Evaluating Proximity Relations Under Uncertainty

Zhengdao Xu and Hans-Arno Jacobsen
Department of Computer Science and
Department of Electrical and Computer Engineering,
University of Toronto
zhengdao@cs, jacobsen@eecg {.toronto.edu}

Abstract

For location-based services it is often essential to efficiently process proximity relations among mobile objects, such as to establish whether a group of friends or family members are within a given distance of each other. A severe limitation in accurately establishing such relations is the inaccuracy of dynamically obtained position data, the point in time, and the frequency with which the position data is collected. In this paper, we use the common model of interpreting the unknown position of an object by a probability distribution centered around the last know position of the object. While this approach is straight forward, it poses severe difficulties for establishing the truth or falsehood of the proximity relation. To address this problem, we analytically quantify the lower and upper bounds of the size of the smallest circle that covers the mobile objects involved in the proximity relation. Based on this result we propose two novel algorithms that closely monitor the relation at low location update cost. Furthermore, we develop a cost-effective estimation technique to determine the probability of match for a given proximity relation.

1 Introduction

Location-based services have become an important industrial sector [10]. This is largely due to the proliferation and miniaturization of various mobile devices that are capable of tracking the position of subscribers. New applications of location-based services range from buddy tracking (e.g., *"notify me if my best friend is nearby"*), group tracking (e.g., *"notify me if all members of a group of friends are nearby"*), mobile gaming (e.g., treasure hunts) to mobile advertisement (e.g., *"notify me if I am within 50 meters of a coffee shop"*). All these applications require the continuous tracking of a *proximity relation* of a set of mobile subscribers or a set of mobile subscribers and a static point of demarcation. Existing work studies this problem under the assumption that the location constraints all have the same alerting distance and location data is accurate [14, 20]. Other approaches solve the related, but different, closest pair and k nearest neighbor problem [4, 9, 21]. In our own prior work [20] we model the proximity relation among moving objects with two types of location constraints: the n-body and n-body static constraints. Both constraints assume precisely defined positions of the associated moving objects. It is this assumption that we relax in this paper. Moreover, this paper considers the evaluation of a single constraint and not the continuous tracking of a large set of these constraints.

The *n-body constraint* is of the form $|{p_1}^t, {p_2}^t, ..., {p_n}^t| < d$. It is satisfied if the n moving objects, identified by, $p_1, p_2, ..., p_n$, can be enclosed by a sphere with diameter d at some time, t. This constraint models whether a given set of objects is in proximity to each other, where proximity is defined to be within a sphere of diameter d. d is referred to as the *alerting distance*.

The *n-body static constraint* is of the form $|A, {p_1}^t, {p_2}^t, ..., {p_n}^t| < d$, where A is the coordinate of some static point. It is satisfied if the n moving objects, identified by, $p_1, p_2, ..., p_n$, are within the given range, d, of the static point A at some time, t. This constraint models whether a given set of objects is in proximity to a static point A, where proximity is defined to be within a sphere of radius d of the static point A.

Intuitively, when objects are in close proximity, their pairwise distances are small. This intuition can be formalized to show that the proximity relation defined by the enclosing circle (our n-body constraints) and the pairwise distances are equivalent. We capture this in the following property, formally proved in [19].

Property: (1) if the pairwise distance of a set of n objects is below d, then these objects can be covered by a circle with the diameter $\frac{2\sqrt{3}}{3}d$. (2) if a set of n objects can be covered by a circle with diameter d, then the pairwise distance of these objects is smaller than d.

Establishing the pairwise distance among n objects has quadratic complexity ($O(n^2)$). The enclosing circle compu-

tation is linear in the number of object ($O(n)$), as we review in Section 3. This suggest that using n-body constraints as proximity relation is more efficient.

In [20] we develop algorithms for the efficient evaluation of large numbers of these constraints under the assumption that the location position is given as a precise point in 2-D space and the position data is reported to the constraint evaluating server periodically without cost considerations (e.g., location query cost, network bandwidth cost etc.) In reality, however, the imprecision of the location information is an intrinsic characteristic of location positioning. The exact position of a mobile object at a point in time is hardly ever consistent with the last location update of the object. The imprecision of the position mainly stems from two factors, (1) imprecision of location positioning technique used and (2) discrete location update policy. Even if the position is accurate at the point in time the location is reported, during the time in between location quotes, the precise position is unknown due to the continuous movement of the object. Therefore, neither the location position data, nor the constraint matching results are accurate. The periodic manner underlying the location updates may further exacerbate the missing of matches. The location constraint is only evaluated against discrete location updates, entirely missing potential constraint matches in between updates (Fig. 1).

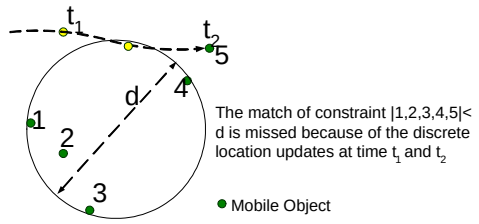

Figure 1. Match Missed

To take these issues into account, we model the position of the object with an uncertainty region, which models the unknown position of the object as a circle around the last known position of the object. The actual position of the object is located somewhere inside the circle. The circle is defined by the object's velocity and the accuracy of the location positioning technology used. Now, the challenge becomes to evaluate the above defined n-body (static) constraints over the uncertainty regions of the associated objects and to determine whether a constraint is satisfied, satisfied with some probability, or not satisfied.

Depending on the source of the location information (i.e., computed and available on the device or computed by the network infrastructure and available at the server), we propose two constraint evaluation algorithms, the Nearest Update Time (NUT) and the Safe Region Update (SRU) algorithms.

