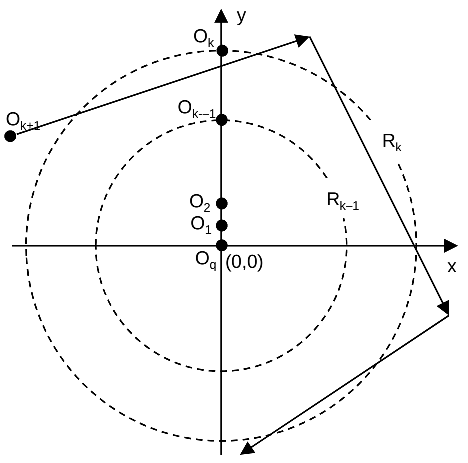
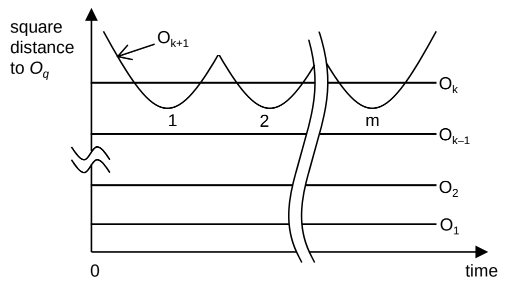
Appendix A: Proof of Proposition 2.5

PROPOSITION 2.5: *Assume that the query object and the moving objects all move piecewise linearly, where each object can have at most m linear pieces. $\beta_k(n) \geq 2m(n-k)+1$.*

PROOF: For the motion in Figure A(a) the curves arrangement is as in Figure A(b), and the details are as in Proposition 2.2. □



(a) Configuration in the motion space



(b) Arrangement in the Time-Square_Distance space

Figure A. The auxiliary figure for the proof of Proposition 2.5.

Appendix B: Proof of Lemma 3.5

LEMMA 3.5. *Let A be a connected sequence of m_A parabola-pieces and B be a connected sequences of m_B x-monotone parabola-pieces, in the Time-Square_Distance space. There are at most $2(m_A+m_B-1)$ intersections between A and B.*

PROOF. Denote by S_1 the set of pieces in sequence A that are intersected by the first piece in sequence B, by S_2 the set of pieces in A that are intersected by the second piece in B, and so on. Denote by |S| the size of a set S. Since any pair of pieces intersect at most twice, there are at most $2(|S_1|+|S_2|+...+|S_{m_B}|)$ intersections between A and B. On the other hand, due to the monotonicity of time, there is at most one common piece between S_i and S_{i+1}. Thus, the total number of distinct pieces in A that are intersected by B is at least

$$|S_1|-1+|S_2|-1+...+|S_{m_B}| = (\sum_{i=1}^{m_B}|S_i|)-(m_B-1)$$

Since the number of distinct pieces in A is m_A, the following inequality holds:

$$(\sum_{i=1}^{m_B}|S_i|)-(m_B-1) \leq m_A$$

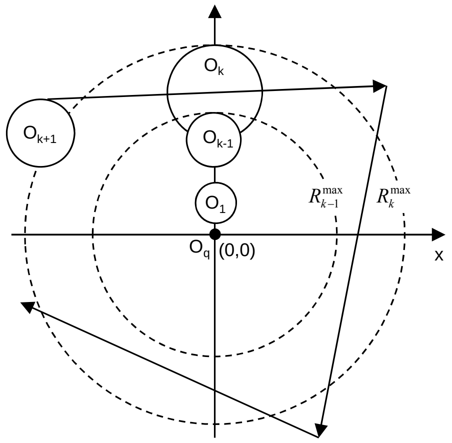
Thus, $\sum_{i=1}^{m_B}|S_i| \leq m_A+m_B-1$. Thus there are at most $2(m_A+m_B-1)$ intersections between A and B. □

