

# QUERY MELTING

## *A New Paradigm for GIS Multiple Query Optimization*

Haifa Elsidani Elariss, Souheil Khaddaj and Darrel Greenhill  
*Faculty of Computing, Information Systems and Mathematics, Kingston University London  
Kingston upon Thames, U.K.*

**Keywords:** Query Optimization, Query Melting, Proximity Analysis, Dynamic Complex Queries, Large-scale GIS Servers, Visual Query Languages and Mobile GIS.

**Abstract:** Recently, non-expert mobile-user applications have been developed to query Geographic Information Systems (GIS) particularly Location Based Services where users ask questions related to their position whether they are moving (dynamic) or not (static). A new Iconic Visual Query Language (IVQL) has been developed to handle proximity analysis queries that find  $k$ -nearest-neighbours and objects within a buffer area. Each operator in IVQL queries corresponds to an execution plan to be evaluated by the GIS server. Since commonalities exist between the execution plans, the same operations are executed many times leading to slow results. Hence, the need arises to develop a multi-user dynamic complex query optimizer that handles commonalities and processes the queries faster especially with the large-scale of mobile-users. We present a new query processor, a generic optimization framework for GIS and a middleware, which employs the new Query Melting paradigm (QM) that is based on the sharing paradigm and push-down optimization strategy. QM is implemented through a new Melting-Ruler strategy that works at the low-level, melts repetitions in plans to share spatial areas, temporal intervals, objects, intermediate results, maps, user locations, and functions, then re-orders them to get time-cost effective results, and is illustrated using a sample tourist GIS system.

## 1 INTRODUCTION

During the last decades, computer applications have been deployed to manage spatial data with Geographic Information Systems. GIS are computer-based tools mostly used to handle the geo-features of the real world. They provide the capability to input, store, manipulate, analyze, retrieve, transform, and display geographical data related to earth surface and its events as well as to measure aspects of geographic phenomena and processes. With the emergence of mobile technologies, the use of GIS Geo-data by mobile devices becomes very common. Mobile GIS is the combination of systems which include mobile devices, Global Positioning Systems (GPS), wireless communication, and GIS software. In tracking systems, they provide the facility to track employees and children in the backyards (Ladd, 2005; Kim, 2007). In road networks, they provide the ability to organize public transportation (Repenning, 2006), query moving objects (Guting, 2006), guide tourists (Beeharee, 2007), and exploit

photos (Beeharee, 2006). In data mining, they play a major role in trajectory pattern recognition of moving users (Gianotti, 2007; Andrienko, 2007). Mobile GIS are typically used in tourist and navigation systems for Proximity Analysis which includes querying the  $k$  Nearest Neighbor ( $k$ NN) and finding the facilities within a buffer area. The existing Mobile GIS applications have textual or menu-driven environment, do not provide a user friendly environment, and in some applications they are aimed at expert users. Moreover they allow the user to formulate one query at a time only. Thus, a new iconic visual query language (IVQL) has been developed in (Elsidani Elariss, 2006a, 2006b) to provide the mobile user with the facility to formulate a visual query using expressive icons and send complex queries, therefore reducing the overheads significantly.

Query Optimization includes a list of tasks that are executed in a particular order for processing a simple query, which form an execution plan. Having a complex query made up of a number of simple

queries requires the Query Processor to execute a number of corresponding plans, combine their result in one map, and send the result map to the mobile user. If one or more of the simple queries are dynamic, the processor repeats the same steps for every new time instance that corresponds to the current user location. These repetitions raise the need to develop a Query Processor for Dynamic Complex Queries which aims to eliminate all repetitions, share intermediate results, facilities, spatial areas, time intervals, and generate one Global Execution Plan for each dynamic complex query.

## 2 RELATED WORK

To deal with the different types of queries, different query optimization strategies have been considered. In this section we briefly describe some of them showing some of their advantages and limitations.

### 2.1 The Sharing Paradigm

The existing query optimization strategies that use the sharing paradigm are concerned with processing multiple queries in data analysis applications. An example is the Virtual Microscope Processor (Andrade, 2001, 2002c, 2002d, 2006; Afework, 1998). The runtime system was designed for shared-memory multi-processors. It tackles queries that are user-defined, allows the input data to be shared, and the query results to be reused by other queries. It aimed at optimizing query processing by (1) maintaining intermediate data structures generated by queries for intermediate results, (2) caching input data in memory, and (3) providing support for multi-threaded execution where each query is executed as a thread. Caching methods are used to store query results in the memory in the aim to speed up the execution of queries (Andrade, 2002a, 2002b, 2002e, 2003). These results along with input data can be used by other queries to produce new results. When caching is implemented at the server side, multiple clients can share the query results.