The contributions of this paper are four fold: (1) We introduce and formalize the lower and upper bound of the size of the smallest enclosing circle which covers a given set of mobile objects with imprecisely given positions and propose an algorithm to compute these bounds. (2) We develop two cost-effective algorithms, NUT and SRU, for evaluating location constraints and scheduling the location updates under position imprecision. (3) We develop a technique to efficiently approximate the probability of match, if a match cannot be determined unequivocally. We also develop an optimization for pruning constraints to avoid their evaluation. (4) We conduct extensive experiments to evaluate our algorithms under varying experimental conditions.

The outline of this paper is as follows. Section 2 briefly reviews current location position techniques, mostly focusing on their reported accuracies. Section 3 proposes a recursive algorithm to effectively compute the upper and lower bounds of the circle enclosing the objects involved in a constraint under uncertainty. Section 4 develops two new constraint evaluation algorithms for the constraint evaluation and the location update scheduling. A sampling method is used to estimate the matching probability. A strategy to prune constraints prior to their evaluation are detailed in Section 5. Performance evaluations are presented in Section 6. Finally, in Section 7 we put our approach in perspective to related work.

2 Problem Context

In this section we briefly present background information to further motivate the imprecision inherent to location positioning techniques. We first review data we gathered to establish the accuracy of a commercial location positioning technology available to us and then review location positioning technology deriving two mainstream categories, which motivate our algorithm design in subsequent sections.

Most location positioning techniques are imprecise. In a prior application-oriented study we leveraged the location positioning technology of a wireless carrier. The technology combines GPS, CDMA network triangulation, and cell site location to identify a subscriber's location. Position data is retrieved in a pull-based manner. To evaluate the accuracy of this location positioning approach we conducted an experiment recording locations retrieved over time under different environmental conditions. A location request to the network infrastructure is responded to with a location position and the accuracy that was achievable. Fig. 2 summarizes the data recorded. The data suggests that outdoor environments and good weather conditions greatly improve the accuracy of the position returned. Under most favorable conditions (i.e., in the park on a sunny day) the accuracy reached up to about 10 meters. However, under least favorable conditions (i.e., inside an office at a rainy day), the

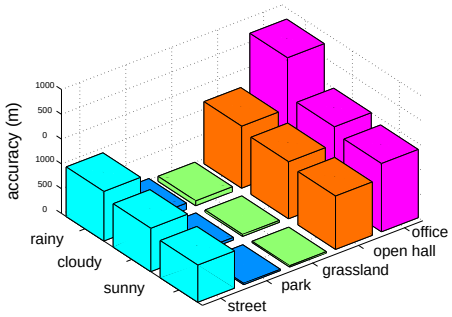

Figure 2. Accuracy of Location Data

accuracy deteriorated by up to 1000 meters for repeated location requests from the same location.

There are many other location positioning techniques [10]. As reported in the literature, the accuracy of GPS in the open air is around 10-300m [5]. WiFi (e.g., 802.11) can be interpreted as an indoor form of GPS for location positioning, with the access points acting as satellites. At least three access points are necessary to calculate a position, and additional access points improve accuracy. The accuracy of WiFi highly depends on the density of the access points, and is reported to be around 2-50m [2]. Active badge and RFID use radio frequency identification to position the user for an estimated accuracy to within a few meters in a designated area [15, 1]. Other tracking methods include Cricket, GSM fingerprinting, and Ubisense etcetera [8, 7]. None of these existing methods can provide perfectly accurate location positioning. All approaches are subject to various levels of inaccuracy. Apart from the accuracy, the methods also differ in terms of where the position information is calculated. Two categories determining location-aware algorithms design can be distinguished:

1. *Externally supported devices*: many WiFi-based approaches, active badges, and the network we used in our evaluations are examples. For devices in this category, either the device cannot independently obtain the position data (e.g., no GPS available on the device) or the device has very limited capabilities or power and is not capable of computations such as evaluating whether its position is within a certain geographic area, for example.

2. *Self-sustained devices*: onboard GPS and many telecom network solutions are examples. The position is calculated typically with triangulation based on the signal strength or time of arrival and delay from a number of satellites, access points, base stations or antennas by the mobile device itself. Often devices in this category are capable of other multi-purpose computations.

Generally speaking, the self-sustained device is considered more powerful than the externally supported device. However, a self-sustained device is usually more power-consuming, more expensive and more bulky. This difference in functionality requires location-aware algorithms to work with devices and location positioning techniques from either of the two categories.

3 Computing Uncertainty Bounds

In this section we develop algorithms to decide whether a n-body (static) constraint is satisfied, satisfied with a probability, or not satisfied. The constraint is defined as discussed in the introduction, with the positions of the associated objects defined by an uncertainty region (i.e., not precisely known.)

At time t, the position of the object i is represented by the uncertainty region UR_i^t. UR_i^t is a circle in 2-D space with the last position update of object i as the center and $\delta_m + v_i^{max}(t - t_l)$ as the radius. δ_m is the accuracy of the tracking method m, v_i^{max} is the maximum velocity of object i and t_l is the last location update time prior to t. The real position of the moving object is anywhere inside the uncertainty region. For many tracking methods, the position of the object is reported actually as a circle with a certain probability distribution function (PDF) describing the likelihood of the object appearing at any point inside the circle. GPS, for example, adopts the normal distribution as the PDF. Empirically, it is also feasible to establish the parameters of the PDF experimentally, but this is outside the scope of this paper.

