Prioritized Evaluation of Continuous Moving Queries over Streaming Locations

Kostas Patroumpas and Timos Sellis

presented by Panagiotis Bouros

School of Electrical and Computer Engineering
National Technical University of Athens, Hellas

Managing Streaming Locations

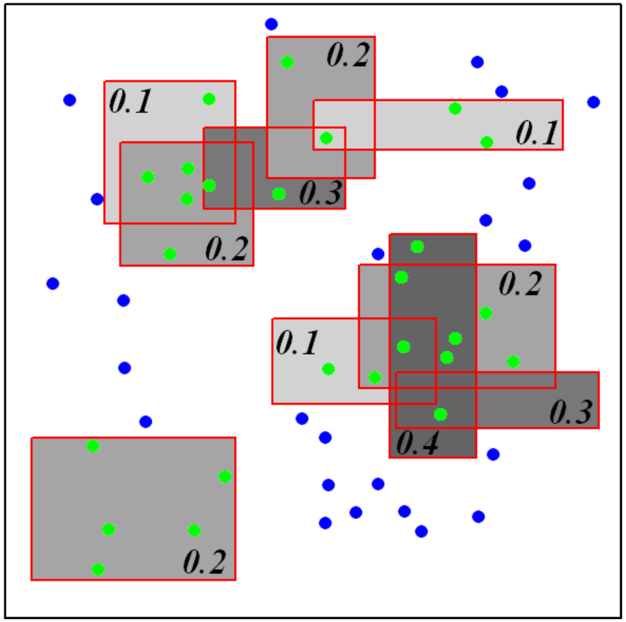
- Proliferation of location-enabled mobile devices
  - mobile phones, PDA's, GPS, …

  for tracking *moving objects*: people, vehicles, animals …
- Continuous user requests require real-time results
  - e.g., range or *k*-nearest neighbor search, skyline computation
  - against *data streams* of massive positional updates
  - Typically, all queries are considered of equal importance

➢ IDEA: introduce *priorities* to moving continuous queries
  - Model users' interest to receive response promptly or frequently
    - Message classification according to criticality
    - call for ambulance vs. search for restaurants or cinemas
  - Priorities may be also assigned by the central processor
    - Schedule execution of queries so as to better utilize system resources

Problem Specs

- Assuming a centralized processor that:
  - Receives timestamped locations from *N* moving objects
    - streaming tuples like $\langle oid, x_i, y_i, \tau \rangle$
  - Evaluates *M prioritized* moving range queries
    - also updated as tuples $\langle q_j, a_j, \rho_j, \tau \rangle$
    - specifying rectangles of interest a_j
    - with time-varying rank values $\rho_j \in [0..1]$
  - Runs periodically in *execution cycles*
    - Each cycle lasts for *T* time units
    - Refreshes location and query updates
    - Evaluates registered user requests against concurrent locations
    - *Synchronized data*: No out-of-order items for object or query streams



Our Approach

- Investigate priority-based evaluation strategies
  - for *range* queries with user-specified and time-varying ranks
- ➢ Distinguishing features
  - *Timeliness*
    - always provide fresh, but perhaps *approximate* responses
  - *Fairness*
    - treat queries according to their rankings
  - *Robustness*
    - handle scalable number of moving objects & queries
- ➢ Search for solutions that
  - share computation among queries
  - exploit common spatial predicates in query specifications

Related Work

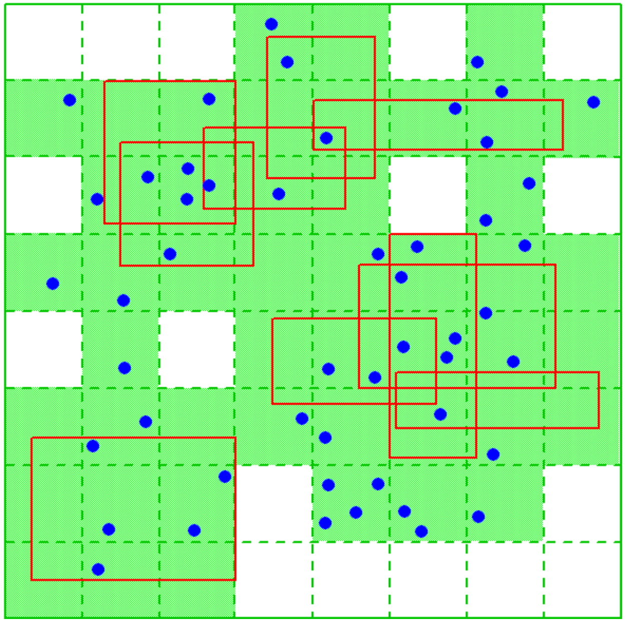
- Personalized queries (Koutrika & Ioannidis)
  - Query answering with respect to user profiles [ICDE'04]
  - Ranking model for selecting preferences [ICDE'05]
- Quality contracts (Qu & Labrinidis [ICDE'07])
  - Resource allocation between queries and data updates in websites
    - Model combining user preferences for both QoS and QoD
    - Specify time deadlines and freshness constraints
- Distributed processing on data streams
  - Prioritized transmission of local query results (Zhang *et al*. [ICDE'07])
  - LIRA: effective *region-aware* load shedding (Gedik *et al*. [ICDE'07])
- Continuous monitoring of moving objects
  - Range (Gedik & Liu [EDBT'04]) or *k*-NN (Mouratidis *et al.* [SIGMOD'05])
  - …*but* without ranking of requests or prioritized delivery of results

