Database Issues in Sensor Networks
Part I: Query Processing using Models

Jun Yang
CPS 296.4, Fall 2007
SAMSI Course on Sensor Networks

Some slide contents come from
C. Guestrin, A. Deshpande, and A. Silberstein
A note on course projects

http://www.cs.duke.edu/courses/fall07/cps296.4/assignments.html

- **Written project proposal**
  - 1 page detailing group composition and topic
  - Due Oct. 16

- **Brief progress presentation**
  - Approx. 5-10 min. for feedback
  - In Oct. 30 class

- **Report**
  - 10 page limit
  - Due Nov. 27 (last day of class)

- **Conference-type presentation**
  - 20-minutes + 5 min for questions in mini-conference
  - In Nov. 27 class
Analogy: sensor network as a DB

Query

TinyDB

Data aggregation:
- Can reduce communication

Declarative interface:
- Sensor nets are not just for PhDs
- Decrease deployment time

Every time step
Limitations?

- **Data representation/interpretation**
  - Plain SQL on raw data may give misleading results
    - Sensors readings do not represent “truth”
  - Plain SQL is awkward
    - No convenient way to handle missing data

- **Efficiency**
  - Every node must wake up at every time step
    - For SELECT * (collect all), every node transmits to the root
  - Doesn’t take advantage of data correlation in a principled way
Correlation in sensor data

- Natural consequence of continuous physical phenomena + dense network
- Correlation across time
  - History of readings → info about future readings
- Correlation across space
  - One sensor’s readings → info about others’ readings
- Correlation across modalities
  - One attribute (e.g., light) value → info about another attribute (e.g., temperature) value
Model-driven data acquisition

- Amol Deshpande, Carlos Guestrin, Samuel Madden, Joseph M. Hellerstein, and Wei Hong.
  “Model-Driven Data Acquisition in Sensor Networks.” International Conference on Very Large Data Bases, 2003
Under the hood

New Query

SQL-style query with desired confidence

Probabilistic Model

Data gathering plan

Condition on new observations

posterior belief

Δt

Data

Confidence gathering plan on new observations
Advantages

- Use of prior knowledge about correlations
- Observe fewer/cheaper attributes
  - Avoid not only transmission but also acquisition
    
  What’s the caveat?

- Solution to missing data
- Incorporation of new observations in knowledge
- Reuse of information among queries and over time
Working with probabilistic models

- Learn joint distribution $p(X_1, \ldots, X_n)$ from historical data
- Example query: know $X_2$ within $\pm \varepsilon$ with prob. at least $1-\delta$
  - Marginalize: $p(x_2) = \int p(x_1, x_2) \, dx_1$
  - Compute mean: $\mu_2 = \int x_2 \, p(x_2) \, dx_2$
  - Compute confidence: $P(X_2 \in [\mu_2-\varepsilon, \mu_2+\varepsilon])$
    - $\mu_2 - \varepsilon \int \mu_2 + \varepsilon \, p(x_2) \, dx_2$
    - If it’s at least $1-\delta$, return $\mu_2$
    - What if it’s not?
Working with probabilistic models (cont’d)

- Example query cont’d
  - Acquire the value of $X_1$, and exploit correlation to better estimate/bound $X_2$
  - Posterior distribution:
    \[
    p(x_2 \mid X_1 = 18) = \frac{p(18, x_2)}{p(X_1 = 18)}
    \]
  - Compute new mean and confidence based on this distribution
    - If new confidence is good enough, return new mean
    - If not, acquire more attributes and condition further on these observations (in the worst case, acquire $X_2$ itself)
Dynamic models

- Assume Markovian transition model: $p(x^t \mid x^{t-1})$, learned from historical data.