For the n-body constraint, the smallest enclosing circle covering its associated objects could be as small as the smallest circle intersecting all the uncertainty regions of the objects or it could be as large as the smallest circle enclosing all the uncertainty regions of the objects. We call these circles the *internal circle* (denoted as C_{int}^t) and the *external circle* (denoted as C_{ext}^t), respectively. They represent the *lower* and *upper bounds* of the circle that actually encloses all the moving objects. Fig. 3 shows 6 uncertainty regions (dotted), their internal circle (dashed), and their external circle (solid).

Suppose, at time t, the smallest circle intersecting all uncertainty regions has the diameter D_{int}^t and the smallest circle enclosing all uncertainty regions has the diameter D_{ext}^t. If $D_{ext}^t < d$, then the constraint is satisfied with probability 1; if $D_{int}^t \geq d$, then the constraint is not satisfied (i.e., probability 0); if $D_{int}^t < d \leq D_{ext}^t$, then the constraint is satisfied with some probability, which is estimated depending on the distribution function of the object position inside the uncertainty region. In the last case, the estimated probability of match is returned to the user, and its interpretation are up to the application.

Enclosing circle computation for n-body constraints:

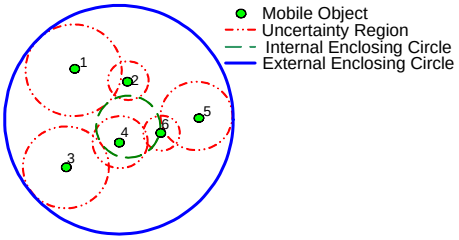

Figure 3. Bounds of Uncertainty Regions

The computational geometry literature developed algorithms for computing the circle enclosing a set of points or circles [18, 16, 3]. However, these approaches do not discuss algorithms to compute the internal circle intersecting a set of circles. We have adapted the methodology from Welzl [16] to solve this problem. Below, we develop a recursive algorithm for computing both internal circle C^t_{int} and external circle C^t_{ext} for a set of uncertainty regions.

The computation of the internal circle is done in a recursive manner. Suppose that we want to compute the internal circle, C_j, for j uncertainty regions. We first pick an arbitrary one of the j regions and compute C_{j-1} for the remaining $j-1$ uncertain regions. We put the last one that has been taken out back; if it intersects C_{j-1}, then we are done, $C_j = C_{j-1}$. If it does not intersect C_{j-1}, then the last circle inserted is a bounding region of C_j (it is tangent to C_j). A similar property holds for the external circle. This is stated in the following lemma. The proof can be found in the extended technical report [19].

Lemma 1 *Suppose that C_{j-1} is the internal (or external) circle of $j-1$ uncertainty regions, $UR_k, (1 \leq k \leq j-1)$, and C_j is the internal (external) circle of j uncertainty regions, $UR_k, (1 \leq k \leq j)$. If $UR_j \cap C_{j-1} = \emptyset$ ($UR_j \notin C_{j-1}$), then UR_j is a bounding region of C_j.*

The above lemma suggests the following recursive algorithm for computing the internal circle for a set of uncertainty regions. The main function (`INTERNAL`) computes the internal circle of a set of uncertainty regions, UR. It arbitrarily takes one uncertainty region out of UR (line 3) and computes the internal circle C for the rest of the uncertainty regions (line 4); then the one taken out is put back, if it is intersecting C (check with function `INTERSECT`), we are done; if not, a second function `INTERNAL_1BD` is called and it computes the internal circle of UR with one bounding uncertainty region that is taken out initially (line 6). `INTERNAL_1BD` works in a similar fashion. It calls `INTERNAL_2BD` which computes the internal circle with the knowledge of two bounding uncertainty regions, and so on. It is not difficult to prove that the time complexity of the algorithm is $O(n)$ (n is the number of uncertainty regions). For each call to `INTERNAL` `INTERNAL_1BD` or `INTERNAL_2BD`, the size of the first parameter UR will be reduced by one, unless it is already 0.

The recursive function to compute the external circle is nearly the same as `INTERNAL`. It is therefore omitted. The only difference is that all `INTERSECT` functions below are replaced with the function `ENCLOSE`, which checks whether the circle is enclosing an uncertainty region. It also has a linear complexity.

Algorithm INTERNAL(UR)
1. **if** ($|UR| == 0$) **return** ϕ;
2. randomly choose $ur_k \in UR$;
3. $UR = UR$ - ur_k;
4. C = `INTERNAL`(UR);
5. **if** !`INTERSECT`(C, ur_k)
6. C = `INTERNAL_1BD`(UR, ur_k);
7. **return** C;

Algorithm INTERNAL_1BD(UR, B_1)
1. **if** ($|UR| == 0$) return B_1;
2. randomly choose $ur_k \in UR$;
3. $UR = UR$ - ur_k;
4. C = `INTERNAL_1BD`(UR, B_1);
5. **if** !`INTERSECT`(C, ur_k)
6. C = `INTERNAL_2BD`(UR, B_1, ur_k);
7. **return** C;

Algorithm INTERNAL_2BD(UR, B_1, B_2)
1. **if** ($|UR| == 0$) **return** circle intersecting B_1 and B_2;
2. randomly choose $ur_k \in UR$;
3. $UR = UR$ - ur_k;
4. C = `INTERNAL_2BD`(UR, B_1, B_2);
5. **if** !`INTERSECT`(C, ur_k)
6. C = circle intersecting B_1, B_2 and ur_k;
7. **return** C;

Enclosing circle computation for n-body static constraints: The internal circle and external circle of a set of uncertainty regions for n-body static constraint are slightly different from the n-body constraint, because the center of the circle has to be at a static point. The computation of the lower and upper bounds (the internal and external circles) for n-body static constraint is given in the BOUND_NBS algorithm. Here, o_i and r_i are the center and the radius of the uncertainty region i. Since the static point A is already given, the radius of the internal circle is the maximum of the smallest distances from A to all the uncertainty regions ($|o_i - A| - r_i$), the radius of the external circle is the maximum of the maximum distances from A to all the uncertainty regions ($|o_i - A| + r_i$) [1].