Outline

- Flexible spatial indexing
  - handle massive updates for moving objects and queries
- Ranking model
  - Rank aggregation schemes for multiple queries
  - Assignment of ranking scores to cells
  - Balance query rankings vs. object distribution
- Prioritized query evaluation
  - Execution in presence of changing ranking scores
  - Alternative examination strategies for cells
- Experimental Study
- Concluding Remarks

Spatial indexing

- Typical spatial access methods seem inadequate
  - Cannot easily cope with streaming positional updates at high rates
- Apply a regular *grid partitioning* of 2-*d* plane
  - Subdivide area of interest into $c \times c$ *square cells*
  - Common for locations and query ranges
- At each execution cycle
  - Hash current locations and rectangles against the grid
  - For each grid cell, update:
    - List of objects within cell
    - List of queries overlapping with cell
  - No history maintained for cells
    - *Queries always refer to current time*

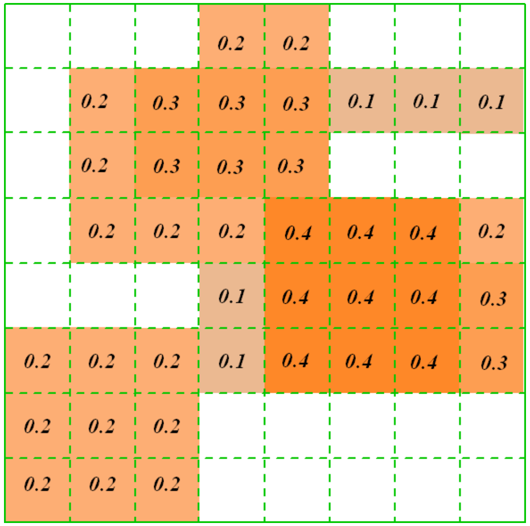


Ranking model

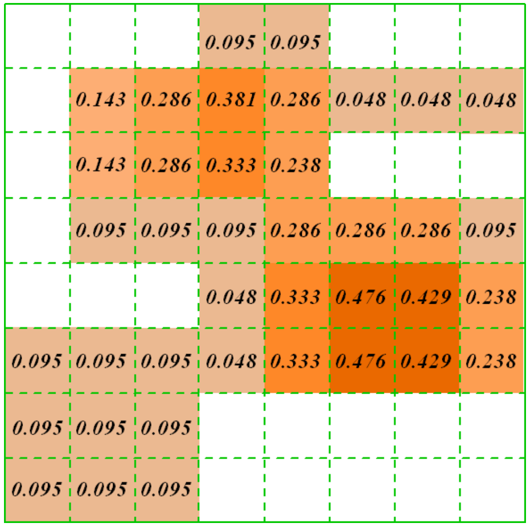
- Attempt to estimate *collective ranking* of multiple queries
  - Organize examination of queries in groups, not in isolation
    - Do not excessively penalize low-ranked queries
    - *… to the benefit of a few top-ranked ones!*
- Working *at cell level*
  - Each cell is assigned a *score* according to its query rankings
    - A mixture of query rankings is expected in each cell
    - Also take into account current distribution of object locations
  - Determine a visiting order for cells
  - Provide responses to queries affecting the cell under examination
- Introduce a family of representative *scoring functions*
  - to determine current collective rank *σ per cell*
    - queries are moving and their ranks may be fluctuating at each cycle
    - consider range extents that cover *or* partially overlap a given cell

Scoring Functions

- *Dominant* $\sigma(c_k) = \max_{q_i \in c_k}(\rho_i)$
  - Cell rank : the *highest* priority observed among its overlapping queries
  - Biased policy:
    - Give precedence to (a few) urgent requests against (many) non-critical ones