- Joint distribution at time $t - 1$: $p(x^{t-1} \mid o^1, \ldots, t-1)$

- Apply transitional model: $p(x^t \mid o^1, \ldots, t-1)$
  $$ = \int p(x^t \mid x^{t-1}) p(x^{t-1} \mid o^1, \ldots, t-1) \, dx^{t-1} $$
  
  Typically adds more uncertainty

- Make new observations $o^t$, and further condition $p(x^t \mid o^1, \ldots, t-1)$ on $o^t$ to get $p(x^t \mid o^1, \ldots, t)$
  
  Typically reduces uncertainty

- Repeat
Supported queries

- Value query: value of $X_i \pm \varepsilon$ with prob. at least $1 - \delta$
- Range query: value of $X_i \in [a, b]$ with prob. at least $1 - \delta$ or no more than $\delta$
  - Compute $\int_a^b p(x_i) \, dx_i$
- Aggregation query: average of all $n$ readings within $\pm \varepsilon$ with prob. at least $1 - \delta$
  - $p(Y = y) = \int p(x_1, \ldots, x_n) 1[(\sum_i x_i)/n = y] \, dx_1 \ldots \, dx_n$

Requires solutions to integrals
- In general requires numerical integration or sampling
- For “nice” distributions (e.g., Gaussian), sometimes can compute in closed-form
Query optimization

- Which readings shall we acquire?
- How do we collect them?
- Utility?
  - Query-driven, model-based: How much does it help us resolve remaining uncertainty?
- Cost?
  - Acquisition
  - Transmission
Network and query plan

- Assume quasi-static network topology

- Plan collects a subset $\mathcal{O}$ of sensor readings
  - Using a path that starts and ends at the root and visits all nodes in $\mathcal{O}$

Why not a tree?
Choosing a plan

- Example query: $X_i \in [a, b]$ with prob. $\geq 1 - \delta$
  - Benefit of observing $\mathcal{O} = \text{a specific } o$:
    $R_i(o) = \max[P(X_i \in [a, b] | o), 1 - P(X_i \in [a, b] | o)]$
  - But since we don’t know $o$, we settle for expected benefit:
    $R_i(\mathcal{O}) = \int p(o) R_i(o) \, do$

- Optimization problem
  - Minimize $\mathcal{O} \subseteq \{1, \ldots, n\}$ $\text{Cost}(\mathcal{O})$ such that $R(\mathcal{O}) \geq 1 - \delta$
  - Exhaustive search
  - Greedy heuristic: next to acquire is the reading with the highest benefit/cost ratio
    - Note that benefit changes as more readings are acquired
Experimental results

- Redwood tree and Intel lab
- Learned models from data
  - Learned a different transition model for each hour of the day (domain knowledge)
Cost vs. confidence
Approximate range queries

- Confidence set at 95%
Comparison with competitors

Where did approximate caching lose big?
Discussion

- How much do you trust your model?
  - What if it isn’t Gaussian after all?
  - Since we have assumed Gaussian in deciding what not to acquire, would the decision reinforce our (false) assumption?
  - What if the goal is to learn the model instead?
    - How would this change the utility of observations?

- How dynamic/adaptive is this approach?
  - Can the transition model itself be adapted on the fly?

- How do you measure utility in more complex situations?
  - MIN/MAX? Multiple queries?

- Outliers—can we really avoid acquisition?
- Expensive to optimize at the root for every epoch
- Is further compression worthwhile on the tour?
Alternative: data-driven philosophy

- Models can help, but they cannot substitute for actual readings
  - Particularly when we are still *learning* about the physical process being monitored
- Correctness of queries should not depend on correctness of models
- Models can still be used to optimize queries
Data-driven: push

- Exploit correlation in data + put smarts in network
  - Representatives: Ken [Chu et al., ICDE 2006], Conch [Silberstein et al., ICDE 2006, SIGMOD 2006]

\[
\text{Base station} \quad \text{Model } p(X^{(t)} | o^{(t-1)}, o^{(t-2)}, \ldots)
\]

\[
\text{Sensor network} \quad \text{Model } p(X^{(t)} | o^{(t-1)}, o^{(t-2)}, \ldots)
\]

\[
\begin{aligned}
\text{Compare actual reading } x^{(t)} \text{ with model prediction } \\
E(X^{(t)} | o^{(t-1)}, o^{(t-2)}, \ldots)
\end{aligned}
\]

\[
\begin{aligned}
\text{Transmit } o^{(t)} \text{ such that } \\
\|x^{(t)} - E(X^{(t)} | o^{(t)}, o^{(t-1)}, \ldots)\| \leq \varepsilon
\end{aligned}
\]

\[
\begin{aligned}
\text{Differ by more than } \varepsilon? \\
\end{aligned}
\]