Algorithm BOUND_NBS(UR, A)
($*$ Computation of the internal and external circle of UR centered at A $*$)
1. **for** $ur_i \in UR$
2. o_i = center of ur_i;
3. r_i = radius of ur_i;
4. $r_{int} = max(|o_i - A| - r_i)$;
5. $r_{ext} = max(|o_i - A| + r_i)$;

[1] The smallest (maximum) distance from a point to a circle is the smallest (maximum) Euclidean distance from that point to any point on the boundary of the circle.

4 Constraint Evaluation Algorithms

For each one of the two models of operation of location tracking, we develop one algorithm for addressing the constraint evaluation problem. The Nearest Update Time (NUT) algorithm is developed for processing constraints of mobile objects that are externally supported. The Safe Region Update (SRU) algorithm targets constraint processing for mobile objects that are self-sustained. In the following subsections, we focus on n-body constraint to illustrate the algorithms. For the n-body static constraint, the solution is similar.

4.1 The Nearest Update Time Algorithm

The NUT algorithm is given below. NUT first computes the upper and lower bounds for the circle enclosing the uncertainty regions. If the diameter of the external circle is smaller than the alerting distance, the constraint is matched with matching probability 1 (line 3). We define the location update bound as a circle whose diameter equals the alerting distance and whose center coincides with the center of the external circle. NUT then computes the closest point in time, t, when some uncertainty region will go beyond this location update bound (line 4). See Fig. 4 for an illustration. Finally, t is returned to the user who uses it to trigger its next location update (line 5). t could also be used by the server as the next location probe time, if the tracking device does not support a time trigger. If different maximal velocities of objects are allowed or the moving direction are known (expressed in vector $v_{p_i}^{max}$), the center of the location update bound is the center of the enclosing circle with offset $\sum v_{p_i}^{max} t_\delta$. t_δ is chosen such that the closest time point some uncertainty region goes beyond this bound is maximized.

However, if the diameter of the internal circle is larger than the alerting distance, the constraint is not matched (line 6). In this case, NUT computes the time points that any two uncertainty regions getting closer than the alerting distance and the latest time point of which is returned to the users (line 9) as the next time point for a location update. Assuming a unique global maximum velocity, only the furthest pair of regions need to be checked. This further reduces the complexity of the comparison.

Function *matching_probability* (line 10) estimates the matching probability using the sampling method introduced in Section 4.3; the result is returned to the user (line 11) and further interpretation of this result is made with the application level logics. In this case, the next location update time is set to some default value, $t_{default}$. $t_{default}$ depends on the tracking device or constrained to resources, e.g., network bandwidth or battery power, it could also be a function of matching probability, p_{match}, computed, e.g., the closer p_{match} is to 1 (any critical value), the high sampling frequency (smaller $t_{default}$) is required.

Algorithm NUT($UR, alertdist$)
1. compute C_{ext} and C_{int} of UR;
2. **if** $C_{ext}.d < alertdist$
3. p_{match} = 1;
4. t = MIN (time point $ur_i (\in UR)$ goes beyond the circle $[C_{ext}.o, d = alertdist]$);
5. **return** [p_{match}, t] to all;
6. **if** $C_{int}.d > alertdist$
7. p_{match} = 0;
8. t = MAX (time point when distance b/w. ur_i *and* $ur_j (\in UR)$ $\leq alertdist$);
9. **return** [p_{match}, t] to all;
10. p_{match} = $matching\_probability(UR, alertdist)$;
11. **return** [p_{match}, $t_{default}$] to all;;

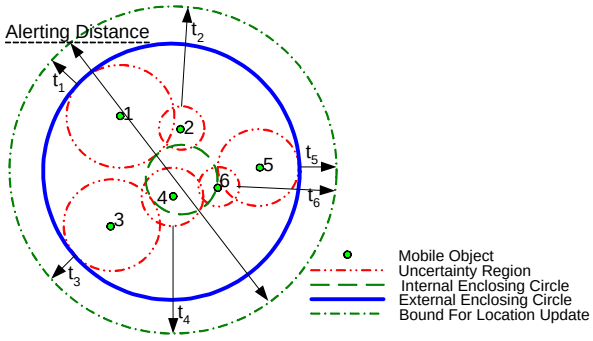

Figure 4. Next Time Point for Location Update

4.2 Safe Region Update Algorithm

The SRU algorithm is given below. Similar to NUT, first, the upper and lower bound of the enclosing circle is compared with the alerting distance (line 2, 6) to determine if the constraint is matched. The difference from NUT is that when the constraint is matched, the safe region is computed and returned to the user (line 5). A safe region is a circle centered at the external circle's center with the alerting distance as the diameter (line 4). The mobile user tracks his own position and does not need to update its new location unless it moves beyond the safe region.