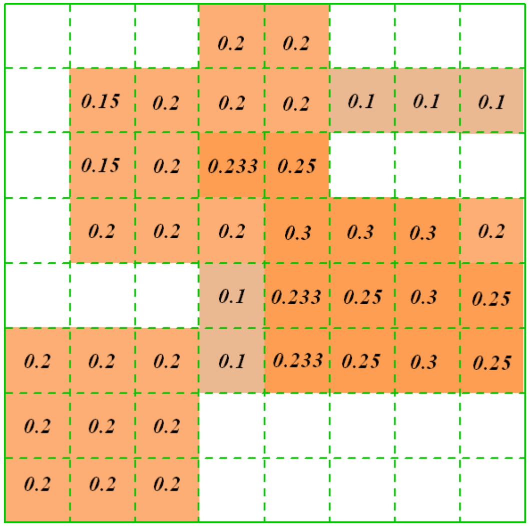


- *Normalized* $\sigma(c_k) = \frac{\sum_{q_i \in c_k} \rho_i}{\sum_{q_j \in Q} \rho_j}$
  - Cell rank : the relative importance of this cell over the cumulative ranking of all queries
  - Proportional scheme:
    - based on distribution of priorities across cells
    - attempts to examine cells with impartiality

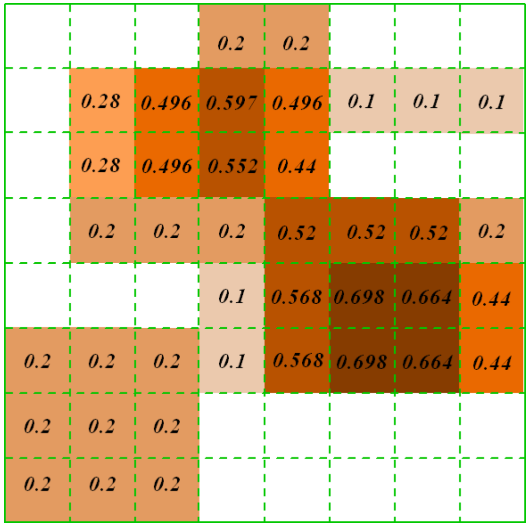


Scoring Functions (*cont'd*)

- ***Average*** $\sigma(c_k) = \frac{1}{m_k} \cdot \sum_{q_i \in c_k} \rho_i$
  - Cell rank : average priority of *local* queries
  - Egalitarian approach:
    - Smooth down the effect of extreme ranks
    - Presence of many low-ranked queries may cause delays to urgent requests



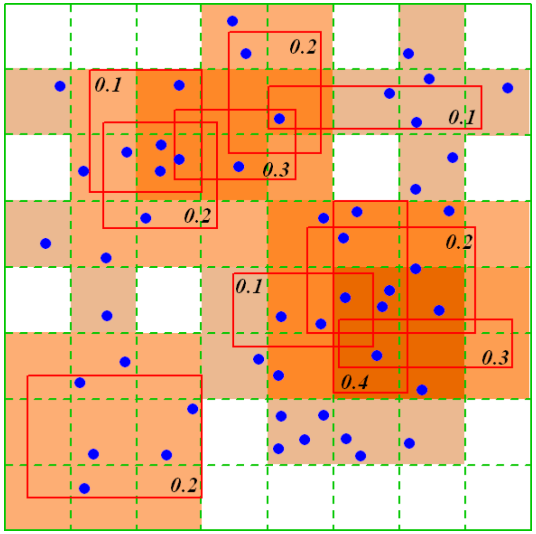
- ***Inflationary*** $\sigma(c_k) = 1 - \prod_{q_i \in c_k} (1 - \rho_i)$
  - Cell rank : extremely increasing when many high-ranked queries are found in this cell
  - Biased approach:
    - Queries of higher priority are given superior influence on cell scores
    - Favor query "clusters" of greater interest