The experiment was conducted on an 8-processor Symmetric Multiprocessing machine, running Linux Kernel version 2.4.3. Two scenarios were considered in order to examine the performance of the runtime system and for each scenario two executions were done; each considered as a case making a total of 4 cases. In the first scenario, 16 clients were emulated. The calculated value of the overlap index was large 70% reflecting a high overlap among queries. In the second scenario, 8 clients were emulated making a

relatively small overlap index of 59%. Two executions were done for each scenario. In the first one, the Data Store Manager was ON, hence maintained intermediate results. In the second one, it was OFF, hence did not maintain intermediate results. The execution time in seconds of each case was recorded. An evaluation of the performance was conducted where results showed that a better performance was obtained when maintaining intermediate results. In the high overlap index case, the execution time decreased by about 30% to 40%. In the low one, it decreased by about 18%. Results also reported that the query execution time decreased as the number of threads increased and that the query evaluation time decreased as the size of the data store manager cache increased. The execution time decreased by 38%.

### 2.2 The Push-down Strategy

The existing query optimization strategies that use the push-down approach are concerned with processing execution plans where the order of execution of the operators affects the execution time without affecting the output result of the query such as having both the Selection and the Join operators in the same execution plan. In some cases and depending on the data records, the execution of one the Selection operator before the Join operator produces faster results than the execution of the Join operator before the Selection operator, and vice versa. The push-down strategy is based on swapping operators in an execution plan in order to get faster results. An example of the push-down paradigm is described in (Elmongui, 2005, 2006) which deals with the optimization of multiple predicate spatio-temporal queries for applications such as to find in which region of a continuously monitored city the number of suspects is greater than the number of police officers. The proposed idea is building a uniform adaptive query optimization framework that includes: selectivity estimation, cost estimation, adaptive query optimization model, and an extension of query optimization to cover multiple multi-predicate spatio-temporal queries. The Spatio-Temporal Histogram (ST-Histogram) was used to estimate the selectivity of a continuous spatio-temporal query operator that is sent from the query executor to the histogram manager periodically in form of statistics. An extension was proposed in order to accommodate multiple feedbacks especially the spatial relationship between queries. A data-to-plan grid was proposed as a framework to prevent executing the same query plan on all location

updates. The data-to-plan grid directs each location update to a relevant plan. A dynamic plan formation mechanism is used to provide the facility to add, remove, or reshuffle operators in the evaluation plan. Also, an extension of query optimization to handle multiple multi-predicate spatio-temporal queries was proposed based on sharing sub-plans between multiple multi-predicate queries.

### 2.3 Both Sharing and Push-down

The existing query optimization strategies that use the sharing paradigm and the push-down strategy are concerned with processing scalable incremental spatio-temporal queries. An example is in (Mokbel, 2003, 2004b, 2004c, 2005a, 2005b) which deals with location-aware services. The proposed idea is sharing the underlying space, operator, and objects. The first type is sharing the underlying space by the queries that find whether one or more objects, such as cars, are located inside a spatial area. All the queries have the same object but different areas. A plan was illustrated as a decision tree cost model that reflects the expected number of comparisons, calculated as the product of the number of comparisons and the probability of finding an object in the region (area). Sharing the query operator is the second type of sharing. The queries continuously inform the user about a group of objects that are located in different areas. All the queries share the same table, but they search various regions. In order to optimize the query execution, a new Shared Global Plan was implemented and evaluated. Sharing the objects of the same interest is the third type of sharing. The queries continuously inform the user about the number of objects that are located in an area. Another new Shared Global Plan was implemented and evaluated and the push-down approach was applied by pushing the Selection operator pushed below the Join operator. Three different joining policies introduced in (Xiong, 2004) namely Clock-triggered Join Policy (CJP), Incremental Join Policy (IJP), and Hot Join Policy (HJP), were used in (Mokbel, 2004a). In the CJP, the spatial Join is reevaluated every T seconds. The IJP does not execute the spatial Join for the objects and queries that did not change their location since the last T interval of time. The HJP evaluates hot objects, if their movement affects the result, at each evaluation time. The results of the evaluation of the policies showed that for “moving queries on stationary objects” and “stationary queries on moving objects”, the CJP policy had a constant number of I/O regardless of the percentage of

moving queries. The IJP policy had a number of I/O that was significantly smaller than CJP. The same applied to the HJP policy when compared to IJP. The IJP policy had a much lower CPU cost than CJP and the HJP policy had a lower CPU cost than IJP.

### 2.4 Comparison of Strategies

The above described query optimization strategies have demonstrated a considerable improvement in the field of query optimization. However, the Sharing Paradigm approaches the multiple single predicate queries but not multiple queries with multiple predicates. It discusses sharing of the underlying space and object of interest for the type of queries that deal with finding if an object is located inside an area but does not consider the queries that deal with proximity analysis and it alters the cache memory size. The push-down strategy approaches re-ordering the operators of a plan based on the Selectivity and executes the updates in queries instead of re-executing the whole queries several times. However it alters the database schema which is not feasible in real-life applications. In order to address the limitations of the existing strategies a new query optimization framework is needed for multiple dynamic complex queries which does not change the cache memory size or the database schema. The proposed query optimization framework employs the new Query Melting paradigm (QM) that includes common sub-expression elimination, sharing objects of interest, spatial areas, time intervals, underlying space, and intermediate results. QM is implemented by using the new sliding ruler strategy.

