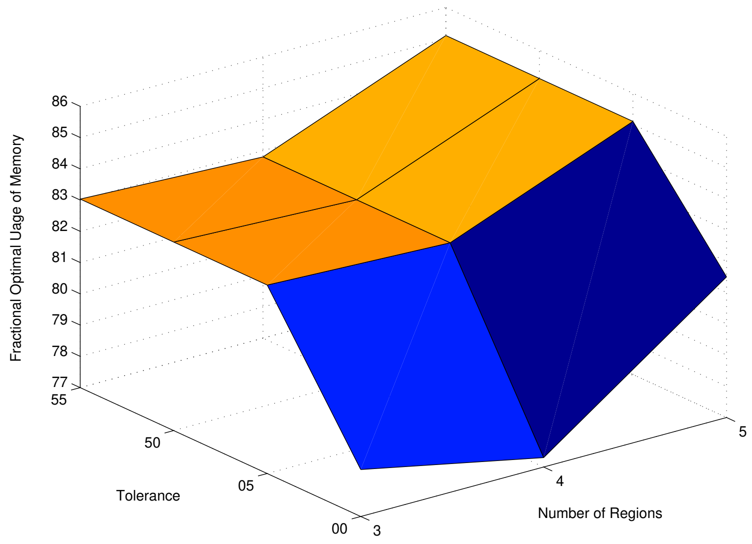

Figure 1: Fractional optimal memory usage

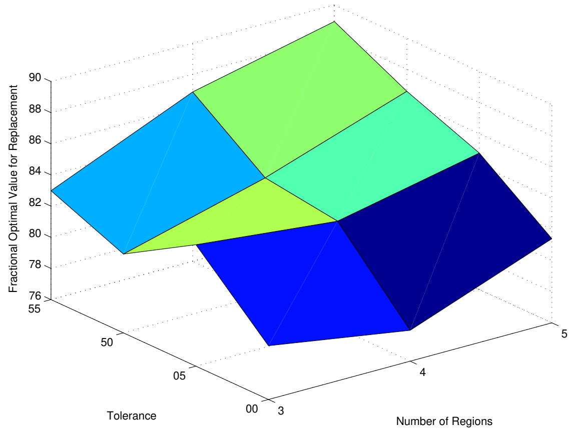

Figure 2: Fractional optimal value function

9 Experiments

9.1 Choice of the application

In order to test our replacement model for mobile environment, we conduct experiments in a controlled, simulated environment. The choice of our representative application is that of a mobile navigation system where a user is moving in a car from one location to another, following a predetermined path. It is a client server environment where the client (the user) requests information from the server through the navigation system present in the vehicle. The location of the server(s) is unknown to the client, but it is assumed that the client will have some wireless connectivity via which it will communicate with the nearest server(s) along the way. During the journey, the user keeps asking queries pertaining to his need and/or interests. The queries range from finding McDonald's that is closest to the current location of the vehicle, to finding any gas station along the way where the gas price is the lowest. The user asks queries intermittently, switching between think time and ask time. The on board system has a limited amount of memory that can be used as a cache to store most frequently requested data. Note that the data requested by the user could be of spatial nature as well, such as maps, location information etc.
For conducting the experiments in a simulated environment the following things are required.

- Workload
- Modeling the behavior of the moving objects
- Query execution guidelines

Workload: Datasets and queries are together referred to as the workload. We can use either real datasets obtained from different sources or artificially generate spatiotemporal data using a synthetic dataset generator. For our application real data sets are difficult to obtain and even those that can be obtained are too complex to work with. The complexity arises not only because of the huge size of datasets to be stored and manipulated, but also because of the complex structures of spatial objects. Also, in real datasets, parameters are more or less fixed depending on the real world behavior and it is extremely difficult to customize them. With the synthetic dataset generator it is easy to control various parameters related to the data and queries such as desirable cardinality, statistical distribution of various temporal and geometric features, etc. For generating datasets, we use the existing synthetic dataset generator GSTD [PT00] for this purpose. GSTD generates sets of moving points or rectangular data that follow extended set of distributions (random, Gaussian or skewed). The detailed analysis on the choice of the dataset generator is given in the next section. Extensive sets of domain specific queries are created that are location aware, location dependent or simply non-location related. Queries to be executed are then picked up from these sets in a manner determined by parameters like the expected selectivity of the chosen queries and overlap rate. The specifications of these parameters are included in the query execution guidelines to be discussed in the next section.