Cell ranking scores

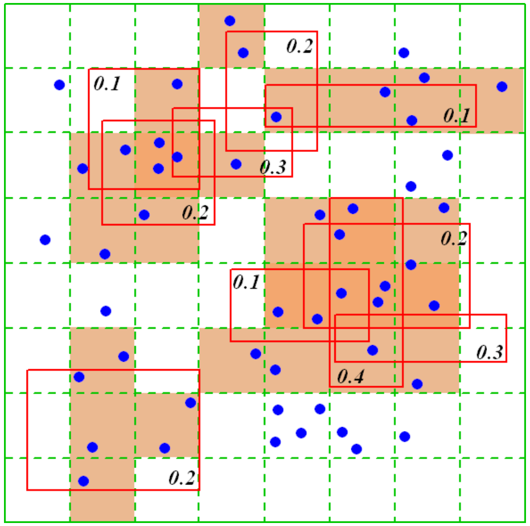
- Determine an *overall ranking score* β_k per cell c_k :
  - Collective query rank $\sigma(c_k)$
  - Present *object density* P_k in each cell
  - System-wide *regulation parameter* λ
    - to leverage between actual query rankings and object distribution
  - Propose alternative ranking schemes
    - *balanced, harmonic, combined*

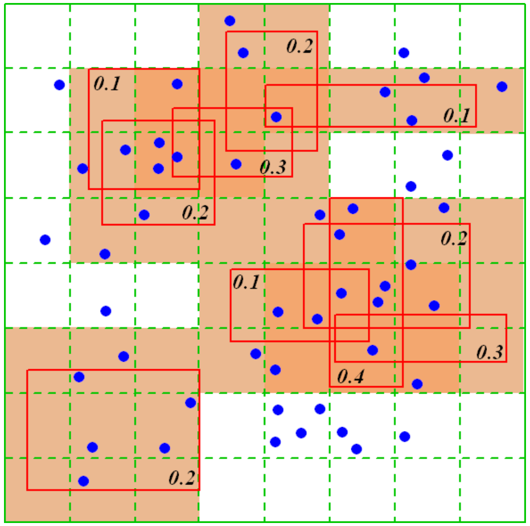
- *Balanced score* $\beta_k = \lambda \cdot \sigma(c_k) + (1 - \lambda) \cdot P_k$
  - Weigh importance of queries vs. objects
  - Possible settings for $\lambda \in [0..1]$:
    - $\lambda = 0.5$: equal weight
    - $\lambda = 1$: ignore object distribution
    - $\lambda = 0$: ignore query rankings



Cell ranking scores (*cont'd*)

- *Harmonic score* $\beta_k = \frac{(1+\lambda)\cdot\sigma(c_k)\cdot P_k}{\lambda\cdot\sigma(c_k)+P_k}$
  - Weighted harmonic mean per cell c_k
    - collective query rankings $\sigma(c_k)$
    - object distribution P_k
  - Possible settings for $\lambda \geq 0$:
    - $\lambda = 1$: equal importance
    - $\lambda < 1$: accentuate importance of queries
    - $\lambda > 1$: emphasize on object densities
- *Combined score* $\beta_k = \frac{m_k+n_k}{M+N} \cdot \sigma(c_k)$
  - Regularize collective ranking score by mixed density of m_k queries and n_k objects in cell
  - Take into account cell "*popularity*"
    - with pending queries and observed locations





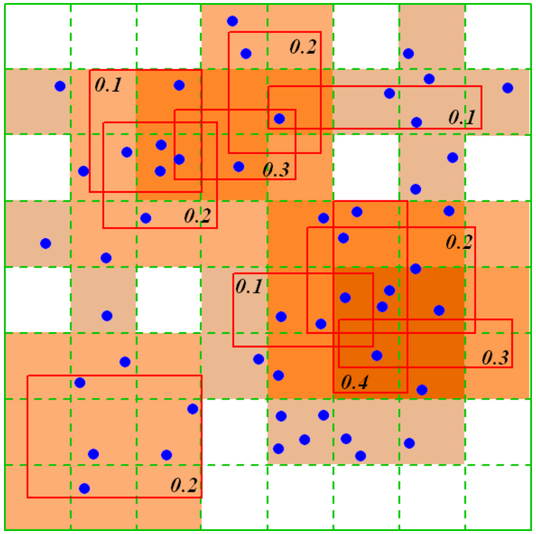
Prioritized Query Evaluation

- Evaluation in execution cycles
  - During each period *T* :
    1. Update data structures linked to grid cells
    2. Compute new cell rankings
    3. For the remaining interval, visit cells and generate query results
  - When deadline is reached, *start a new cycle*
    - Current state is dropped altogether
    - Some queries may receive *incomplete* or even *no response* at all !
  - Estimate *QoS* after each cycle:
    - *Global success ratio* $$\gamma(\tau) = \frac{\sum_{c_k} \sum_{q_i \in R} \rho_i}{\sum_{c_k} \sum_{q_j \in Q} \rho_j}$$
    - for all grid cells
    - *R*: queries that received some response during this cycle
    - *Q*: current query workload
    - If $\gamma(\tau) = 1$, then all queries received complete response.

