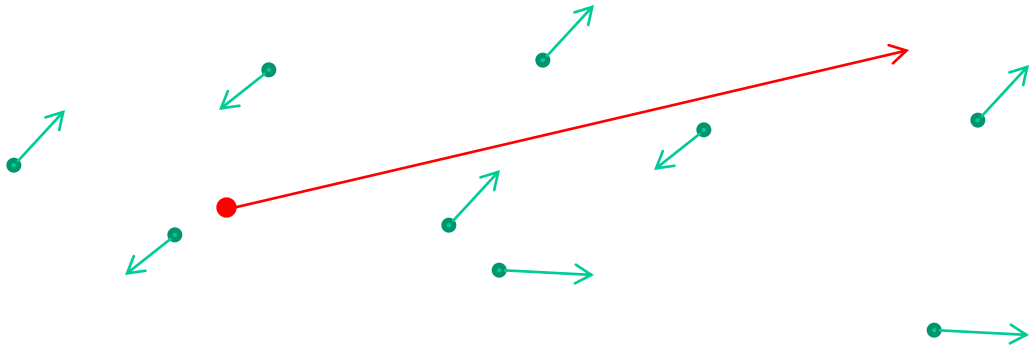
3.3 Methoden für kontinuierliche Anfragen (Continuous (Monitoring) Queries)

- Eine kontinuierliche Anfrage gibt zu einem gegebenen räumlichen Anfrageprädikat P die entsprechenden Ergebnisse für alle Zeitpunkte innerhalb eines (oder mehrere) gültige(n) Zeitbereiche(s)
- Varianten:
  1) Trajektorien aller beweglichen Objekte sind bekannt (continuous queries)
  2) Objekte bewegen sich frei im Raum (continuous monitoring Queries)

3.3.1 Kontinuierliche Anfragen (mit bekannten Trajektorien)

- Annahme: Trajektorien der Objekte sind bekannt (zumind. bis zu einem gegebenen gültigen Zeithorizont)
- Ziel: Ermittlung von zusammenhängenden Trajektorien/Zeit-Abschnitte für die gilt:
  - Konsistenz (bzgl. des Anfrageprädikates)
  - Vollständigkeit (Maximalität)
- Konsistenz:
  - Ein Trajektorien/Zeit-Abschnitt heißt *konsistent* bzgl. eines (räumlichen) Anfrageprädikates, wenn für je zwei Punkte in dem Abschnitt das entsprechende Anfrageergebnis gleich ist.
- Vollständigkeit:
  - Ein konsistenter Trajektorien/Zeit-Abschnitt T heißt *vollständig*, wenn kein weiterer konsistenter Trajektorien/Zeit-Abschnitt T'$\neq$ T existiert, der T vollständig enthält.

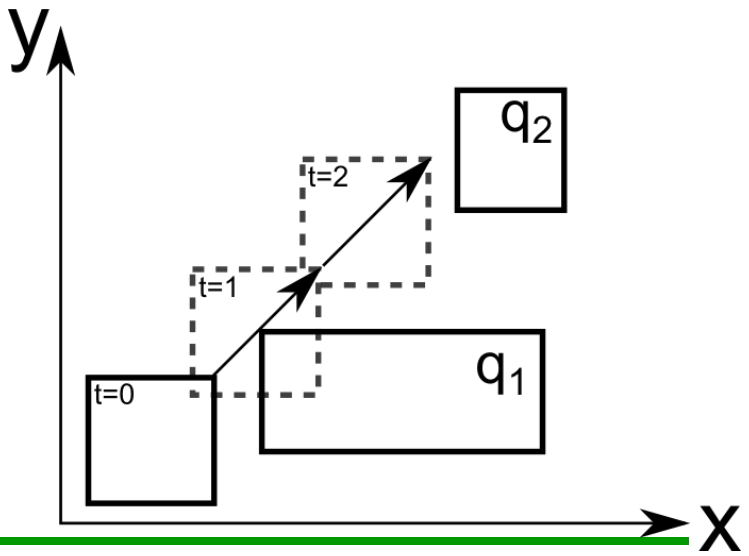
- Im Folgenden werden zwei Ansätze diskutiert:
  - Zeit-parametrisierte Fenster- und NN-Anfragen (TP-Queries) [TP02]
  - Kontinuierliche NN-Anfragen (CNN) [TPS02]
- Annahmen:
  - Anfrageobjekt (und evtl. Datenbankobjekte) bewegen sich kontinuierlich.
  - Anfrageauswertung bezieht sich auf die Gegenwart (t_0) und Zukunft bzw. Vergangenheit.
  - (Zukünftige/historische) Bewegungstrajektorien sind bekannt.