## 3 QUERY MELTING

### 3.1 Commonality in GIS

A thorough examination of different GIS and Location Based Services (LBS) applications, where users ask questions related to their position whether they are moving (dynamic) or not (static), shows that they have in common some functionalities, operations, and objects in execution plans. A predicate is the operator used by the user while formulating a query such as *Find Within a buffer* and *Find the k Nearest Facilities*. There are two types of operators namely the static and the dynamic. Each operator is decomposed as per its own query evaluation plan which is made up of a list of functions / operations that are to be executed in the

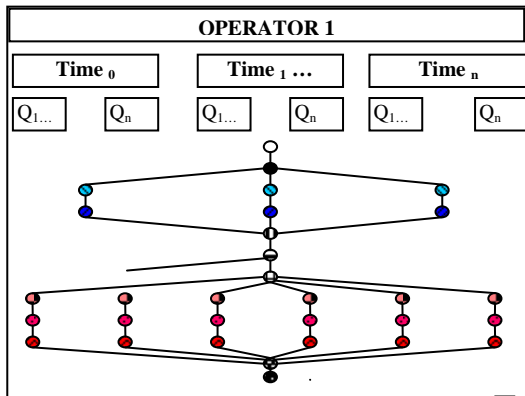


Figure 1: The Global Evaluation Plan of Operator1.

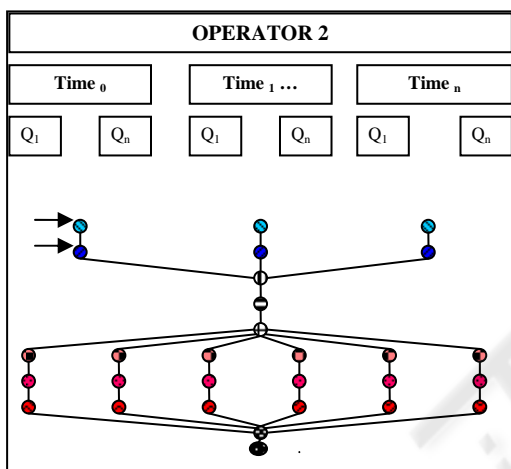


Figure 2: The Global Evaluation Plan of Operator2.

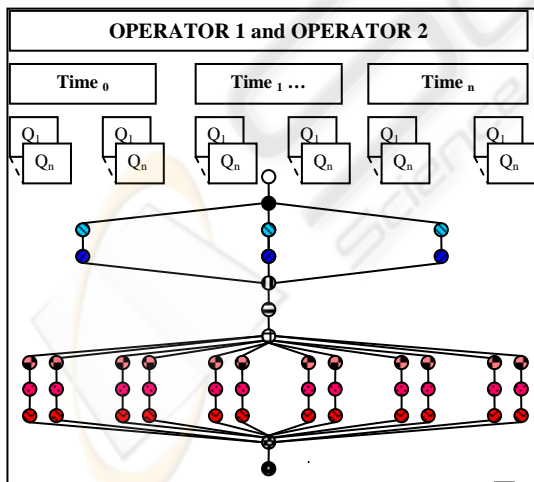


Figure 3: The Global Evaluation Plan of Both.

proper sequence. When multiple dynamic complex queries are formulated with multiple operators the global evaluation plan of each operator might

include functions that can be reused by other global evaluation plans. Figures, 1 and 2 show the two plans of the two different operators Operator1 and Operator2. The dots marked with arrows in Figure 2 are the same as in Operator1 plan so they can be melted. The other dots of Operator2 are appended to those of Operator1's because they can not be melted as shown in Figure 3. In this global execution plan there are 3 categories of operators: (a) Those that are executed once at the beginning only, (b) those that are executed at every new time instance, and (c) those that are executed for each query.

### 3.2 Query Melting Paradigm

The Query Melting Paradigm (QMP) is a generic optimized framework for GIS. It aims at building the optimal global evaluation plan for GIS spatio-temporal dynamic complex queries. Its main objective is to produce the most cost effective query processing in terms of execution time and memory storage, thus, minimizing the queries execution time cost. It is based on a combination of the Sharing paradigm, Query Optimization, and Push-down strategy. The Query Optimization that was applied by (Kang, 1994; Andrade, 2001; Elmongui, 2006; Mokbel, 2005a) is here extended to include “multi-user spatio-temporal multi-predicate dynamic complex queries”.

## 4 QUERY MELTING PROCESSOR

Query Melting is implemented using the Query Melting Processor QMP which is a middleware software system located on the server. Its major function is to input user queries, optimize them based on the Query Melting Paradigm, generate global execution plan and execute it, produce the resulting maps, and send the output to the user.

### 4.1 QMP Components

The Query Melting Processor consists of a number of components as shown in Figure 4. The architecture shows the three major steps that dynamic complex queries have to pass through in order to produce a query global evaluation plan. The Preprocessor takes as an input the dynamic complex query, parses it into multiple simple queries, groups them by category, and sorts them. Each simple query is decomposed according to its operator template.