In case of a mismatch (line 6), the safe region is defined as the region outside a stripe (formed by two parallel lines) with the alerting distance as the width and bisecting the smallest line segment between some uncertainty regions ur_i and ur_j (line 9). This is illustrated in Fig. 5. ur_i and ur_j could be any pair whose distance is larger than the alerting distance. For simplicity, we pick the pair with the furthest distance in our implementation. The intuition is that as long as the distance between ur_i and ur_j is no less than the alerting distance, this constraint can not be satisfied. Objects i and j are called *sentinel objects*, because before the next safe region is computed they are the only objects that track if a match is even possible. The safe region computed in this manner is returned only to user i and j. The rest of

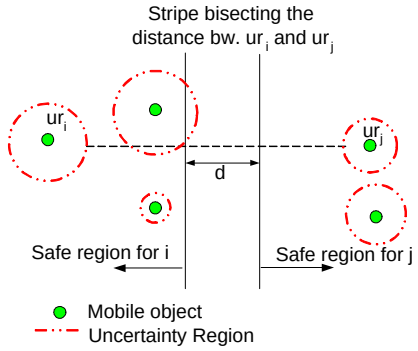

Figure 5. Safe Region Outside the Stripe

the objects receive "$infinity$" as the safe region (line 10). Next, i or j track their own position and do not need to update the location as long as the uncertainty region does not go beyond the safe region (i.e., intersects the stripe). On receiving the location update from i or j, the server sends out a forced location update message to all objects and forces them to update their location. After the server collects the position information from all the objects, new sentinel objects are validated and the safe region is recomputed based on the new sentinel objects. The movements of the other objects (other than i and j) are not constrained, they will not initiate any location update until a forced location update message from the server is received. If such sentinel objects do not exist (e.g., the distances of all pairs are smaller than the alerting distance) then the safe region is set to "-" (line 11). On receiving "-", each user updates his location with some default frequency. Note that we only list SRU algorithm running on the server side. The operations on the object's side involves checking its position against the safe region, waiting for a forced location update request or updating its location with a default frequency, depending on the latest message from the server. Finally, *matching_probability* is used to estimate the matching probability (line 12) if a match can not be crisply determined.

Algorithm SRU($UR, alertdist$)
1. compute C_{ext} and C_{int} of UR;
2. **if** $C_{ext}.d < alertdist$
3. p_{match} = 1;
4. SafeRegion = $[C_{ext}.o, d = alertdist]$;
5. **return** [p_{match}, SafeRegion] to all;
6. **if** $C_{int}.d > alertdist$
7. p_{match} = 0;
8. **if** $\exists\ ur_i, ur_j\ (\in UR)$ at least $alertdist$ apart
9. SafeRegion = region outside the strip with width $alertdist$ that bisects the distance b/w. ur_i, ur_j;
10. **return** [p_{match}, SafeRegion] to i and j and [p_{match}, $infinity$] to the rest;
11. **else return** [p_{match}, -] to all;
12. p_{match} = $matching\_probability(UR, alertdist)$;
13. **return** [p_{match}, -] to all;

Complication is expected when a constraint involves both externally supported and self-sustained tracking devices, and this complication can be smoothly resolved with a hybrid approach, which uses a combination of both NUT and SRU algorithms.

4.3 Calculation of Matching Probability

It may not be possible to determine a constraint match, due to the position uncertainty of the involved objects. The question then becomes one of approximating the probability of match (to estimate the likelihood of the match). Even with perfect knowledge of the bounds of enclosing circle and the probability distribution function (PDF) of the moving objects, the precise computation of the matching probability is expensive because it involves numerical integration of the PDF over irregularly shaped regions. Typically, the response time is an important metric for the location-based services, because the position data and matching results are only valid for relatively short periods of time due to the ongoing movement. To prevent lengthy computations, we approximate the matching probability, with efficient, but less precise methods. This is a trade-off between the precision and the efficiency. In order to achieve efficiency, we choose to use a Monte Carlo simulation to estimate the probability of match. Assuming the position of the object follows some known PDF within the uncertainty region, our algorithm generates each user's position according to the PDF and checks whether the constraints are matched with the sampled position data. This sampling process is repeated a number of times and the percentage of match is used as an estimate of the matching probability. More formally, suppose that the sampling process is repeated k times, then the matching probability p_{match} is calculated as:

$$p_{match} = \frac{\sum_{i=1}^{k} match(|{p_1}^i, {p_2}^i, ..., {p_n}^i| < d)}{k}$$

${p_i}^i$ is the i-th generation of object i's position based on the PDF of i. Function $match$ above returns 1, if the constraint is matched, and 0 otherwise. As $k \rightarrow \infty$, p_{match} becomes more accurate, it converges to a stable value. Through experimentation, we find that very few samples are needed to obtain an estimation (e.g., 10%-15% relative standard error).

5 Optimizations

To reduce the circle bound computations, we assume that the size of an uncertainty region has an upper bound. This can be achieved by forcing location updates (with some time threshold or distance deviation threshold above which the location update must be issued). [2]

[2] If the uncertainty region goes beyond this bound (e.g., because the object fails to update its location for a very long time), we assume the object is disconnected and the evaluation of the location constraints it is associated with are suspended until further location updates are received.