Appendix C: Proof of Theorem 3.6

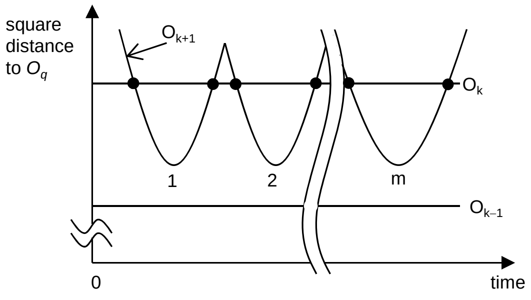
THEOREM 3.6. Assume that the query object and the expected locations of each data object move piecewise linearly, where each object has at most m pieces. Then for any constant k $\widetilde{\beta}_k(n) = \Omega(n^2 \cdot m)$.

PROOF: We prove the theorem by constructing a feasible case in which the number of answer-pairs is quadratic in *n* and linear in *m*. The construction proceeds as follows. Let object O_q and the expected locations of O_1, O_2,…, and O_k be static in the motion space such that O_i has the *i*-th maximum possible distance to O_q (see Figure B(a)). Denote by $R_i^{\max}$ the circle the center of which is the location of O_q and the radius of which is the maximum possible distance between O_i and O_q. Let the route of the expected location of object O_{k+1} have *m* linear pieces such that for each piece, the trace of O_{k+1}'s farthest-to-O_q-point intersects $R_k^{\max}$ twice but does not intersect $R_{k-1}^{\max}$, as shown in Figure B(a). Let O_{k+2} have the same route as O_{k+1} and move behind O_{k+1} such that its farthest-to-O_q-point enters $R_k^{\max}$ after that of O_{k+1} leaves $R_k^{\max}$. Construct the same for O_{k+3},…, $O_{n/2}$. Figure B(b) shows the max-curves of the first *k*+1 objects.

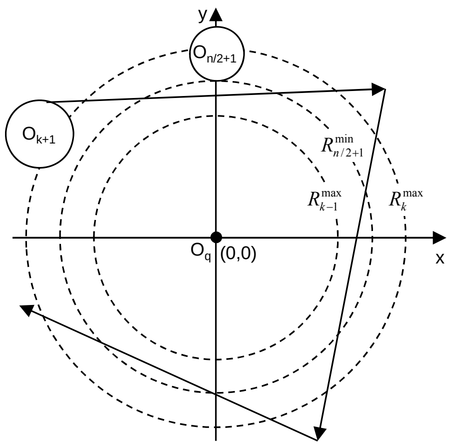
Denote by $R_i^{\min}$ the circle the center of which is the location of O_q and the radius of which is the minimum possible distance between O_i and O_q. Let $O_{n/2+1}$ be static such that $R_{n/2+1}^{\min}$ intersects the trace of O_{k+1}'s farthest-to-O_q-point for 2*m* times (see Figure B(c)). Do the same construction for $O_{n/2+2}$,…, O_n. Figure B(d) shows how the min curves intersect the max k-level. It is not difficult to see that there are $2 \cdot \frac{n}{2} \cdot m \cdot (\frac{n}{2}-k)$ critical intersections.□



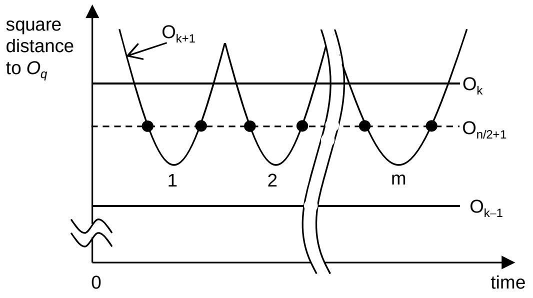
(a) Configuration of the first *k*+1 objects in the motion space.



(b) Max-curves of the first *k*+1 objects.



(c) Configuration of $O_{n/2+1}$ in the motion space.



(d) Intersections between the min-curve of $O_{n/2+1}$ and the max *k*-level.