Cell Examination Strategies

- Trivially, when visiting a cell:
  - queries should be probed in descending rank order
- Issues arising:
  - Examine all queries in a given cell, before going to next one ?
  - Respond first to all high-ranked queries in each cell ?
  - How to handle potential starvation of low-priority queries ?
- Three *alternative evaluation strategies*:
  - Exhaustive
  - Stratified
  - Threshold-guided
- Exhaustive Evaluation
  - Start from the cell with highest score and *probe all its queries*
  - Continue will other cells *in descending score order*
    - as long as deadline *T* is not reached

Stratified Evaluation

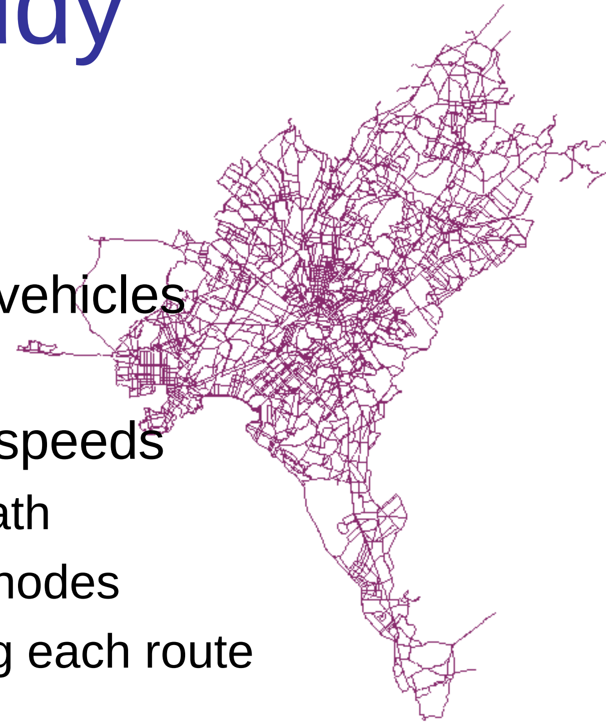


- Classify cells into *l strata*:
  - Like an *equi-sum histogram* or *quantile*
    - based on ranking scores
- *Rotating scheme* to prioritize queries:
  - Top-stratum cells examined at *each* cycle
  - Second-stratum cells get precedence every *two* cycles
  - Third-stratum cells every *four* cycles, etc.
  - Prevent possible *starvation or queries*…
- At each cycle, it favors a different stratum
  - Examine all other strata and their cells in descending score order
  - Cells with lower scores sometimes get prioritized, even not so often
  - Out-of-order prioritization depends on number *l* of strata

Threshold-guided Evaluation

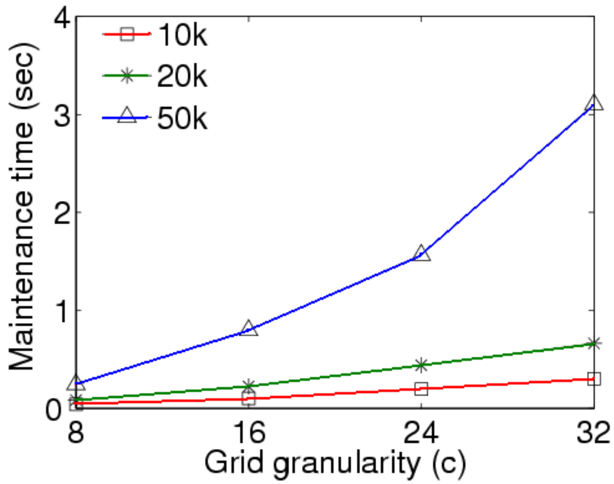
- IDEA: In each cell, examine high-ranked queries only
  - If all cells were visited within the deadline, and there is time left:
    - Start a new round, with adjusted cell scores
    - Provide response to more queries of lower ranks within current cycle
  - *BUT*, difficult to choose a suitable *threshold*
    - to discriminate such *élites* of high-ranked requests …
- Opt for a *dynamically adjusted threshold*:
  - Set global success ratio $\gamma(\tau - 1)$ of previous cycle as *target*
  - For currently examined cell c_k :
    - Compute a *local success ratio* $\gamma_k(\tau)$
    - Continue examination of this cell for as long as $\gamma_k(\tau) < \gamma(\tau - 1)$
  - To avoid degradation of system performance :
    - *Optimistically* raise the expected target by a small percentage
    - ➢ Attempt to improve global success ratio in successive cycles