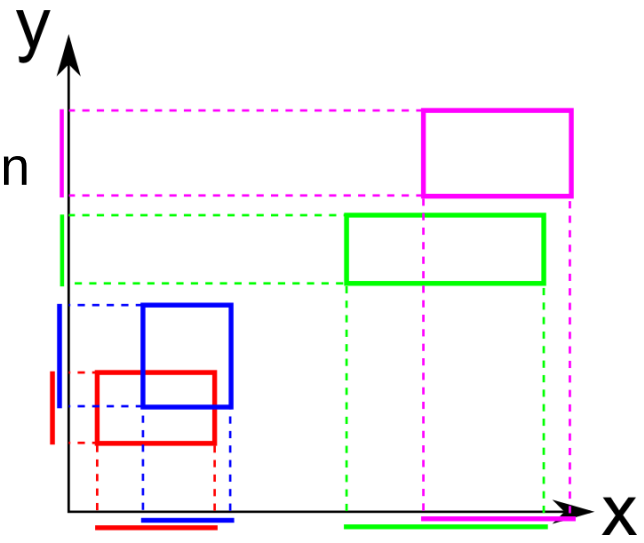
- Allgemeines zu Zeit-Parametrisierten Anfragen in ST-DBS
  - Eine Zeit-Parametrisierte Anfrage gibt folgende Antwort zurück:
    - Objekte, die das Anfrageprädikat zum Zeitpunkt t_0 (Gegenwart) erfüllen.
    - Zeitpunkt t_1 bis zu dem das (aktuelle) Anfrageergebnis gültig bleibt.
    - Änderungen der Ergebnissmenge ab Zeitpunkt t_1.
  - Weitere Annahmen:
    - Unterstützung der Anfrage durch Verwendung des TPR-Baums zur Indexierung der Datenbankobjekte.
    - Anfrage beschränkt auf den Zeithorizont für den die gegebenen Objekttrajektorien (inkl. Anfrageobjekttrajektorie) gültig sind.
  - Beispielanfrage:
    - Ein Reisender fragt alle Hotels im Umkreis von 5km zur aktuellen Position an. Zusätzlich zur aktuellen Ergebnismenge (z.B. A,B,C), die aktuell in der Nähe des Reisenden liegt, enthält das Ergebnis die Zeit (z.B. 1 min), bis zu der die Antwort aktuell ist (gegeben die Richtung und Geschwindigkeit des Reisenden), und zusätzlich die neue Ergebnismenge, die sich aus der veränderten Position ergibt.

– Zeit-Parametrisierte Fensteranfrage (TP-Window Query)

- Annahme: [TP02]
  - Query-Fenster q und Objekte o bewegen sich linear
  - Ggf. lineare Skalierung von q und o (definierende Eckpunkte bewegen sich unterschiedlich schnell)
- *Influence Time* $T_{INF}(o,q)$: Zeit, zu der Objekt o die Query q beeinflusst (o wird Teil des Ergebnisses oder ist nicht mehr Teil des Ergebnisses)
  - Im Beispiel: Ergebnis zur Zeit t_0: Ø
  - Ergebnis ändert sich zur Zeit t=0.5 zu $\{q_1\}$ =>$T_{INF}(o,q)=0.5$
- Ergebnis verfällt zur Zeit $\min_{o \in DB}\{T_{INF}(o,q)\}$
- Es müssen also die o mit minimaler $T_{INF}(o,q)$ gefunden werden
- $T_{INF}(o,q)$ kann über Zeitintervall berechnet werden, in dem sich o und q schneiden, bzw. nicht mehr schneiden



- Schnitt-Berechnung von MBRs:
  Zwei MBRs schneiden sich,
  wenn sie sich bzgl. aller Dimensionen schneiden
- MBR eines Objektes:
  $\{(o_{1L}, o_{1R}), \ldots, (o_{nL}, o_{nR})\}$
- Geschwindigkeit eines Objektes:
  $\{(o.V_{1L}, o.V_{1R}), \ldots, (o.V_{nL}, o.V_{nR})\}$