Figure B. The Concrete Example for the Proof of Theorem 3.6

Appendix D: Explicit Updates and Piecewise Linear Model in the Uncertain Case

Addition. Consider the addition of an object *O′* at a time point *t′*. The first step is to add *O′* to *MOH* in O(log*n*) time as described in section 4.2.4. Depending on how the root node *r* changes as a result of the addition, there are three cases for the second step.

Case 1: The max-curve of *O′* is always farther from O_q than that of *r.object* between *t′* and *r.time*. In this case, neither *r.object* nor *r.time* changes. To process this case, compute the intersections between the min-curve of *O′* and the max-curve of *r.object* that will occur between *t′* and *r.time*. Create an intersection event for each of these intersections (at most two) and insert it to the event queue.

Case 2: The max-curve of *O′* switches the distance order with that of *r.object* some time after *t′* and before *r.time*. In this case, *r.object* does not change but *r.time* decreases. This case is illustrated by Figure 5.1, in which *r.time* decreases from a farther time to *t″* due to the addition of *O′*. In this case, the intersection events in the event queue that will occur after the new *r.time*, such as intersection *p* in Figure 5.1, are no longer valid. However, we leave them in the event queue. They will be eliminated by the max 1-level event triggered at *r.time*.

Case 3: The max-curve of *O′* is currently closer to O_q than that of *r.object* and therefore *r.object* changes. In this case, eliminate the existing event queue; invoke the procedure Max1LevelUpdate(*r*, *t′*) to compute and schedule new intersection events.

The complexity of the addition is dominated by Case 3 which is O(*n*).

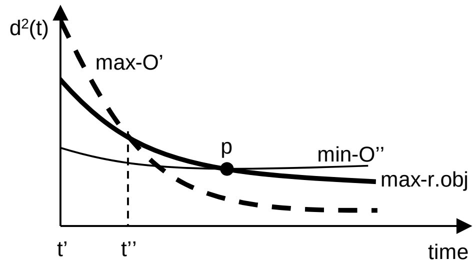


Figure 5.1. Case 2 of addition and Case 2 of deletion.

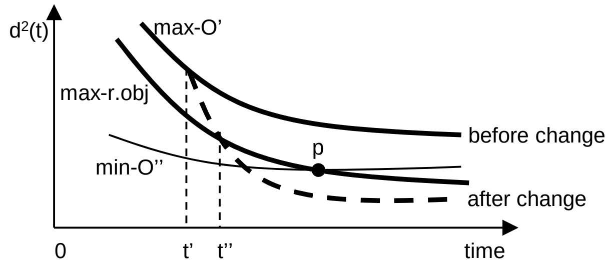


Figure 5.2. *r.time* may change due to the velocity-vector change of an object other than *r.object*.

Deletion. Consider the deletion of an object *O′* at a time point *t′*. The first step is to delete *O′* from *MOH* in O(log*n*) time as described in section 4.2.4. Depending on how the root node *r* changes as a result of the deletion, there are three cases for the second step.

Case 1: The max-curve of *O′* is always farther from O_q than that of *r.object* between *t′* and *r.time*. In this case, neither *r.object* nor *r.time* changes. To process this case, remove the critical intersections incurred by *O′* if any from the event queue.

Case 2: The max-curve of *O′* switches the distance order with that of *r.object* some time after *t′* and before *r*.time. In this case, *r.object* does not change but *r.time* increases. This case is illustrated by Figure 5.1, in which *r.time* increases from *t″* to a farther time due to the deletion of *O′*. In this case, some new intersection events may need to be scheduled, such as intersection *p* in Figure 5.1. Observe that there may be O(*n*) such intersection events. Inserting them to the existing event queue takes O(*n*log*n*) time. On the other hand, eliminating the existing event queue and reconstructing a new one takes only O(*n*) time. Thus, in Case 2 we eliminate the existing event queue; invoke the procedure Max1LevelUpdate(*r*, *t′*) to compute and schedule new intersection events.