Two-dimensional arrays are used to store the decompositions. The idea of using a Sliding ruler is applied here to melt the repetitions that exist in multiple plans. First, the Query Melting Ruler 1 melts the templates functions that are shared among multiple simple queries in the aim to implement common sub-expression elimination, sharing sub-plans, and sharing the underlying space (map). The Query Melting Ruler 1 works on a two-dimensional plane, and its output is the Initial Evaluation Plan for *Time 0* of the whole dynamic complex query. Second, the Query Melting Ruler 2 is responsible for implementing sharing the space and areas, sharing the time intervals, and sharing the object of interest. Sharing the space and areas occurs when multiple queries share the same spatial area or when an area is included in another. The Query Melting Ruler 2 draws the buffer of an area only once and uses it multiple times to clip or find the objects of other multiple simple queries. It allows sharing an interval of time if it is included or equal to the interval of other multiple queries. Finally, sharing the object of interest allows sharing the object of interest between multiple simple queries by reading the object table once and using it for many queries. The Query Melting Ruler 2 works on the same two-dimensional plane as the Query Melting Ruler 1, and its output is the Final Global Evaluation Plan for *Times 1...n*.

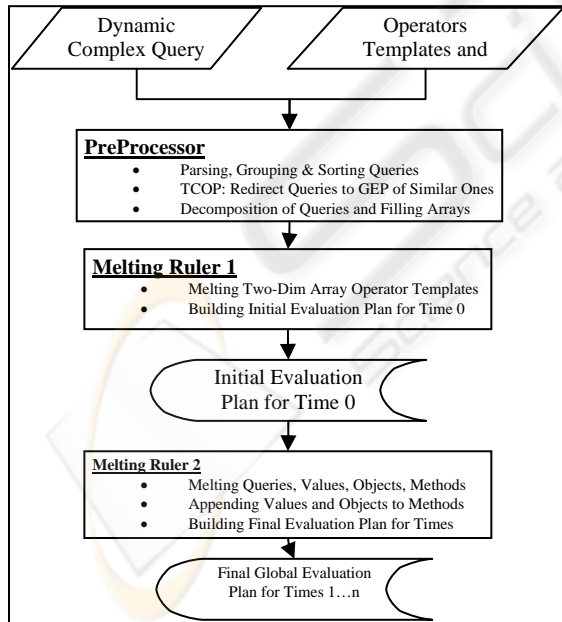


Figure 4: The Components of the QMP.






## 4.2 System Mechanism


Before describing the mechanisms of the execution plans, in the next few sections, we are describing some abbreviation for the representation of the icons of the IVQL which represent basically operators, values, and objects. Also we are showing some examples of simple as well as complex visual queries. The smiley icons that represent operators are listed in Table 1 along with their abbreviation and explanation. The first six operators are used for static queries, i.e. they ask about objects at the current *Time 0* only. The next six operators are used for dynamic queries, i.e. they launch a continuous query that lives *n* minutes. Every *m* minutes, they update the user with the new results of the queries based on his new location that is received through the GPS and satellite systems. Table 2 shows some of the icons that represent objects and facilities such as restaurant, hospital, and gymnasium.

Table 1: The Icons Used as Operators to Formulate Dynamic Complex Queries.

|    | Icon | Abbreviation | Explanation  |
|----|------|--------------|--|
| 1  |      | SW           | Find the facilities that are within a certain buffer   |
| 2  |      | SN           | Find the k nearest facilities  |
| 3  |      | SP           | Find the shortest path   |
| 4  |      | SNP          | Find the k nearest facilities and their shortest paths   |
| 5  |      | STL          | Find the time left to reach the k nearest facilities   |
| 6  |      | SDL          | Find the distance left to reach the k nearest facilities   |
| 7  |      | DW           | Continuously find the facilities that are within a certain buffer  |
| 8  |      | DN           | Continuously find the k nearest facilities   |
| 9  |      | DP           | Continuously find the shortest path on my way  |
| 10 |      | DNP          | Continuously find the k nearest facilities and paths   |
| 11 |      | DTL          | For the next n minutes, update me every m minutes with the time left to reach the k nearest facilities     |
| 12 |      | DDL          | For the next n minutes, update me every m minutes with the distance left to reach the k nearest facilities |

Table 2: The Icons Used as Objects.

|   | Icon  | Abbreviation | Explanation  |
|---|---|--------------|--------------|
| 1 |  | R            | Restaurant   |
| 2 |  | M            | Motel        |
| 3 |  | H            | Hospital     |
| 4 |  | U            | Tube / Metro |
| 5 |  | G            | Gymnasium    |

The AND operator  is used to combine multiple simple queries into a complex one. Some examples of simple queries are:

- [SN 1 M]: find nearest 1 motel
- [SNP 5 M]: find 5 nearest motels with their paths
- [SW 500 R]: find all restaurants that are within 500 meters from my location
- [SW 1000 R]: find all restaurants that are within 1000 meters from my location
- [DTL 60 10G]: continuously, find the Time Left to reach the nearest gymnasium for the next 60 minutes. Keep on supplying me with an updated result every 10 minutes. The result includes the shortest path.

An example of a dynamic complex query is shown in Table 3 summarizing a number of queries: Find the nearest theatre and its shortest path. While I am on my way, keep on supplying me with the 3 nearest motels and their shortest paths that are within 200 meters until I reach the theatre or 60 minutes overlap. Update my map every 5 minutes.