- Wann schneiden sich bewegliche MBRs?
  - Schnitt beginnt, wenn sich die Untergrenze des einen MBRs mit der Obergrenze des anderen MBRs trifft (und sich alle anderen Dimensionen bereits schneiden)
  - Schnitt endet, wenn sich die Obergrenze des einen MBRs mit der Untergrenze des anderen MBRs trifft (alle anderen Dimensionen können sich weiter schneiden)
  - Damit muss für zwei Rechtecke o und q für jede Dimension i lediglich berechnet werden:
    - » $T_{iLR} = (o_{iL}-q_{iR})/(q.V_{iR}-o.V_{iL})$
    - » $T_{IRL} = (o_{iR}-q_{iL})/(q.V_{iL}-o.V_{iR})$

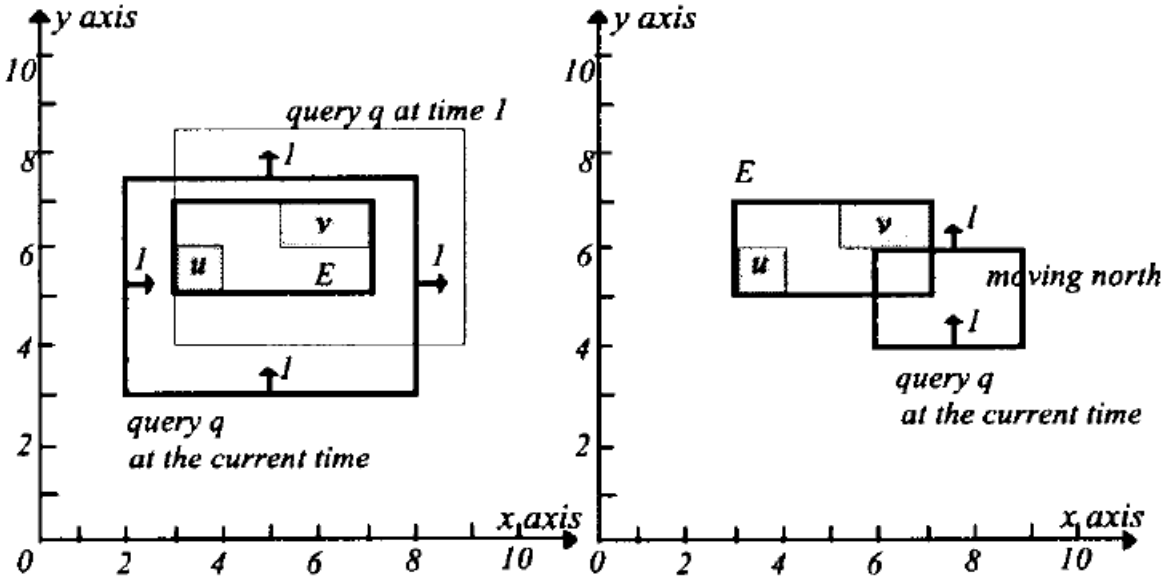
- Über Rechteckschnitt wird T_{INF} von Schlüsseln innerer Knoten sowie Objekten berechnet
- Branch-and-Bound-Suche der entsprechenden Objekte
- T_{INF} von Objekten:

Compute_Intersection_Period (o,q)

$[T_s, T_e[= [0,\infty[$

For each dimension i	Betrachte den Schnitt bei jeder Dimension einzeln
$T_{iLR} = (o_{iL}-q_{iR})/(q.V_{iR}-o.V_{iL})$	Zeitpunkt, zu dem der rechte Punkt von q den linken von o passiert
if $T_{iLR} < 0$ then $T_{iLR} = \infty$	Die Punkte treffen sich nie, falls $T_{iLR} < 0$
$T_{IRL} = (o_{iR}-q_{iL})/(q.V_{iL}-o.V_{iR})$	Zeitpunkt, zu dem der rechte Punkt von o den linken von q passiert
if $T_{iRL} < 0$ then $T_{iRL} = 0$	Die Punkte treffen sich nie, falls $T_{iRL} < 0$
if $[o_{iL}, o_{iR}]$ does not intersect $[q_{iL}, q_{iR}]$	Zu t=0 schneiden sich de beiden Intervalle noch nicht, dann:
$T_{is} = \min(T_{iLR}, T_{iRL})$	Berechne Beginn …
$T_{ie}=\max(T_{iLR},T_{iRL})$	… und Ende der Intervall-Überschneidung.
else	Punkte schneiden sich zu t = 0 (aktueller Zeitpunkt)
$T_{is} = 0$	Startpunkt ist dann 0, und …
$T_{ie} = \min(T_{iLR}, T_{iRL})$	… Endzeitpunkt ist das Minimum der zwei Überschneidungen
$[T_s, T_e[= [T_s, T_e[\cap[T_{is}, T_{ie}[$	Schneide die Intervallschnittzeiten aller Dimensionen, dadurch berechnet sich das Zeitintervall, in dem das Objekt o die Query q schneidet.
return $[T_s, T_e[$	T_{INF} ist T_S, falls keine Überschneidung zu t=0 und T_E sonst