Regardless of model quality, base station knows \(x^{(t)}\) to within \(\varepsilon\)

Better model \(\Rightarrow\) fewer transmissions
A simple example

(Transition) model: value predicted at \( t+1 \) ← value predicted at \( t \)

Source reporting rule:
report if \( |\text{actual} - \text{predicted}| \geq \varepsilon \)

Model update rule:
use actual if report received

Guarantee:
\[ |\text{actual} - \text{predicted}| < \varepsilon \]

Another possibility: A linear model

\[ X^{t+1} = a X^t + b \]
Ken


- Forget about fancy queries—many sensor network users still want to *collect all data*

- Use model to optimize transmission, but not to avoid acquisition
From temporal to spatiotemporal

Ken explores the spectrum between these two

What’s the trade-off?
Example: disjoint cliques

- Nodes form cliques with roots
- In each time step
  - Nodes in a clique transmit to clique root
  - Clique root runs model (mirrored at base station) and decides which readings (if any) to transmit to base station
Optimizing disjoint cliques

- **What to consider**
  - Intra-source: cost of reporting to clique root (for model maintenance)
  - Source-sink: cost of reporting to the base station
    - Depends on data reduction factor

- **How to compute data reduction factor**
  - Use the model (poor model gives suboptimal cliques)

- **Done compile-time, at base station**
  - NP-hard; use exhaustive or heuristic search
Example: average model

When would this model work well?
Experiment results

Ignores model maintenance (*why?*)
Discussion

- Main concern of BBQ addressed
  - No longer trust model completely
  - Can handle outliers
  - Though detecting node failures again depends on good models
- Robustness to message loss
  - Ambiguity between suppressed and lost transmissions
  - Assuming Markovian models and transmission of raw values, influence of lost messages is bounded in time
  - Is it easier to ensure that one big message gets through?
- Can you do better in intra-source transmission?
- Adapt model parameters?
- Model hierarchies? Overlapping models?
Conch = **Constraint Chaining**

- Also focuses on collecting *all data*
- Again, use model to optimize transmission, but not to avoid acquisition
- Individual models simpler than Ken, but they are “chained” together
Constraint chaining in action

- Temporally monitor spatial constraints (edges)
  - $x_i$ and $x_j$ change in similar ways $\Rightarrow$ temporally monitor $(x_i - x_j)$
  - One node is reporter and the other updater
    - Reporter tracks $(x_i - x_j)$ and transmits it to base station if its value changes
    - Updater transmits its value updates to reporter
      - I.e., temporally monitor remote input to the spatial constraint

- Base station “chains” monitored edges to recover readings
Chaining starting point; temporally monitored

Only “border” edges transmit to base station
Combines advantages of both temporal and spatial suppression

- Chains have nothing to do with communication paths
- Don’t worry about long chains—they are free
Choosing what to monitor

- A spanning forest is necessary and sufficient to recover all readings
  - Each edge is a temporally monitored spatial constraint
  - Each tree root is temporally monitored
    - Starting point for chaining
  - For better reliability, more edges can be monitored at extra cost

- Observations
  - Choose edges between correlated nodes
    - They transmit the fewest reports
  - Communication tree is not necessarily a good Conch tree
  - Do not connect erratic nodes
    - They can be monitored as singleton trees in the forest
Cost-based forest construction

- **Observe**
  - In pilot phase, use any spanning forest to collect data
    - Even a poor spanning forest correctly collects all data

- **Optimize**
  - Use collected data to assign monitoring costs
    - # of rounds in which monitored value changes
  - Build a min-cost spanning forest (e.g., Prim’s)

- **Re-optimize as needed**
  - When actual costs differ significantly from those used by optimization
Experimental results

- Environment
  - Simulation of Mica2 motes
  - Accounting of bytes sent and received

- Algorithms compared
  - Temporal suppression
  - Spatial suppression
  - Conch, with optimized spanning forest
  - Conch, based on communication tree

- Error tolerance through discretized values
  - \([k \varepsilon, k \varepsilon + \varepsilon) \rightarrow k\)
Wavefront experiment

- Periodic vertical wavefronts move across field, where sensors are randomly placed at grid points.