Table 3: A Dynamic Complex Query with 3 Predicates.

|              |             |            |
|--------------|-------------|------------|
| [DTL 60 5 T] | [ DNP 3 M ] | [DW 200 M] |
|--------------|-------------|------------|

Each operator is decomposed into a set of steps according to the operator’s template. The templates of some of the static operators are shown in Table 4 where each static operator is performed once only for *Time 0*. The steps of each operator are executed in the order in which they appear in the template following the top-down direction. The MakeLayer function creates a new closest facility layer that is used to find the nearest objects. The ReadXY function reads the XY location of the user. The AddXYIncident function adds the user’s location as

an incident to the layer. The Path is set to YES so as to produce the shortest path and set to No if no path is required. The NFacilities is the number of facilities needed to look for. The Impedance is set to Meters, Pedestrian-Time, or Drive-Time. The AddFacilities function specifies the object database table such as restaurants, motels, etc. The Solve function finds the nearest facilities based on the previous parameters. The AddToMap and SendMap functions add the results to the map and send it to the user. The DrawBuffer draws a circle around the XY location of the user and the ClipBuffFac finds the facilities that are located in this buffer. The templates of the dynamic operators are shown in Table 5. Each of the dynamic operators has two templates. The first is applied for *Time 0* when the query is launched and the second is applied for each of the consequent time instances *Times 1...n*. The NewWatcher waits for new XY Locations to arrive then launches triggers to read the data and continue execution accordingly. The NewTimer launches the query life time which triggers a timer that lasts as long as required by the user. The KillQueryIfX is used to terminate the query if it is expired, the user reaches his destination, the user issues a cancellation order, or he gets disconnected. The rest of the functions operate the same as the static operators.

The Query Melting process is performed by the query melting rulers. If an element of the template ends with a star \*, the corresponding elements in the arrays are filled with objects, such as AddFacilities\* leads to AddRestaurant, AddHospital, etc., as shown in Step 9 of Table 6. If an element ends with two stars \*\*, the corresponding elements are increasingly numbered starting with 1, such as Solve\*\* leads to Solve1, Solve2, and so on. If an element ends with three stars \*\*\*, the corresponding elements are increased by 1 for every new time instance, such as ReadXY\*\*\* leads ReadXY1 for *Time 0* and ReadXY2 for *Times 1...n*. Using \*\* at the end of an element means that this particular element is repeated and executed for each simple query whether static or dynamic, whether for *Time 0* or *Times 1...n*. The Melting Ruler follows the top down direction. It eliminates all the repetitions of an element each row at a time. The result of the Melting Ruler is a list of functions that are to be executed, the Global Evaluation Plan.

Table 4: The Templates of Static Operators.

| STATIC OPERATORS TEMPLATES |               |               |               |               |            |
|----------------------------|---------------|---------------|---------------|---------------|------------|
|                            |               |               |               |               |            |
| Step 1                     |               |               |               |               |            |
| Step 2                     |               |               |               |               |            |
| Step 3                     | MakeLayer     | MakeLayer     | MakeLayer     | MakeLayer     |            |
| Step 4                     | ReadXY        | ReadXY        | ReadXY        | ReadXY        | ReadXY     |
| Step 5                     | AddXYIncident | AddXYIncident | AddXYIncident | AddXYIncident |            |
| Step 6                     | Path=Yes      | Path=Yes      | Path=Yes      | Path=No       | DrawBuffer |
| Step 7                     | NFacilities=1 | NFacilities=1 | NFacilities   | NFacilities   | ClipBufFac |
| Step 8                     | Impedance=T   | Impedance=M   | Impedance=M   | Impedance=M   |            |
| Step 9                     | AddFacilities | AddFacilities | AddFacilities | AddFacilities |            |
| Step 10                    | Solve         | Solve         | Solve         | Solve         |            |
| Step 11                    | AddToMap      | AddToMap      | AddToMap      | AddToMap      | AddToMap   |
| Step 12                    | SendMap       | SendMap       | SendMap       | SendMap       | SendMap    |
| Step 13                    |               |               |               |               |            |

Table 5: The Templates of Dynamic Operators.