Modeling the behavior of the moving objects: For our application we use a two dimensional space called the workspace. The workspace represents the area on which the mobile object moves from one location to another. Using GSTD we generate sets of static points and regions to emulate real life objects like rivers, buildings, highways, hotels, gas stations etc. These static objects are collectively called the infrastructure [PT00]. The motion of the mobile objects in the 2D space is constrained by the infrastructure. With the help of GSTD we create trajectories to model the behavior of the mobile objects. Two points on the trajectory are chosen to represent the starting point and the destination for the mobile unit. A variety of parameters affect the behavior of the moving object. Prominent among them are the initial statistical distribution of the points and regions in the workspace, generation of trajectories, selection of source and destination points on the trajectory, direction of motion and speed of the vehicle. These parameters are controlled during the simulation to create a realistic scenario.

Query execution guidelines: The important factors that affect the entire experiment are the query execution guidelines. These are, the frequency of query issuance, selectivity of chosen queries, overlap rate and the type of queries asked (Location related, Location aware, general non location related queries).

9.2 Advantages of using GSTD

By providing the number of moving objects as an input to GSTD, it can generate a set of trajectories, one for each moving object. The motion of each object is simulated by specifying a number of control parameters. The direction of motion and the initial distribution can be specified. The algorithm generates a set of tuples of the form $< x, y, t >$ where (x, y) gives the location of the object on a 2D plane at time t. The trajectory of the moving object is creates by joining consecutive points by linear interpolation. The algorithm also provides ways to create static rectangular regions of different sizes with different statistical distributions. These objects (collectively called infrastructure) can be view as bridges, hotels, buildings, road networks etc. to give a realistic view of the real-world landscape. The motion of the moving object is constrained by these infrastructure objects, just as in a real-world.

Here is a brief summary of other synthetic dataset generators available.
In [YW03] Wolfson and Yin introduce a method to generate synthetic spatio-temporal information called pseudo trajectories of moving objects. They obtain real speed variations by actually driving through the streets of Chicago noting the speed at regular intervals and then superimpose these speed patterns on a randomly selected route to generate the trajectory of a moving object. Though simple and elegant, this method seems to be too restrictive and localized and may not be result in datasets that are truly representative of

the real trajectories generated by moving objects.

With the City Simulator application [Cit], it is possible to simulate the motion of upto 1 million objects moving in a city, driving on the streets, walking along the sideways and even walking up and down the floors of a building. It can even simulate traffic conditions on the road. The customizable parameters are number of objects in the experiment, entry and exit probabilities of these objects in the buildings, up/down movement of object in buildings and the scatter probability. However, since it is not possible to control individual speed or direction of a moving object, this method seems to be far less suitable to model mobile navigation application like ours. Secondly, although most of the parameters are configurable at the start of the experiment, they change when a certain threshold on the number of people in the building is reached. After that we do not have control over these parameters. This application is primarily designed for the evaluation of database algorithms for indexing as in LOCUS [MK02].

In [SM01], Salgio and Moreira come up with an interesting model to synthetically generate datasets. They contend that real-world objects do not move in a completely random fashion. The surrounding environment affects their actions and their behaviors are guided by some goals. On similar lines, they argue that generating completely random datasets and trajectories would not be a representative of real-world moving objects. As a modeling scenario, they use the example of fishing ships that go in the direction of the most attractive shoal of fishes while trying to avoid storm areas. Fishes are themselves attracted by plankton areas. Ships are moving points; planktons and storms are moving areas with fixed centers but variable shapes while shoals are moving regions. Although the approach is novel, the applicability of this model seems to be limited to a few restricted scenarios where attraction and repulsion between various objects can be well defined and is meaningful. Also, in its current state, the model cannot be applied to road networks where most of the infrastructure like buildings, roads, rivers etc., is static.