- Conch beats both pure temporal and pure spatial.

- Communication tree is a poor choice for monitoring; optimization makes a huge difference.
Conch discussion

- Key ideas in Conch
  - Temporally monitor spatial constraints
  - Monitor locally—with cheap two-node spatial models
  - Infer globally—through chaining
  - Push/suppress not only between nodes and base station, but also among nodes themselves
  - Observe and optimize

- Vision for ideal suppression
  - Number of reports \( \propto \) description complexity of phenomenon

What’s the catch?
Message failures in suppression

Failure and suppression

- Message failure common in sensor networks
  - Interference, obstacles, congestion, etc.

![](image)

- Is a non-report due to suppression or failure?
  - Without additional information/assumption, base station has to treat every non-report as plain “missing”—no accuracy bounds
A few previous approaches

- Avoid missing data: ACK/Retransmit
  - Often supported by the communication layer
  - Still provides no guaranteed delivery!

- Deal with missing data
  - Interpolation
    - Point estimates are often wrong or misleading
    - Uncertainty is lost—important in subsequent analysis/action
  - Use a model to predict missing data
    - Can provide distributions instead of point estimates
    - But we have to trust the model!
BayBase: basic Bayesian approach

- Model $p(X \mid \Theta)$ with parameters $\Theta$
  - Do not assume $\Theta$ is known
  - Any prior knowledge can be captured by $p(\Theta)$

- $x_{\text{obs}}$: data received by base station

- Calculate posterior $p(X_{\text{mis}}, \Theta \mid x_{\text{obs}})$
  - Joint distribution instead of point estimates
  - Quantifies uncertainty in model; model can be improved

Problem: non-reports are treated as generically missing
- But most of them are “engineered”!
- Non-report $\neq$ no information!

How do we incorporate knowledge of suppression scheme?
BaySail

Bayesian Analysis of Suppression and Failure

- Bayesian, data-driven
- Add back some redundancy
- Infer with redundancy and knowledge of suppression scheme
Redundancy strikes back

- **ACK/Retransmit**: added by the lower layer
  - Still does not help distinguish suppression from failure

- At app level, piggyback redundancy on each report
  - **Counter**: number of reports to base station thus far
    - Good systems idea!
  - **Timestamps**: last $r$ timesteps when node reported
    - Hmm… not that cute…
  - **Timestamps + Direction Bits**: in addition to the last $r$ reporting timesteps, a bit indicating whether each report is caused by
    - $(\text{actual} - \text{predicted} > \varepsilon)$ or
    - $(\text{predicted} - \text{actual} > \varepsilon)$
    - Why on earth?!
Redundancy design considerations

- Benefit: how much uncertainty it helps to remove
  - *Counter* can cover long periods, but helps very little in bounding particular values

- Energy cost
  - *Counter* < *Timestamps* < *Timestamps* + *Direction Bits*

- Complexity of in-network implementation
  - Coding app-level redundancy in TinyOS was much easier than finding the right parameters to tune for ACK/Retransmit! 😊

- Cost of out-of-network inference
  - May be significant even with powerful base stations!
Suppression-aware inference

- Redundancy + knowledge of suppression scheme ⇒ hard constraints on $X_{\text{mis}}$

- Temporal suppression with $\varepsilon = 0.3$, prediction = last reported
- Actual: $(x_1, x_2, x_3, x_4) = (2.5/\text{sent}, 3.5/\text{sent}, 3.7/\text{suppressed}, 2.7/\text{sent})$
- Base station receives: (2.5, nothing, nothing, 2.7)
- With *Timestamps* ($r=1$)
  - (2.5, failed & suppressed, 2.7)
  - $|x_2 - 2.5| > 0.3; |x_3 - x_2| \leq 0.3; |2.7 - x_2| > 0.3$
- With *Timestamps + Direction Bits* ($r=1$)
  - (2.5, failed & under-predicted & suppressed, 2.7 & over-predicted)
  - $x_2 - 2.5 > 0.3; -0.3 \leq x_3 - x_2 \leq 0.3; x_2 - 2.7 > 0.3$
- With *Counter*
  - One suppression and one failure in $x_2$ and $x_3$; not sure which
  - A very hairy constraint!