| DYNAMIC OPERATORS TEMPLATES |               |               |               |               |               |               |               |               |               |               |
|-----------------------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
|                             |               |               |               |               |               |               |               |               |               |               |
|                             | Time 0        | Times 1...n   | Time 0        | Times 1...n   | Time 0        | Times 1...n   | Time 0        | Times 1...n   | Time 0        | Times 1...n   |
| Step 1                      | NewWatcher    |               | NewWatcher    |               | NewWatcher    |               | NewWatcher    |               | NewWatcher    |               |
| Step 2                      | NewTimer      |               | NewTimer      |               | NewTimer      |               | NewTimer      |               | NewTimer      |               |
| Step 3                      | MakeLayer     | MakeLayer     | MakeLayer     | MakeLayer     | MakeLayer     | MakeLayer     | MakeLayer     | MakeLayer     |               |               |
| Step 4                      | ReadXY        | ReadXY        | ReadXY        | ReadXY        | ReadXY        | ReadXY        | ReadXY        | ReadXY        | ReadXY        | ReadXY        |
| Step 5                      | AddXYIncident | AddXYIncident | AddXYIncident | AddXYIncident | AddXYIncident | AddXYIncident | AddXYIncident | AddXYIncident | AddXYIncident | AddXYIncident |
| Step 6                      | Path=Yes      | Path=Yes      | Path=Yes      | Path=Yes      | Path=Yes      | Path=Yes      | Path=No       | Path=No       | DrawBuffer    | DrawBuffer    |
| Step 7                      | NFacilities=1 | NFacilities=1 | NFacilities=1 | NFacilities=1 | NFacilities   | NFacilities   | NFacilities   | NFacilities   | ClipBufFac    | ClipBufFac    |
| Step 8                      | Impedance=T   | Impedance=T   | Impedance=M   | Impedance=M   | Impedance=M   | Impedance=M   | Impedance=M   | Impedance=M   |               |               |
| Step 9                      | AddFacilities | AddFacilities | AddFacilities | AddFacilities | AddFacilities | AddFacilities | AddFacilities | AddFacilities |               |               |
| Step 10                     | Solve         | Solve         | Solve         | Solve         | Solve         | Solve         | Solve         | Solve         |               |               |
| Step 11                     | AddToMap      | AddToMap      | AddToMap      | AddToMap      | AddToMap      | AddToMap      | AddToMap      | AddToMap      | AddToMap      | AddToMap      |
| Step 12                     | SendMap       | SendMap       | SendMap       | SendMap       | SendMap       | SendMap       | SendMap       | SendMap       | SendMap       | SendMap       |
| Step 13                     | KillQueryFX   |               | KillQueryFX   |               | KillQueryFX   |               | KillQueryFX   |               | KillQueryFX   |               |

Table 6: Multiple Queries with One Static Operator during Query Melting.

| MULTIPLE QUERIES WITH ONE STATIC OPERATOR BEFORE QUERY MELTING |                |               |               |               |               |
|--|----------------|---------------|---------------|---------------|---------------|
| STL  |                |               |               |               |               |
|  | TEMPLATE       | RESTAURANT    | HOSPITAL      | MOTEL         | GYM           |
| Step 1   |                |               |               |               |               |
| Step 2   |                |               |               |               |               |
| Step 3   | MakeLayer      | MakeLayer     | MakeLayer     | MakeLayer     | MakeLayer     |
| Step 4   | ReadXY         | ReadXY        | ReadXY        | ReadXY        | ReadXY        |
| Step 5   | AddXYIncident  | AddXYIncident | AddXYIncident | AddXYIncident | AddXYIncident |
| Step 6   | Path=Yes       | Path=Yes      | Path=Yes      | Path=Yes      | Path=Yes      |
| Step 7   | NFacilities=1  | NFacilities=1 | NFacilities=1 | NFacilities=1 | NFacilities=1 |
| Step 8   | Impedance=T    | Impedance=T   | Impedance=T   | Impedance=T   | Impedance=T   |
| Step 9   | AddFacilities* | AddRestaurant | AddHospital   | AddMotel      | AddGym        |
| Step 10  | Solve**        | Solve1        | Solve2        | Solve3        | Solve4        |
| Step 11  | AddToMap**     | AddToMap1     | AddToMap2     | AddToMap3     | AddToMap4     |
| Step 12  | SendMap        | SendMap       | SendMap       | SendMap       | SendMap       |
| Step 13  |                |               |               |               |               |

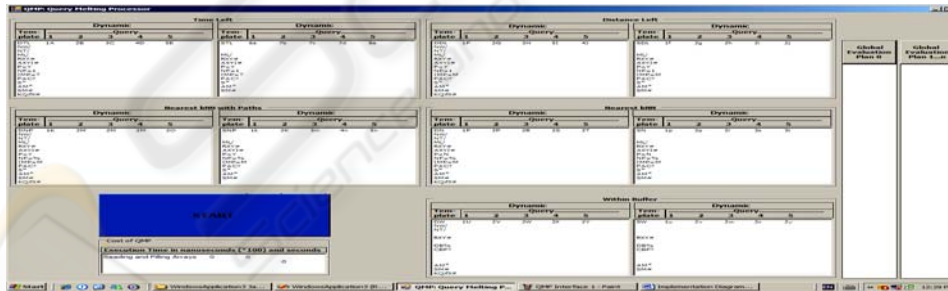


Figure 5: The User Interface before Melting.

### 4.3 Implementation

The QMP is implemented using the Microsoft Visual Basic.NET. The underlying experimental environment consists of a 1.8 GHz Intel Centrino Duo PC with 1 GB RAM running Microsoft Windows XP Home Edition. The User Interface of QMP is divided into sections as shown in Figures 5 and 6. Each Operator such as “Time Left” and “Distance Left” has its own section that is made up

of two boxes, the first for the dynamic queries and the second for static. Each box contains six columns, the first for the template and the next for queries. Each row corresponds to a function. The two long list boxes are the global evaluation plans. After processing a complex query, the execution time is displayed in nanoseconds in the box called Cost of QMP. During execution, the queries are displayed in their relative boxes.