Case 3: *O′* is currently *r.object* and therefore *r.object* changes. In this case, eliminate the existing event queue; invoke the procedure Max1LevelUpdate(*r*, *t′*) to compute and schedule new intersection events.

The complexity of the deletion is dominated by Case 2 and Case 3 which is O(*n*) each. Thus the complexity of the deletion is O(*n*).

Velocity-vector Change. Consider that an object *O′* changes its velocity-vector at time *t′*. The first step is to perform velocity-vector update to the object heap as described in section 4.2.4. Depending on how the root node *r* changes as a result of the update to the object heap, there are three cases for the second step:

Case 1: Both old(*O′*) and new(*O′*) are always farther from O_q than the max-curve of *r.object* between *t′* and *r.time*, where old(*O′*) and new(*O′*) represent the max-curves of *O′* before and after the velocity-vector change, respectively. In this case, neither *r.object* nor *r.time* changes. To process this case, compute the intersections between the new min-curve of *O′* and the max-curve of *r.object* that will occur between *t′* and *r.time*. Create an intersection event for each of these intersections (at most two) and insert it to the event queue.

Case 2: *r.object* does not change but *r.time* decreases. This case may happen when *O′* is *r.object*, i.e., it is *r.object* that changes its velocity-vector. Case 2 may also happen when *O′* is not *r.object*, as illustrated in Figure 5.2. In the figure, *r.time* decreases from a farther time to *t″* due to the velocity-vector change of *O′* which is not *r.object*. No further processing is needed in this case.

Case 3: *r.object* does not change but *r.time* increases. Again this case may happen when either *r.object* or some other object changes velocity-vector. In this case, eliminate the existing event queue and invoke the procedure Max1LevelUpdate(*r*, *t′*).

The complexity of processing each velocity-vector update is dominated by Case 3 which is O(*n*).

THEOREM 5.1. *Assume that each object can change its velocity at most m times. Then the cumulative complexity of the on-line algorithm for the uncertain case is* $O(n^2 m\alpha(n)\log n)$.

PROOF. There are four types of processing involved in the on-line algorithm, i.e., the processing of velocity-vector updates, the processing of intersection events, the processing of max 1-level events, and the processing of implicit updates to the object heap.

Processing velocity-vector updates. Since processing each velocity-vector update takes O(*n*) time, the cumulative complexity for processing velocity-vector updates is $O(n^2m)$.

Processing intersection events. According to Theorem 2.4, there are $O(nm\alpha(n))$ parabola pieces in the max 1-level. Each min-curve has O(*m*) parabola pieces. According to Lemma 3.5, there are $O(nm\alpha(n))$ intersections between each min-curve and the max 1-level. Thus there are totally $O(n^2m\alpha(n))$ intersections between *n* min-curves and the max 1-level. Processing each intersection event takes O(log*n*) time to remove from the event queue. Processing each intersection event takes O(log$n$) time. Thus the cumulative complexity of processing intersection events is O($n^2 m\alpha(n)$log$n$).

Processing max 1-level events. Each implicit update to the object heap triggers a max 1-level event. According to the analysis given in section 4.2.4, there are O($nm\alpha(n)$log$n$) implicit updates to the object heap and thus there are O($nm\alpha(n)$log$n$) max 1-level events. Processing each max 1-level event takes O(n) time. Thus the cumulative complexity of processing max 1-level events is O($n^2 m\alpha(n)$log$n$).

Processing implicit updates to the object heap. The cumulative complexity for processing implicit updates is $O(nm\alpha(n)\log^2 n)$ as discussed in section 4.2.4.