Further more, the whole space is partitioned with grids and it is used to index the moving objects in space. We define the active bound of the object as the *Minkowski* sum of the upper bound of uncertain region of the object and the grid. The Minkowski sum of two sets B(uncertain region) and G_i (grid), denoted as $B \oplus G_i$, is defined as $B \oplus G_i = \{p + q | p \in B, q \in G_i\}$, where $p + q$ is the vector sum of the vectors p and q. (See Fig. 6)

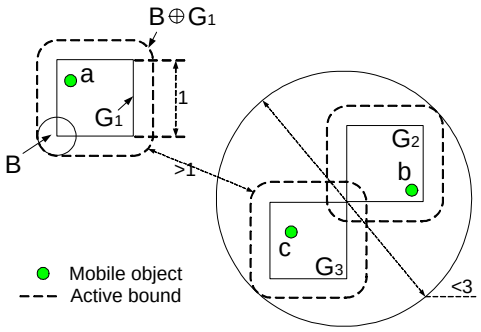

Figure 6. Constraint Pruning

If there exist two active bounds (for the objects in a constraint) whose distance is greater than the alerting distance, then this constraint cannot be satisfied as long as the mobile objects remain within their original grids. On the other hand, if the active bounds of all the objects involved in the constraint can be covered by a circle with the diameter smaller than the alerting distance, then this constraint must be satisfied as long as the objects remain within the grids. Fig. 6 illustrates that constraint $|b, c| < 3$ is satisfied, because the active bounds of the partition G_2, G_3 can be covered by a circle with diameter 3. Constraint $|a, c| < 1$ is unsatisfied, because the distance between active bounds of the partition G_1 and G_3 is larger 1. These constraints are immediately pruned based on the grid information (no bound computation is necessary). Only the constraints that can not be evaluated solely by the partition information are subjected to further processing. The computation of the constraint based on partition information happens only when the object moves across the grids (referred to as a partition update) and it can reduce the constraint processing time to more than 60%, as shown in our experimental result.

If one object is associated with multiple constraints, each location update may affect a number of constraints and the next location update time (for externally supported devices) is rescheduled for each one of them and the closest one in time is used as the next location update time. Likewise, the safe region (for self-sustained devices) is the intersection of safe regions of all constraints involved.

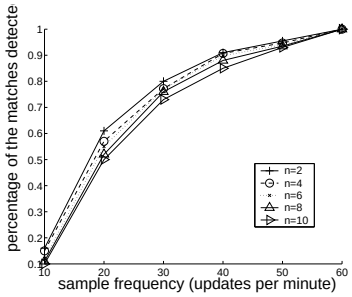

Figure 7. Location Update Frequency and Matches Detected

6 Evaluation

We have fully implemented both algorithms NUT and SRU in the L-ToPSS prototype [3]. In the experiment, each constraint is associated with a different number of objects and has an alerting distance drawn uniformly in the range [1000m-1200m]. All objects are moving randomly with a velocity drawn uniformly from the range [1m/s-10m/s]. In the experiments we assume that location tracking techniques have various accuracies, uniformly distributed in [20m-500m].

In the first experiment, we model 10,000 n-body location constraints associated with 10,000 moving objects. n is chosen to be 2, 6, and 10, respectively. We compare NUT and SRU against a basic scheme where the mobile objects update their location according to a predetermined update frequencies: 12/min (f_1), 30/min (f_2) and 60/min (f_3)[4] f_3 is chosen such that any frequency higher than f_3 does not result in an increase of the matches detected. Fig. 7 shows the percentage of matches detected as a function of location update frequency (with different bodies). A match is counted when the probability of match is greater than zero. f_3 is considered as most sensitive to matches (i.e., by capturing all matches.) The frequencies f_1 and f_2 reduce the number of location updates by a factor of 5 and 2, respectively; however, at the cost of missing matches (less sensitive). With multiple simulation runs, we measure the location updates and matches (as a percentile with respect to the fixed sampling rate with frequency f_3), and the matches detected per location update (shown in Fig. 8).

Fig. 8(A) shows that the number of location updates with NUT and SRU are very close to the basic scheme with frequency f_1 (about 20-30% of the f_3). However, NUT and SRU capture almost all matches (close to 1 as in Fig. 8(B)). Notice that the bars for NUT and SRU are the measurements when the default location update frequency

[3] L-ToPSS is the Location-aware Toronto Publish/Subscribe System research project

[4] f is location updates per unit of time.