10 Experimental Settings and Results

For our experiments we need Dataset and Query workload.
We assume that all objects move in a 2D space within the area 1000*1000 sq. units. Controllable parameters of the dataset and their respective settings are:

- Type of object: Point object.
- Cardinality of moving object: Since our main emphasis is on semantic caching, we select a single moving object for this purpose.
- Speed of moving object: The speed of the moving object is varied randomly during the course of its journey. We specify the statistical distribution of the speed as either random, Gaussian or skewed.
- Direction of motion: We randomly vary the direction of motion (north/south/east/west/random).
- Initial position of the object: Follows Gaussian, skewed or random distribution.
- Frequency of updates: The frequency at which the observation regarding the moving object (current-location,current-time) is recorded. This is chosen using Gaussian or random statistical distribution. We record 30-100 points for every trip.
- Number of queries: On an average, we record around 40 observations per trip, i.e., on an average a user asks around 40 queries along the way.

Workload specifications
We consider the following relations for our query load selection.

- Restaurants: Location-x, Location-y, Restaurant-Name, restaurant-ID, number of tables, Occupancy, City, State, Zip
- GasStations: Location-x, Location-y, Station-Name, Station-ID, Gas-Ordinary, Gas-Medium, Gas-Premium, City, State, Zip

	Performance		Load		
Case	HitRate	DateTransferred	CacheSize	Db size	Num of Queries
LRU					
NEW					

Table 1: Comparative analysis of LRU versus New Location based cache replacement scheme

Query workload selection parameters are:

- Type of queries: Rectangular queries (Select and Project operations only)
  Locations Aware Queries: e.g. "SELECT NAME FROM GASSTATIONS WHERE Zip = '21227"'.
  Location Dependent Queries: e.g. "SELECT NAME, Location-X, Location-Y FROM GASSTATIONS WITHIN 5 miles of Current Location".
  Non Location Related Queries: e.g. "SELECT num of rooms from Hotels where name = 'Holiday Inn"'.
- Temporal/Prediction queries (Optional): Queries involving time as one of the predicates: e.g. "Find Hotels near Current Location after 10 minutes".
- Selectivity of results: Number of tuples present in the result. We place a max cap on the number of tuples fetched, depending on the size of the cache. It ranges from 25-50 percent of the cache size.
- Overlap rate: If there is no overlap between the queries (and hence their corresponding results), we would not derive any benefits from caching. We define the overlap rate to be the rate at which the queries overlap each other. We vary the overlap rate to be between 25 - 4 0 percent. This means that at least one in four query overlaps with one or more previous queries.
- Number of queries issued: We keep the range between 30 and 60.
- Frequency of issuance: The frequency of query issuance is tied to the observation points used to generate the trajectory. We randomly select points from the set of observations points and these points act as time instances when the query is issued by the user.

10.1 Results

Table 1 summarizes the results of the experiments.

Since may a times we have partial query results that can be obtained from cache, is not clear what the definition of hit rate should be. We consider 3 cases of hit rate.

- **Hit Rate One**: In this case, to qualify as a hit, at least one record should be obtained from local cache. This is the most liberal definition of a cache hit.
- **Hit Rate All**: In this case, all records requested by the query should be obtainable from local cache. Is is equivalent to page hits.
- **Hit Rate Mid**: In this case at least fifty percent of the records should be obtained from local cache. This seems to be the most reasonable of hit rate definitons for our purpose.

Figure 3 shows the results of the 3 hit rates along with the data transfer rate for LRU while figure 4 shows the same results for NEW.