Thus the cumulative complexity of our on-line algorithm in the piecewise linear model is $O(n^2 m\alpha(n)\log n)$.□

Appendix E: Figures for Section 6.3

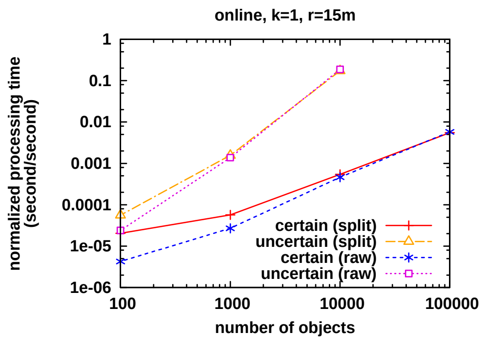


Figure 6.16. Normalized processing time versus number of objects

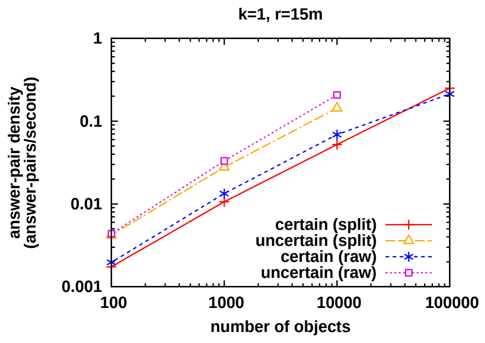


Figure 6.17. Answer-pair density versus number of objects

Appendix F: Applicability of Proposition 3.1 when there is location uncertainty associated with the query object

When there is location uncertainty associated with the query object O_q, Proposition 3.1 does not hold anymore. Specifically, O_i's min-curve being below the k-th level at a time t is only a necessary, but not a sufficient condition, for O_i to be a PkNN at t. This is because the distances of different objects to the query object are not independent to each other; they depend on which location within the query object's uncertainty region is used to compute the distance. This concept is illustrated by the following simple example. Figure C shows the uncertainty regions of the query object O_q and the only two moving objects O_a and O_b, at a time t. The radius is r for all the uncertainty regions. According to the figure, the maximum possible distance between $C_a(t)$ and $C_q(t)$ is 4r. The minimum possible distance between $C_b(t)$ and $C_q(t)$ is smaller than $4r$. Thus O_b's min-curve is below O_a's max-curve at time t. If we apply Proposition 3.1, we would conclude that O_b is a P1NN at time t, but this is obviously wrong. The reason is that the query object O_q can be at only one location at a time. Specifically, in order for O_q to get the maximum distance to O_a, O_q has to take the location at P in Figure C. On the other hand, in order for O_q to get the minimum distance to O_b, O_q has to take the location at P'. But O_q can only take one location at time t.

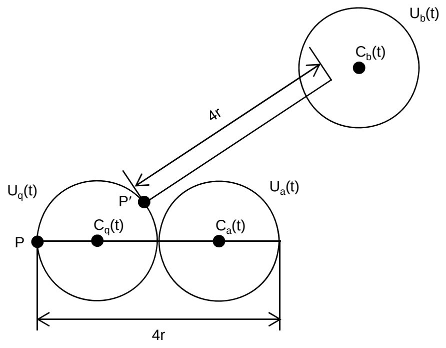


Figure C. Proposition 3.1 does not hold when there is location uncertainty associated with the query object.

Appendix G: Comparison between Certain Case and Uncertain Case

We focused on on-line processing. For on-line processing, every GPS point was treated as a velocity-vector update. The query processing stopped at the 3600th simulated second which is the end time of the GPS trace for all moving objects. Observe that the processing time may end before or after the simulated time. For example, if the average time it takes to process an update is bigger than the average inter-arrival time for updates, then the query processing time will be larger than the simulated time, otherwise it will be smaller. Figures 6.12 to 6.14 show the results for the query processing time. The data point for N=1,000,000 for the uncertain case in Figure 6.12 is not attainable due to a prohibitively long computation time. On the other hand, on-line processing easily scales to N=1,000,000. Furthermore, the difference between the two cases increases with the number of objects. Particularly, the processing time of the uncertain case grows much faster than that of the certain case as the number of objects grows (see

Figure 6.12). This phenomenon matches the analytical results (recall the O(n) factor of difference according to Table 1.1).