- T_{INF} des Schlüssels eines inneren Knotens E:
  - Ziel: Pruning von Teilbäumen
  - Abschätzung des Minimums der T_{INF} der Kinder (~ MINDIST)
  - Zeit, zu der ein Intermediate Entry E das Ergebnis beeinflussen kann:
    - » Aktuell kein Schnitt mit q: Minimale Zeit, zu der E q zu schneiden beginnt
    - » E schneidet q teilweise: keine Aussage möglich, absteigen notwendig
    - » E ist in q enthalten: Frühestens zu der Zeit T_{PI}, zu der E q nur noch partiell schneidet
    - » Unterschied Schlüssel eines inneren Knotens ↔ Objekt: Schneidet ein Schlüssel eines inneren Knotens das Anfragefenster teilweise, kann sich das Ergebnis ändern, bei Objekten nicht!



- Berechnung von TPI:
  - Ähnlich zur Berechnung von TINF
  - Dieses mal müssen nicht entgegengesetzte Intervallenden von E und q verglichen werden, sondern die gleichen
  - Wieder Reduktion auf einzelne Dimensionen möglich
  - Relevant ist die Dimension, bei der E zuerst nicht mehr in q enthalten ist

Compute_T_{PI}(E,q, $[T_s,T_e[$)

$T_{PI} = \infty$

For each dimension i

$T_{iLL} = (E_{iL}-q_{iL})/(q.V_{iL}-E.V_{iL})$

$T_{iRR} = (E_{iR}-q_{iR})/(q.V_{iR}-E.V_{iR})$

if $T_{iLL} \in [T_s,T_e[$ and $T_{iRR} \in [T_s,T_e[$

$T_{iPi} = \min(T_{iLL}, T_{iRR})$

if $T_{iLL} \in [T_s,T_e[$ and $T_{iRR} \notin [T_s,T_e[$

$T_{iPi} = T_{iLL}$

if $T_{iLL} \notin [T_s,T_e[$ and $T_{iRR} \in [T_s,T_e[$

$T_{iPi} = T_{iRR}$

if $T_{iLL} \notin [T_s,T_e[$ and $T_{iRR} \notin [T_s,T_e[$

$T_{iPi} = \infty$

$T_{PI} = \min(T_{PI}, T_{iPi})$

Return T_{PI}

T_{PI}: Zeit, zu der E nicht mehr vollständig in q enthalten ist
Betrachte wieder jede Dimension einzeln
T_{iLL}: Zeit, zu der der linke Punkt von q den linken Punkt von E trifft
T_{iRR}: Zeit, zu der der rechte Punkt von q den rechten Punkt von E trifft
Es ist immer der kleinere Zeitpunkt relevant. Liegen beide im Zeitintervall, zu dem sich E und q schneiden, wird das Minimum gewählt.

Liegt nur T_{iLL} im Zeitintervall, wird T_{iLL} gewählt

Liegt nur T_{iRR} im Zeitintervall, wird T_{iRR} gewählt

Liegt keiner im Zeitintervall, verlässt E niemals q, und T_{iPi} wird auf unendlich gesetzt
Interessant ist das Minimum über alle Dimensionen, weil eine Dimension ausreicht, damit E nicht mehr vollständig in q enthalten ist

– Zeit-Parametrisierte NN-Anfrage (TP-NN Query) [TP02]

- Ziel: Minimale Zeit berechnen, zu der o näher an q ist als der aktuelle Nächste Nachbar P_{NN}
- Der Einfachheit halber werden nur Punkte betrachtet
- Berechnung von $T_{INF}(o,q)$ für Objekte
  - $T_{INF}(o,q)$: Zeit zu der $dist(o(t),q(t)) \leq dist(P_{NN}(t)q(t))$ und $t > 0$
  - Umformung in $At^2+Bt+C \leq 0$ mit (Euklidische Distanz):