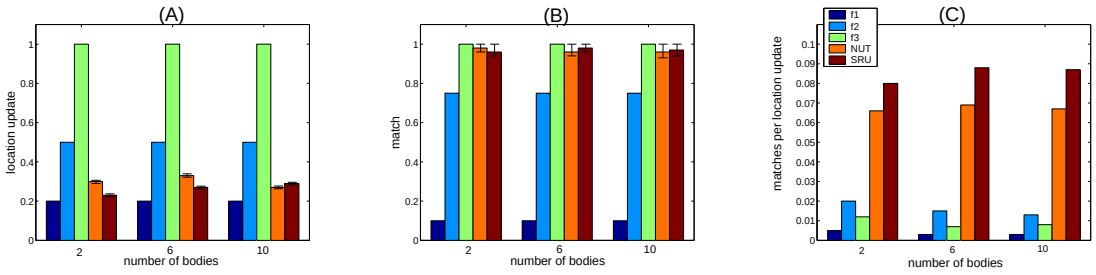

Figure 8. Sensitivity to Matches

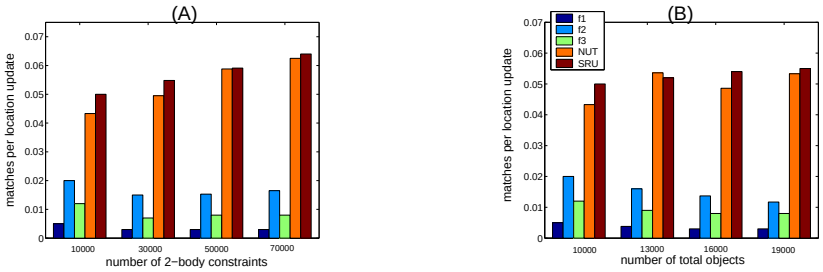

Figure 9. Scalability on Loads of Constraints and Objects

($1/t_{default}$) equals f_2; the lower and upper bounds of the error bar are the measurements when the default location update frequency is set to f_1 and f_3, respectively. We observe that the difference between the bounds are less than 6%, which indicates that the choice of the default update rate does not have much effect on the performance. In terms of the matches detected per location update (Fig. 8(C)), in which the difference between default frequencies is diminished, NUT and SRU are ranked at the top, which are four to five times better than a periodical sampling algorithm. SRU is more efficient because with safe region management, the next location update time is not strictly specified. In Fig. 9(A), for a fixed number of moving objects (10,000), the number of 2-body constraints is increased from 10,000 to 70,000. A slight increase of matches per location update is observed as the constraints increase. For a large number of location constraints with relative low number of moving objects, each position information is exploited for an aggregation of constraints, this makes the algorithm more efficient. However, we do not see this trend when we increase the number of total objects with a fixed number of constraints (Fig. 9(B)). As with more objects, each object is associated with less constraints and the effect of aggregation is diminished. For the above two experiments, NUT and SRU capture almost all the matches. (data omitted due to space limits.)

The second experiment evaluates the capability of constraint pruning with grid-based partitioning. We track the average number of location constraint computations (i.e., calls to the function NUT or SRU) with or without grid index optimization every minute. As shown in Fig. 10(A), the number of computations for evaluating constraints is reduced by more than 60%, and the number of location updates is reduced to around 20% of the algorithm without optimization. This is because many constraints are pruned with the grid index, and bounds computation and location updates are not necessary. Again, in this experiment, NUT and SRU capture almost all the matches.

To approximate the probability of match, we sample the users' positions according to a given PDF which is set to be a normal distribution with the latest reported position as the mean and 1/6 of the diameter of the uncertainty region as the standard deviation. As more samples are taken, the result converges to the probability of match. The experiment in Fig. 11(A) evaluates the cumulative probability that the probability sampled is within a certain relative standard error range (10% - 15%) of the approximated value, which is considered to be the true value of the probability of match (which is obtained with over 300 samplings). The cumulative probability is collected from multiple runs of the simulation for 2-body (static and non-static) constraints. The cumulative probability shows that within 20 samples over 90% of the computed probability is within 15% of the relative standard error (rse) of the true probability (converged value). To reach 10% relative standard error, about 25 samples are required. This means that the sampling approach is

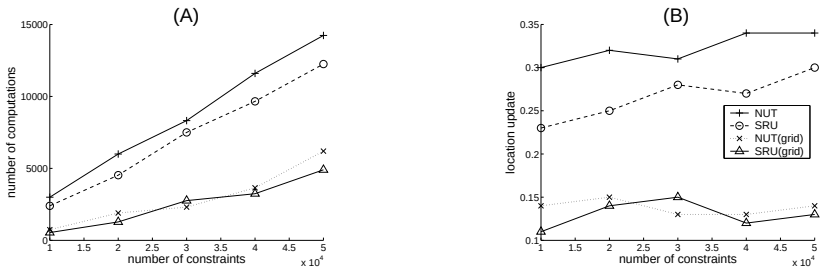

Figure 10. Optimization with Grid index

very cost-effective for coarse probability estimations (i.e., 10% - 15% relative standard error).

When the constraint involves more bodies, the cumulative probabilities for the n-body constraint (n = 2,4,6,8) are converging quickly (Fig. 11(B)); the cumulative probabilities for 4, 6, and 8 bodies are very close to each other. This is because only the bounding uncertainty regions determine the upper/lower bound, therefore, most of the time, only the samples drawn from these regions affect the probability computation. For the static constraint (Fig. 11(C)), the result is similar. Adjusting the standard deviation makes only a small difference, requiring a few more samples for the larger deviation value to reach the same standard error.

7 Related Work

Determining the *close-to* relationship among, either a set of mobile, or static entities has received wide attention in the literature. Corral *et al.* [4] study the problem of determining the k closest pairs between two spatial sets of static points. Their approach is similar to a join query and discovers the k smallest distances between two sets of points. Different from the location constraint processing, this work does not consider the *close-to* relationship involving more than two entities and the points are partitioned into two groups, the distance between the pairs from the same group is not counted.

The nearest neighbor problem determines the nearest object(s) to a given point among all the objects in the space [9, 21]. This is a different query type from the location constraint matching problem, we are looking at, which determines whether a specific set of objects are in a given spatial constellation to each other at a given point in time.

The buddy tracking system by Amir *et al.* [14] is the only work known to us that is looking at a problem statement similar to our location constraint matching problem. Amir *et al.* are, however, exclusively looking at a 2-body problem in the 2-D space and propose and evaluate a distributed algorithm for solving this problem. They assumed that the mobile objects communicate with each other directly to resolve the 2-body constraints. Their objective is to reduce the communication cost. Moreover, their approach is constrained by the assumption that all constraints are defined with the same alerting distance.

All the above approaches do not take the uncertainty of the position data into consideration. All approaches assume the given location position is accurate.

As far as we know, the position uncertain problem is first studied in [17], and extended in [12, 13]. Wolfson *et al.* propose a number of cost-based approaches in order to bound the error (deviation) between actual position of a moving object and the position stored in the database. They postulate that there is a given cost for each unit of deviation and there is a given cost of an update. The location update is sent only when the deviation-cost exceeds the update cost. Their work generally focuses on eliminating the imprecision of mobile data (i.e., reducing the difference between the actual position and the database recorded position of the object), which might not be optimal for a specific query evaluation, because in the latter case, the query evaluation cost also needs to be considered.