- Posterior: $p(X_{\text{mis}}, \Theta | x_{\text{obs}})$, with $X_{\text{mis}}$ subject to constraints
Using modeling/redundancy

<table>
<thead>
<tr>
<th>No knowledge of suppression</th>
<th>Just data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bayesian, model-based AR(1) with uncertain parameter</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Knowledge of suppression &amp; Timestamps</th>
<th>BayBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge of suppression &amp; Timestamps + Direction Bits</td>
<td>BaySail</td>
</tr>
</tbody>
</table>

\[ x_2 \notin [2.2, 3.0] \]

\[ x_3 \in \{x_2 - 0.3, x_2 + 0.3\} \]

\[ x_2 > 3.0 \]
Inference

- Arbitrary distributions & constraints: difficult in general
  - Monte Carlo methods generally needed
  - Various optimizations apply under different conditions

- A simplified soil moisture model: \( y_{s,t} = c_t + \phi y_{s,t-1} + e_{s,t} \)
  - \( c_t \) is a series of known precipitation amounts
  - \( \text{Cov}(Y_{s,t}, Y_{s,t'}) = \sigma^2 \frac{\phi^{|s-t'|}/(1 - \phi^2)}{\exp(-\tau ||s - s'||)} \)
  - \( \phi \in (0, 1) \) controls how fast moisture escapes soil
  - \( \tau \) controls the strength of the spatial correlation over distance

- Given \( y_\text{obs} \), find \( p(Y_\text{mis}, \phi, \sigma^2, \tau|y_\text{obs}) \) subject to constraints

- Gibbs sampling
  - Markovian \( \Rightarrow \) okay to sample each cluster of missing values in turn
  - Gaussian + linear constraints \( \Rightarrow \) efficient sampling methods (which do not generate/reject invalid samples)
Inference cost

\( \frac{\text{Running Time (s)}}{\text{Epsilon}} \)

**Major reason for adding the direction bits!**

- **Timestamps** translate to “\( \ldots \) > \( \varepsilon \)” constraints (disjunction); difficult to work with and led to lots of rejected samples.

- **Timestamps + Direction Bits** translate to a set of linear constraints;

  use [Rodriguez-Yam, Davis, Scharf 2004]

  and there are no rejections

  \( >100 \times \text{speed-up!} \)
Energy cost vs. inference quality

30% message failure rate
Roughly 60% suppression
Cost: bytes transmitted (including any message overhead)
Quality: size of 80% high-density region

Sampling is okay in terms of cost, but has trouble with accuracy

Suppression-aware inference with app-level redundancy is our best hope to get higher accuracy

ACK is not worth the trouble!
BaySail discussion

- Suppression vs. redundancy
  - Goal of suppression was to remove redundancy
  - Now we are adding redundancy back—why?
  - Without suppression, we have to rely on naturally occurring redundancy \( \leftrightarrow \) want to control where redundancy is needed, and how much

- Many interesting future directions
  - Dynamic, local adjustments to degree of redundancy
  - In-network resolution of suppression/failure
  - Failure modeling
  - Provenance: is publishing received/interpolated values enough?
Concluding remarks

All models are wrong, but some models are useful
— George Box

- Model-driven pull: BBQ
- Data-driven approach
  - Use model to optimize, not to substitute for real data
  - Quantify uncertainty in models
  - Use data to learn models

  - Ken: suppression based on spatiotemporal models
  - Conch: suppression based on chaining simple spatiotemporal models
  - BaySail: suppression-aware inference with app-level redundancy to cope with failure—suppression’s dirty little secret