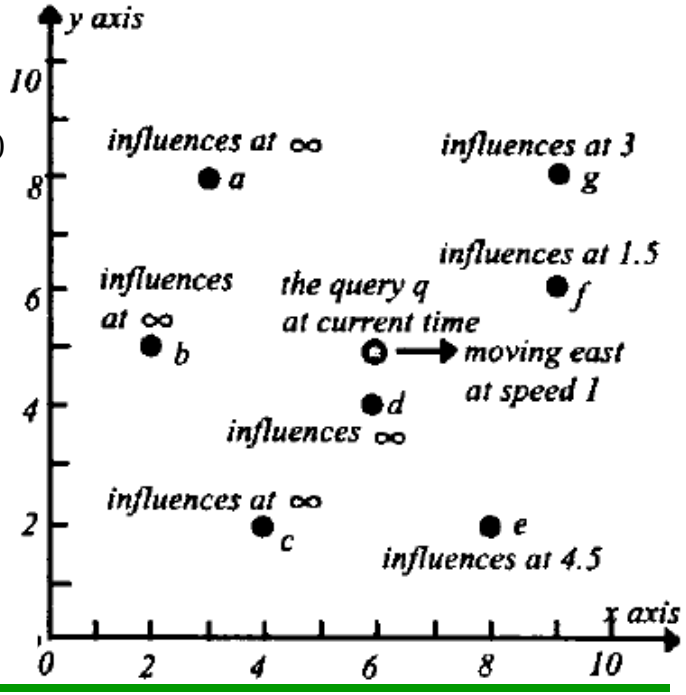
$$A = \sum_{i=1}^{n} \left[(o.V_i - q.V_i)^2 - (P_{NN}.V_i - q.V_i)^2 \right]$$

$$B = \sum_{i=1}^{n} 2\left[(o_i - q_i)(o.V_i - q.V_i) - (P_{NN_i} - q_i)(P_{NN}.V_i - q.V_i) \right]$$

$$C = \sum_{i=1}^{n} \left[(o_i - q_i)^2 - (P_{NN_i} - q_i)^2 \right]$$

» Für kein t Lösung der Ungleichung: setzte $T_{INF} = \infty$

» Sonst: Lösung der Ungleichung ist $T_{INF}(o,q)$



- Berechnung von $T_{INF}(o,q)$ für Schlüssel innerer Knoten
  - Konservative Abschätzung möglich über MINDIST, dann Berechnungen aber sehr komplex (Fallunterscheidungen in MINDIST)
  - Stattdessen: MINDIST unterschätzen, um korrektes Ergebnis zu ermöglichen => orthogonale Distanz zu einer ausgewählten Kante/Seitenfläche des MBRs
    - » Fall 1: MINDIST würde über Eckpunkt des MBRs berechnet werden: Gewählte Kante ist die vom MBR am weitesten entfernte Kante, die mit dem Eckpunkt verbunden ist
    - » Fall 2: MINDIST würde über Kante berechnet werden: Wähle diese Kante
  - Sei l die gewählte Kante, dann kann wiederum die Ungleichung aufgestellt werden:

$$MINDIST(l(t),q(t)) \le dist(P_{NN}(t)q(t)) \Leftrightarrow$$

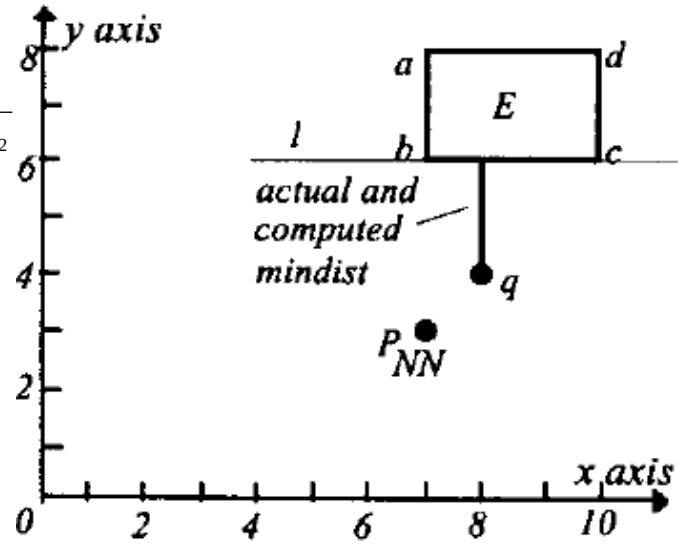
$$|(q_i - l_i) + t(q.V_i - l.V_i)| \le \sqrt{\sum_{i=1}^{n}[(PNN_i - q_i) + t(PNN.V_i - q.V_i)]^2}$$