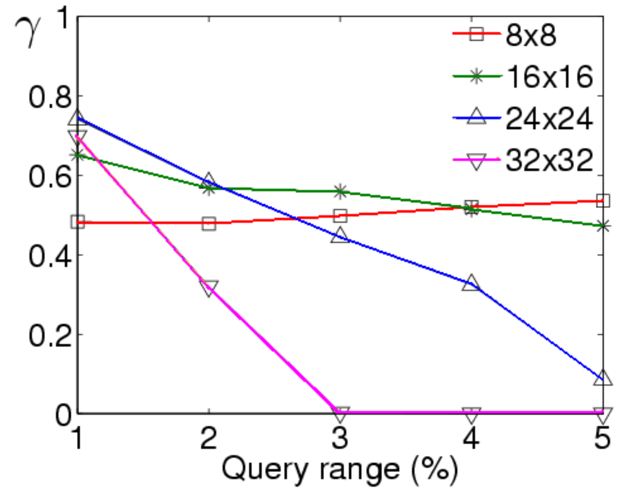
Experimental Study



- Experimental setting
  - Synthetic datasets emulating movement of vehicles
    - Along the road network of greater Athens
  - Objects/Query centroids moving at diverse speeds
    - Trajectories produced by running shortest path between pairs of randomly chosen network nodes
    - Samples of 200 timestamped locations along each route
  - Simulations
    - Number of objects *N* = 100 000
    - Query workload *M* = 10 000 , 20 000 , 50 000 of various ranges
    - Concurrent positional updates, *agility* = 100%
    - Query rankings assigned with a *Zipfian* distribution (*s*=1)
    - 10 rank values varying between 0.1 (low) and 1 (high preference)

Choosing grid granularity

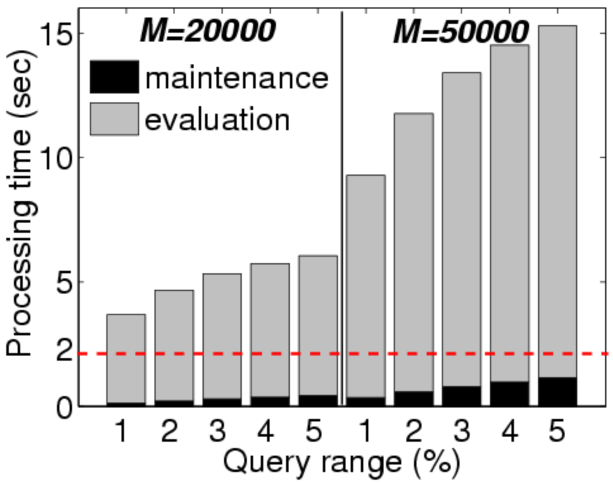


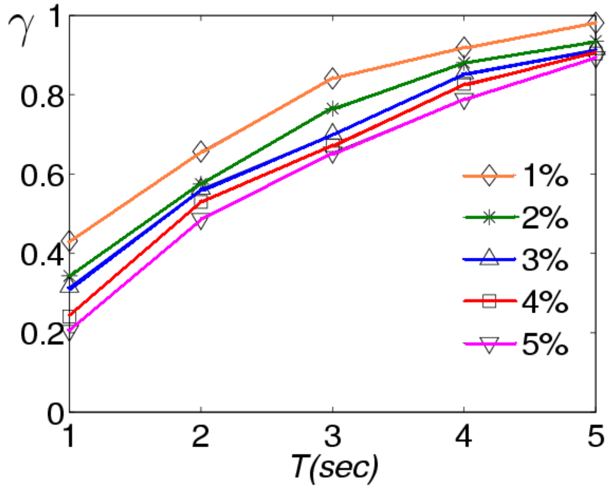


- Per cycle cost for hashing locations and queries
  - Maintenance time deteriorates with larger workloads
    - The finer the granularity, the more cells must be updated with scores
- Global success ratio γ under diverse grid granularities
  - Coarser subdivisions seem more stable for most query ranges