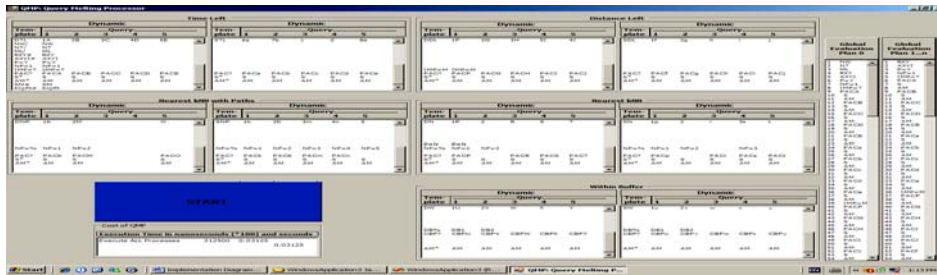


Figure 6: The User Interface after Melting.

## 5 EVALUATION

For the evaluation of the Query Melting Processor (QMP) the ‘time’ quality characteristic is quantified and cost is estimated using the Big-Oh notation which reflects how the cost of computation grows as a function the input size by considering the computation cost to be the running time in terms of the size, thus indicating an approximation to the number of steps taken by a process. Therefore, the computational cost of Melting the Templates is  $O(Ns \times (Nt-1))$  where  $Ns$  is the number of steps,  $Nt$  is the number of templates and the steps correspond to the row index numbers of each template. Since the computational cost of processing  $Ns$  steps is  $O(Ns)$  where for each step the rest  $Nt-1$  templates that are numbered from two to the last one are processed in reverse order and since the computational cost of processing  $Nt-1$  templates is  $O(Nt-1)$ , thus the computational cost of both these dimensions is  $O(Ns \times (Nt-1))$ . The computational cost of Melting the Values is  $O(Nq-1)$  where all the queries are processed starting from the first one until the last one, Melting the Objects is  $O(Nq-1)$  same as Values, Filling and Appending the Methods with Values and Objects is  $O(Ns \times Nq)$  where all the queries are processed for all the steps, and Generating the Global Execution Plans is  $O(Nq \times Ns)$  where all the multi-dimensional array elements are traversed. Hence, it can be concluded that the total time cost of the Query Melting Processor is the sum of the time costs of all its processes and is calculated as follows:

$$O(QMP) = O(Ns \times (Nt-1)) + O(Nt \times Nq) \times \sum_{n=1}^{qk-1} n + O(Nq-1) + O(Nq-1) + O(Ns \times Nq) + O(Ns \times Nq)$$

The second quality characteristic that is quantified in order to evaluate the efficiency of the Query Melting Processor is the memory space. This is done by evaluating the array that stores the

templates of the operators and the functions of the queries. Since the name of each function does not exceed 20 characters and each character occupies 2 Bytes in RAM (some systems 1 Byte) the name occupies maximum 40 Bytes. The total space occupied in the RAM is 40B/Function/Operator. In other words, it is equal to the product of the number of operators, the number of functions of each operator, and 40 Bytes, which is quantified as Space Cost = 40 Bytes  $\times$  Number of Functions  $\times$  Number of Operators. Hence, the space cost is considered minimal.

## 6 CONCLUSIONS

In this paper a query melting approach to processing dynamic complex queries was introduced. The approach aims at designing and implementing QMP, a query melting processor that is a generic optimization framework for GIS. A critical evaluation of the existing query optimization strategies proposed in the literature has been carried out in order to build the presented approach upon the current strategies. An investigation has been conducted to examine the commonalities that exist between the execution plans of different operators by studying the static operators alone, the dynamic ones alone, and a combination of both. The proposed QMP has been introduced, its components explained, and its mechanism described. The mechanism is based on the new query melting ruler strategy that is responsible for implementing the query melting paradigm through sharing spatial areas, time intervals, objects, functions, underlying space (map), user locations, and intermediate results. The proposed QMP implementation and user interface have been described using a tourist GIS system for proximity analysis. Finally, a theoretical evaluation has been carried out in order to quantify the time cost effectiveness of the approach.