  - Umformung ergibt eine Gleichung der Form $At^2+Bt+C \le 0$ mit

$$A = (l.V_i - q.V_i)^2 - \sum_{i=1}^{n}[P_{NN}.V_i - q.V_i]^2$$

$$B = 2(o_i - q_i)(o.V_i - q.V_i) - \sum_{i=1}^{n} 2(P_{NN\,i} - q_i)(P_{NN}.V_i - q.V_i)$$

$$C = (l_i - q_i)^2 - \sum_{i=1}^{n}[(P_{NN\,i} - q_i)^2]$$



– Kontinuierliche Nächste-Nachbarn-Anfrage (CNN Query)

- Gegeben: [TPS02]
  - Menge *P* von statischen Objekten (Datenbankobjekte)
  - Trajektorie q eines beweglichen Anfrageobjektes als Liniensegment q=[s,e]
- Gesucht:
  - Zerlegung von q in vollständige und konsistente Teilsegmente $t_i = [s_i, s_{i+1}]$, so dass:

    $s_i, s_{i+1} \in q$

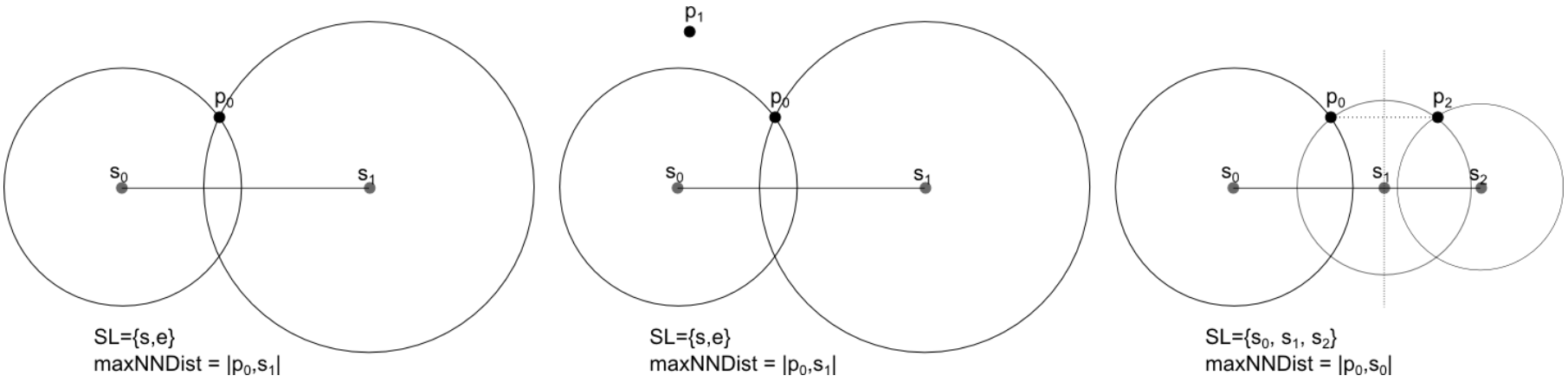
    $\forall p_1, p_2 \in [s_i, s_{i+1}]: \exists n_1 \in NN(p_1), \exists n_2 \in NN(p_2): n_1 = n_2$

    wobei $s_0 = s$ und $NN(p_i)$ = nächster Nachbar von p_i (deterministisch)