➢ Finally, *c*=16 was chosen for subsequent experiments

Impact of cycle duration

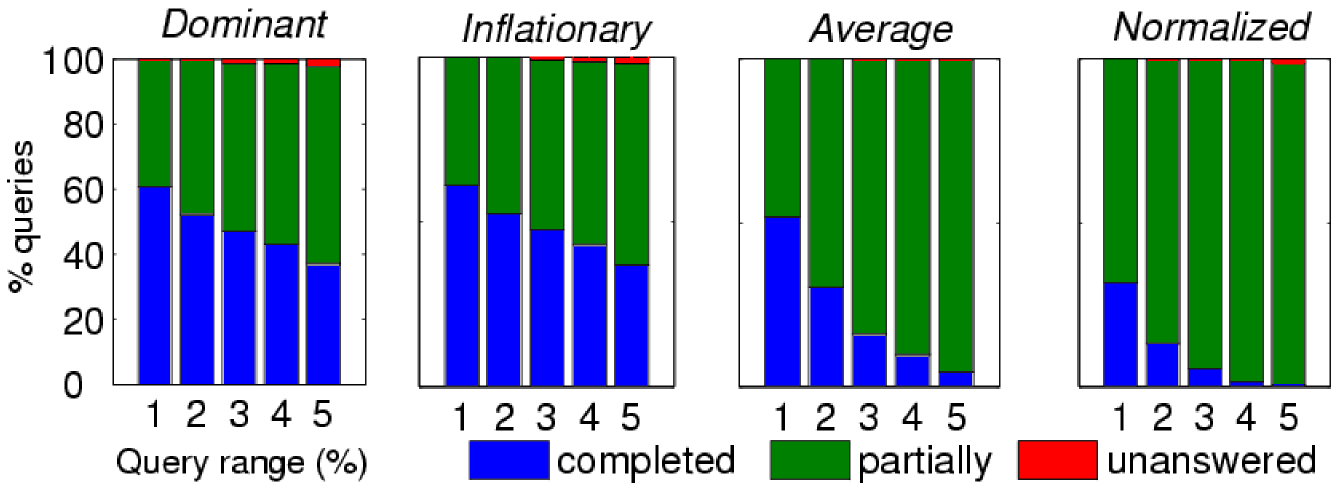




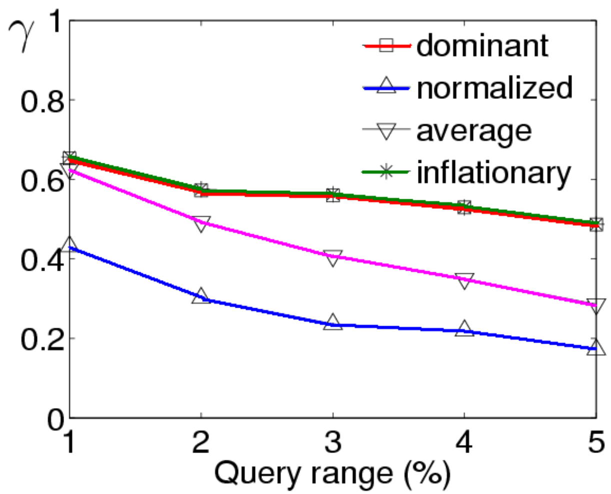
- Processing cost for *complete response* to all queries
  - Index maintenance and rank aggregation is minimal
  - Most time is spent on actual evaluation of queries
- Success ratio for *M* = 20000 queries with various deadlines
  - A strict deadline leads to drop in QoS => many unanswered queries
  - Best *T* is a trade-off between *M* and varying range extents