## REFERENCES

- Afework A., Beynon M. D., Bustamante F., Demarzo A., Ferreira R., Miller R., Silberman M., Saltz J., Sussman A., and Tsang H., 1998. Digital Dynamic Telepathology – The Virtual Microscope, *In AMIA 98, American Medical Informatics Association, 1998*.
- Andrade H., Kurc T., Sussman A., and Saltz J., 2001. Efficient Execution of Multiple Query Workloads in Data Analysis Applications. *Proceedings of SC2001, Denver, USA*.
- Andrade H., Kurc T., Sussman A., Borovikov E., and Saltz J., 2002a. On Cache Replacement Policies for Servicing Mixed Data Intensive Query Workload. *Proceedings of the 2<sup>nd</sup> Workshop on Caching, Coherence, and Consistency, 2002*.
- Andrade H., Kurc T., Sussman A., and Saltz J., 2002b. Scheduling Multiple Data Visualization Query Workloads on a Shared Memory Machine, *Proceedings of the 2002 International Parallel and Distributed Processing, 2002*.
- Andrade H., Kurc T., Sussman A., and Saltz J., 2002c. Multiple Query Optimization for Data Analysis Applications on Clusters of SMPs, *In Proceedings of the 2<sup>nd</sup> International Symposium on Cluster Computing and the Grid, 2002*.
- Andrade H., Kurc T., Sussman A., and Saltz J., 2002d. Active Proxy-G: Optimizing the Query Execution Process in the Grid, *Proceedings of the 2002 ACM/IEEE Supercomputing Conference, 2002*.
- Andrade H., Kurc T., Sussman A., and Saltz J., 2002e. Processing Large-Scale Multi-dimensional Data in Parallel and Distributed Environments, *Parallel Computing*, 28(5), 827-859, 2002.
- Andrade H., Aryangat S., Kurc T., Slatz J., and Sussman A., 2003. Efficient Execution of Multi-Query Data Analysis Batches Using Compiler Optimization Strategies, *Proceedings of the 16<sup>th</sup> International Workshop on Compilers for Computing, LCPC, 2003*.
- Andrade H., Kurc T., Sussman A., and Beomseok N., 2006. Data Management and Query – Multiple Range Query Optimization with Distributed Cache Indexing, *SIGMOD Conference, 2006*.
- Andrienko G., Andrienko N., and Wrobel S., 2007. Visual Analytics tools for Analysis of Movement Data, *In Proceedings of ACM SIGKDD Explorations Newsletter, Vol. (9) 2, ACM*.
- Beeharee A., and Steed A., 2006. A Natural Wayfinding Exploiting Photos in Pedestrian Navigation Systems, *In Proceedings on Human-computer Interaction with Mobile Devices and Services, MobileHCI'06*.
- Beeharee A., and Steed A., 2007. Exploiting Real World Knowledge in Ubiquitous Applications, *Personal and Ubiquitous Computing*.
- Elmongui H., Mokbel M., and Aref W., 2005. Spatio-temporal Histograms, *Proceedings of SSTD, 2005*.
- Elmongui H., Ouzzani M., and Aref W., 2006. Challenges in Spatio-temporal Stream Query Optimization. *Proceedings of MobiDE 2006, Chicago*.
- Elsidani Elariss H., Khaddaj S., and Haraty R., 2006a. Towards a New Visual Query Language for GIS. *IATED Databases and Applications 2006, 195-202, Austria 2006*.
- Elsidani Elariss H., Khaddaj S., and Haraty R., 2006b. An Evaluation of a Visual Query Language for Information Systems. *ICEIS (5) 2006, 51-58, Paphos*.
- Gianotti F., Nanni M., Pinelli F., and Pedreschi D., 2007. Trajectory Pattern Mining, *In Proceedings of the 13<sup>th</sup> ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'07*.
- Guting H., De Almeida T., and Ding Z., 2006. Modeling and Querying Moving Objects in Networks, *VLDB Journal – The International Journal on Very Large Databases, Vol. (15) 2*.
- Kang M., Dietz H., and Bhargava B., 1994. Multiple-Query Optimization at Algorithm-Level. *Proceedings of SSDI 1994, Data Engineering 14, 57-75*.
- Kim S., Diverdi S., Chang J., Kang T., Iltis R., and Hollerer T., 2007. Implicit 3D Modeling and Tracking for Anywhere Augmentation, *VRST 2007, Newport Beach, California, November 5-7, 2007*.
- Ladd A., Bekris K., Rudys A., Kavradi L., and Wallach D., 2005. Robotics-Based Location Sensing Using Wireless Ethernet, *Proceedings of the 8<sup>th</sup> ACM International Conference on Mobile Computing and Networking, MOBICOM, Sep. 2002, Atlanta, GA*.
- Mokbel M.F., Aref W.G., Hambrush S.E., and Prabhakar S., 2003. Towards Scalable Location-Aware Services: Requirements and Research Issues. *In Proceedings of the ACM Symposium on Advances in Geographical Information Systems, ACM GIS*.
- Mokbel M.F., Xiong X., Aref W.G., Hambrush S.E., and Prabhakar S., Hammad M., 2004a. PLACE: A Query Processor for Handling Real-Time Spatio-temporal Data Streams. *In Proceedings of the VLDB*.
- Mokbel M.F., Xiong X., and Aref W.G., 2004b. SINA: Scalable Incremental Processing of Continuous Queries In Spatio-temporal Databases. *In Proceedings of the SIGMOD*.
- Mokbel M.F., 2004c. Continuous Query Processing in Spatio-temporal Databases. *In Proceedings of the ICDE/EDBT PhD Workshop*.
- Mokbel M.F., Xiong X., Hammad M., and Aref W.G., 2005a. Continuous Query Processing of Spatio-temporal Data Streams in PLACE. *GeoInformatica, 9(4), 343-365, 2005*.
- Mokbel M.F., and Aref W.G., 2005b. GPAC: Generic and Progressive Processing of Mobile Queries over Mobile Data. *In Proceedings of the MDM, Aya Napa, Cyprus*.
- Repenning A., and Ioannidou A., 2006. Mobility Agents: Guiding and Tracking Public Transportation Users, *Proceedings of the Working Conference on Advanced Visual Interfaces, AVI'06*.
- Xiong X., Mokbel M.F., Aref W.G., and Prabhakar S., 2004. Scalable Spatio-temporal Continuous Query Processing for Location Aware Services. *In Proceedings of the International Conference on Scientific and Statistical Database Management*.