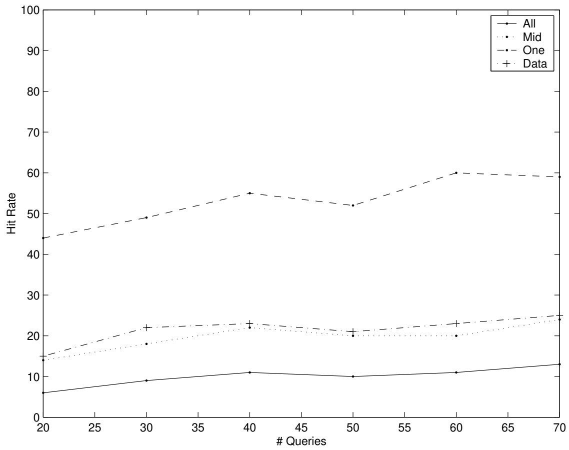

Figure 3: Hit Rate Versus Number of Queries - LRU

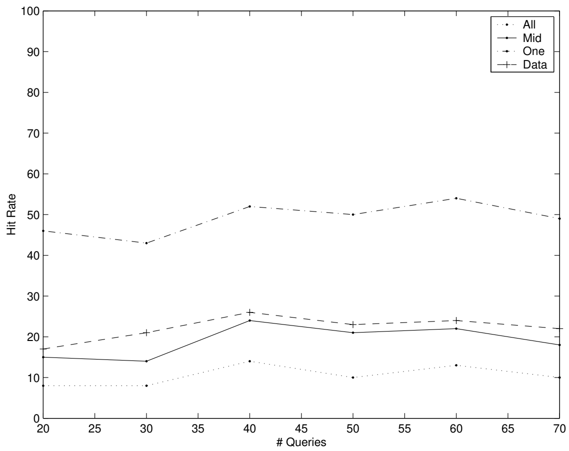

Figure 4: Hit Rate Versus Number of Queries - NEW

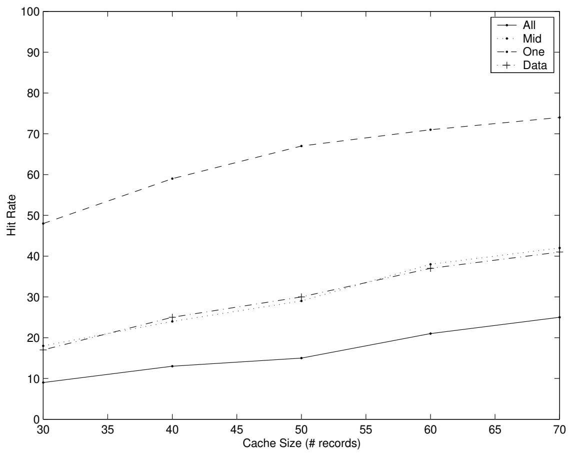

Figure 5: Data Transferred Versus Cache Size

References

[BI94] D. Barbara and T. Imielinski. Sleepers and workaholics: Caching strategies in mobile environments, 1994.

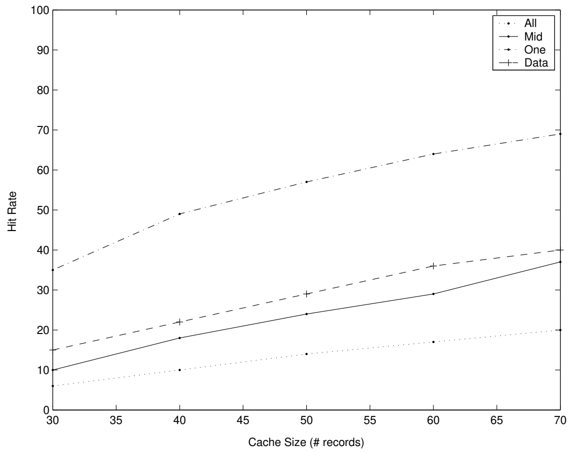

Figure 6: Data Transferred Versus Cache Size - NEW

[Cit] CitySimulator. http://www.alphaworks.ibm.com/tech/citysimulator.

[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.

[CTO97] J. Cai, Kian-Lee Tan, and Beng Chin Ooi. On incremental cache coherency schemes in mobile computing environments. In Alex Gray and Per-Åke Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 114–123. IEEE Computer Society, 1997.

[DFJ+96] Shaul Dar, Michael J. Franklin, Björn T. 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.

[Gib] Pierre-Yves Gibello. http://www.experlog.com/gibello/zql.

[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.

[MK02] J. Myllymaki and J. Kaufman. Locus: A testbed for dynamic spatial indexing, 2002.

[PT00] D. Pfoser and Y. Theodoridis. Generating semantics-based trajectories of moving objects, 2000.

[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.

[SM01] Jean-Marc Saglio and Jose Moreira. Oporto: A realistic scenario generator for moving objects. *GeoInformatica*, 5(1):71–93, 2001.

[SSV96] Peter Scheuermann, Junho Shim, and Radek Vingralek. WATCHMAN : A data warehouse intelligent cache manager. In *The VLDB Journal*, pages 51–62, 1996.

[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.

[YW03] Huabei Yin and Ouri Wolfson. Accuracy and resource consumption in tracking moving objects: Symposium on spatio-temporal databases, santorini island, greece, 2003, 2003.