➢ Set *T=2 sec* for subsequent experiments, to compare performance

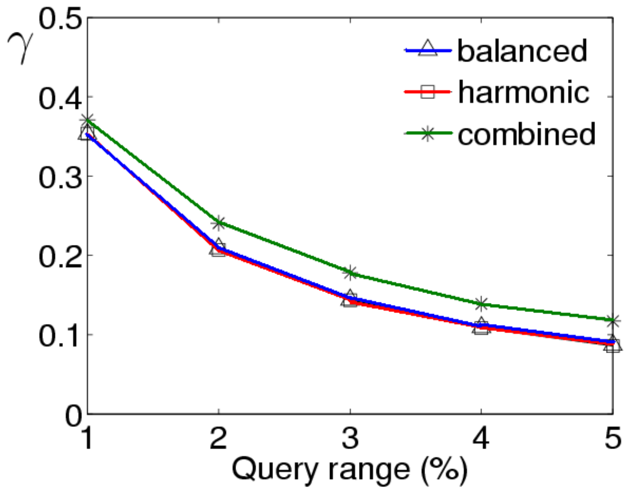
Comparing scoring functions

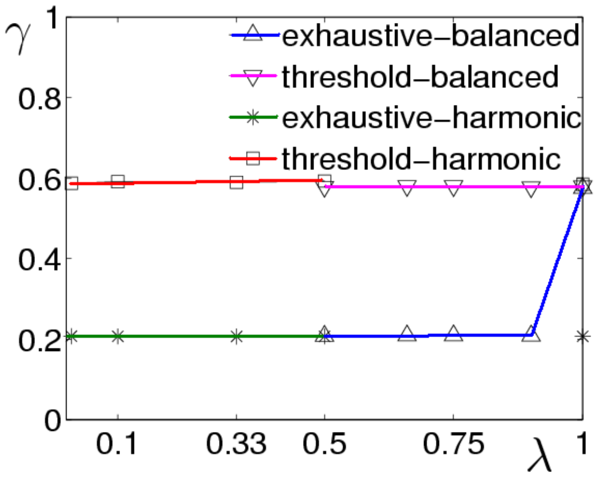


- Tests with $\lambda = 0$ (*i.e. ignoring objects*)
  - Better quality with biased schemes
    - Urgent requests boost rankings of certain cells, favoring affected queries
    - Queries left unanswered only rarely
  - *Inflationary* scheme prevails
    - with diverse ranges & query workloads



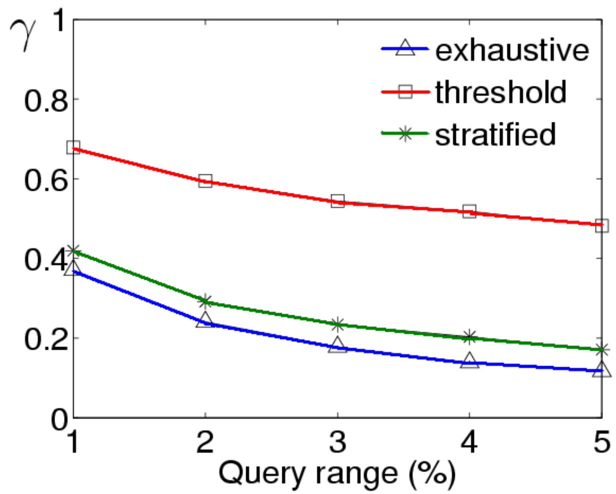
Alternative cell scoring schemes

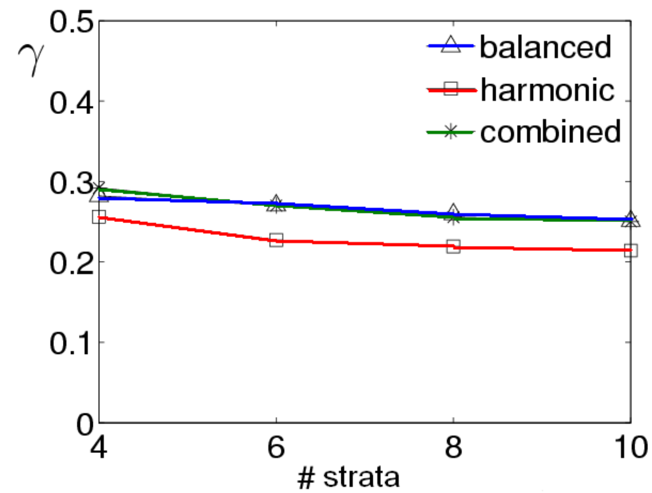




- Assessment of best method for assigning cell scores
  - Exhaustive test with significance of queries = 2 X significance of objects
- *Combined score* is superior
  - "Intensifies" aggregated ranks in proportion to current cell densities
- Results not accidentally biased from regulation effect
  - Balanced and harmonic schemes are practically immune to moderate λ

Alternative examination strategies





- *Threshold-guided execution* prevails
  - Self-regulating strategy which tends to examine more queries
  - At each cycle, this "best-effort" policy tries to gain an even better QoS
  - Stratified execution is slightly better than exhaustive
- Stratified evaluation
  - Under all scoring schemes, quality deteriorates with number of strata
  - More strata => less often each stratum gets prioritized

Conclusions

- Novel processing framework for moving range queries
  - Take into account user-specified priorities for query evaluation
  - Develop a ranking model to organize greater groups of queries
    - enable shared computation against scalable datasets
  - Propose adaptive policies with varying degrees of answering quality
    - most user requests receive in time at least partial results
- Possible future extensions:
  - Estimate accuracy in responses & confidence margins for queries
  - Adjust ranking model with estimated cost & query selectivity
  - Dynamic data-driven specification of regulation parameter
  - Develop an *aging-aware prioritization* mechanism
    - Artificially favor long-penalized queries to avoid starvation
  - Apply similar ranking schemes to other query types, like *k*-NN

Prioritized Evaluation of Continuous Moving Queries over Streaming Locations

Thank you!