A different location update policy is studied by Hu *et al.* [6]. They provide a generic framework to monitor continuous spatial queries (range and kNN queries) over moving object where the join of the safe regions of multiple queries is maintained for the moving objects. The location update is triggered only when the object moves outside of a safe region. This approach is similar to ours. However, the queries they consider are static (range query and nearest neighbor query with static query range and query point) which prevents the application of this scheme to our problem.

Probabilistic query processing has recently gained a lot of attention. For example, Tao *et al.* [11] proposes an U-tree structure for supporting a probabilistic range query, which returns the objects within a query range with a probability higher than a given threshold. They use the idea of a probabilistically constrained region to prune the query space and reduce the expensive integral computation. However, for our problem, the matching probability is the result of the evaluation rather than a means to prune the query space (i.e., constraints) and the computation of the true probabil-

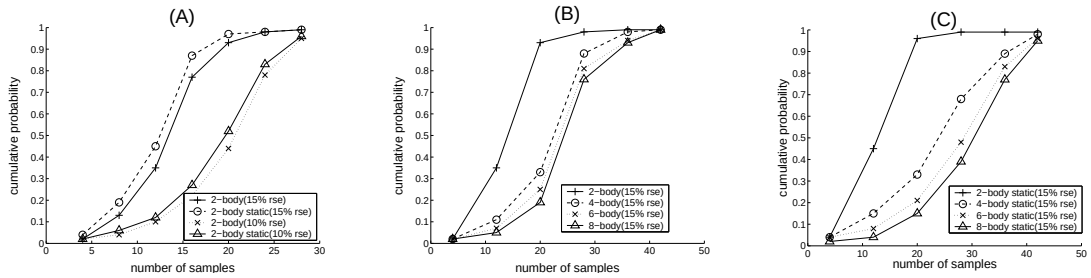

Figure 11. Cumulative Probability as a Function of the Number of Samples

ity is impossible to express as a closed-form function, therefore sampling is the only practically way to approximate the function.

8 Conclusions and Future Work

In this paper, we study the effects of imprecisely given location position data on the problem of location constraint matching. We primarily focus on developing algorithms for efficiently evaluating a single constraint in this setting. We present an algorithm to compute the upper and lower bounds of the circle intersecting the regions of uncertainty delimiting the unknown location of mobile objects. We then propose two algorithms (NUT and SRU) to do constraint evaluation in two different types of environments and under different location update scheduling policies. Experiments show that NUT and SRU significantly reduce the location update cost, while maintaining sensitivity to the matches, performing close to what would be achieved under maximum location update frequency. Our sampling approach to estimate the probability of match for cases where a precise answer can not be given is very cost-effective. For example, 25 samples bring the relative standard error down to 10%. Also, using space partitioning and active bounds, about 60% of the constraints can be pruned without evaluations, which improves the system performance significantly.

In future work, we intend to study the uncertainty of proximity matching on road network where the objects are only moving on the edges of the road network and the metric used is the network distance rather than the Euclidean distance. This will motivate a number of interesting and practical applications.

References

[1] RFID Journal. *http://www.rfidjournal.com*.

[2] P. Bahl and V. N. Padmanabhan. Radar an in-building rf-based user location and tracking system. In *IEEE INFOCOM 2000*.

[3] M. De Berg. In *Computational Geometry*. Springer-Verlag New York, 2000.

[4] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Closest pair queries in spatial databases. In *Proceedings 1994 ACM SIGMOD Conference, Dallas, TX*, pages 189–200, 2000.

[5] A. El-Rabbany. Introduction to GPS: The Global Positioning System. Artech House Publishers, 2002.

[6] H. Hu, J. Xu, and D. L. Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *SIGMOD*, 2005.

[7] V. Otsason, A. Varshavsky, A. LaMarca, and E. de Lara. Accurate gsm indoor localization. In *7th International Conference on Ubiquitous Computing (UbiComp), Tokyo, Japan, September 2005*.

[8] N. Priyantha, A Miu, H. Balakrishnan, and S. Teller. The cricket compass for context aware mobile applications. In *MOBICOM 2001*.

[9] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, 1995.

[10] J. Schiller and A. Voisard. Location-Based Services. The Morgan Kaufmann Series in Data Management Systems, 2004.

[11] Y. Tao, R. Cheng, X. Xiao, W. Kay Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*, 2005.

[12] G. Trajcevski, O.Wolfson, S. Chamberlain, and F. Zhang. The geometry of uncertainty in moving objects databases. In *EDBT*, 2002.

[13] G. Trajcevski, O. Wolfson, and S. Chamberlain K. Hinrichs. Managing uncertainty in moving objects databases. In *TODS*, 2004.

[14] A. Amir. A. Efrat. J. Myllymaki. L. Palaniappan. K. Wampler. Buddy tracking - efficient proximity detection among mobile friends. In *INFOCOM*, 2004.

[15] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. In *ACM Transactions on Information Systems (TOIS) archive*, 1992.

[16] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, LNCS. Springer, 1991.

[17] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and imprecision in modeling the position of moving objects. In *ICDE*, 1998.

[18] S. Xu, R. M. Freund, and J. Sun. Solution methodologies for the smallest enclosing circle problem. In *Computational Optimization and Applications*, 2003.

[19] Z. Xu and H. A. Jacobsen. In *Uncertainty of Location Constraint Processing*. Technical Report, www.cs.toronto.edu/∼zhengdao/report/ uncertaintyreport.pdf, 2005.

[20] Z. Xu and H. A. Jacobsen. Efficient constraint processing for location-aware computing. In *Mobile Data Management*, 2005.

[21] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *Proceedings of ICDE*, 2005.