One fact that is a bit surprising is that the query processing time of the uncertain case does not change with the uncertainty region radius (see Figure 6.14). This is surprising because Figure 6.8 shows that the number of answer-pairs increases with the uncertainty region radius and thus presumably the query processing time should also increase with the uncertainty region radius. The reason for the phenomena shown in Figure 6.14 is as follows. Recall the proof of Theorem 5.1 in section 5.2.2. The cost of query processing for the uncertain case is dominated by three components: processing intersection events, processing max 1-level events, and processing implicit updates to the object heap. Observe that when all the other parameters are fixed and only the uncertainty region radius changes, the cost of processing max 1-level events and processing implicit updates to the object heap are fixed. The cost of processing max 1-level events is fixed because the algorithm always computes the critical intersections between the max 1-level and the min-curves of all the other N−2 objects. The cost of processing implicit updates to the object heap is fixed because the change of the max 1-level is not affected by the uncertainty region radius. The only component that is affected by the uncertainty region radius is processing intersection events. However, the cost of this component is negligible as shown in Table 6.3. Table 6.3 also shows that the most time-consuming part is processing max 1-level events. In this part, time is consumed for computing intersections between the max 1-level and the min-curves.

Table 6.4 shows the processing time distribution for the certain case when N=1,000,000 and k=1. From the figure it can be seen that most of the query processing time is spent on handling velocity-vector updates. This is true for other values of N and k that we tested.

Table 6.3. Processing time distribution for the uncertain case when N=10000, k=1, and r=15m.

Processing component	Processing intersection events	Processing max 1-level events	Processing implicit updates to object heap
Processing time (second)	0.1	621	0.2

Table 6.4. Processing time distribution for the certain case when N=1000000 and k=1.

Processing component	Initialization	Processing implicit updates	Processing velocity-vector updates
Processing time (second)	4	15	190

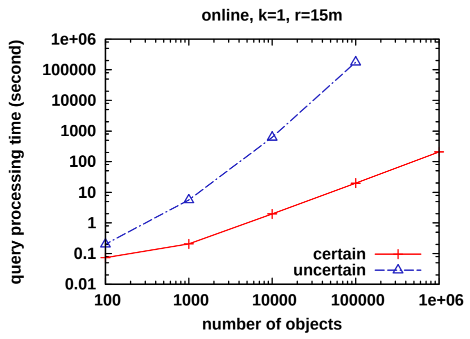


Figure 6.12. Query processing time versus number of objects

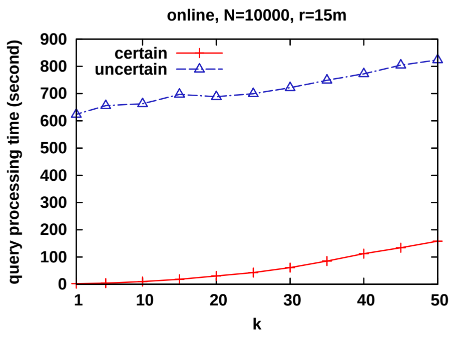


Figure 6.13. Query processing time versus k

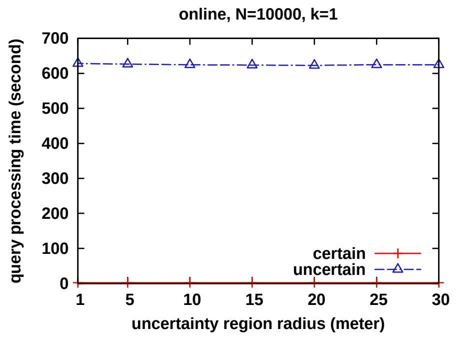


Figure 6.14. Query processing time versus uncertainty region radius