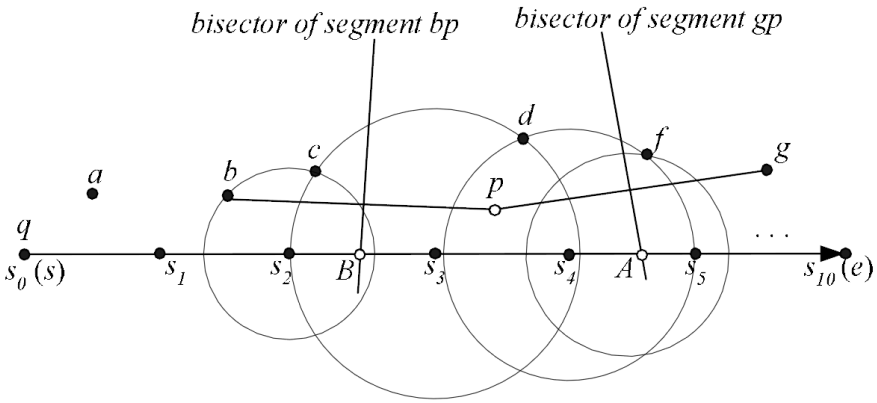
    Bem.: Konsistenz der Teilsegmente bezieht sich auf eine nicht-deterministische NN-Suche (Konsistenz bzgl. der offenen Intervalle).
  - Ausgabe von Tupeln $(t_i, NN(s_i))$ aller Teilsegmente t_i mit entsprechenden Anfrageergebnis $NN(s_i)$.
- Beispiel-Anfrage:

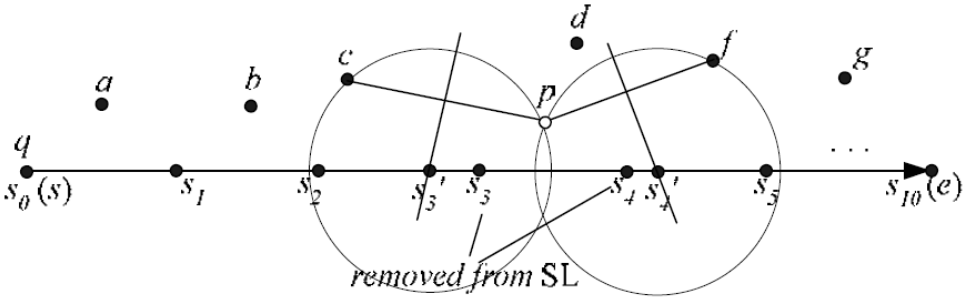
  „ Auf meiner Fahrt von Siegen nach Erfurt, suche mir alle nächstgelegenen Tankstellen“

- Idee:
  - SL: Liste aller Segment-Endpunkte von q=[s,e], sortiert nach Entfernung zu s.
  - Initialisierung von SL mit Start- und Endpunkt: SL_{init} = [s,e],
  - NN(s) = NN(e) = p_0 (p_0 beliebig)
  - Fällt ein weiterer Punkt p_i in die NN-Distanz dist(NN(s_i),s_i) eines Segment-Endpunktes $s_i \in SL$, wurde ein neuer NN für Punkte auf q gefunden
  - In diesem Fall berechne das Äquidistanzpotential AD zwischen NN(s_i) und p_i sowie den Schnittpunkt sp zwischen AD und q.
  - Füge sp in SL ein, setzte NN(sp) = {NN(s_i), p_i}



- Kontinuität
  - Sei S_{COVER}={sl ∈ SL| dist(sl, NN(sl)) > dist(sl,p)}
  - Kontinuität: alle $s_i \in S_{COVER}$ sind benachbart: S_{COVER} ={s_i,…, s_{i+k}}
  - Ermöglicht binäre Suche eins $s_i \in S_{COVER}$ (linkes Bild)
  - Weitere $s_j \in S_{COVER}$ sind immer Nachbarn von s_i. (Performanz!)
  - Ermöglicht einfaches Update von SL:
    - *u=NN(s_{i-1})∩NN(S_i); v=NN(s_{i+k}) ∩NN(S_{i+k+1});*
    - SL = SL \ S_{COVER}
    - $s_{i'}$ *:neuer Segmentpunkt am Schnitt zwischen q und ⊥(u, p)* mit NN($s_{i'}$) =*p*
    - $s_{i+1'}$: *neuer Segmentpunkt am Schnitt zwischen q und⊥(v, p) mit NN($s_{i+1'}$)=p*





- Algorithmus (Depth-First):

SL_0 =[s,e] //Splitpunkte; SL sei sortiert nach aufsteigender Distanz zu s
NN(e) = NN(s) = Ø //Aktuelle NN‘s sind noch nicht bekannt

Function CNN-depth(R-Tree-Node N, SL, &maxNNdist(SL)): //maxNNdist(SL) ist eine Referenz
Falls N is Leaf Node:
Für alle p in N:
S_{COVER}: {sl ∈ SL| dist(sl, NN(sl)) > dist(sl,p)} // S_{COVER} ={s_i,…, s_{i+k}};
u=NN(s_{i-1})∩NN(S_i); v=NN(s_{i+k}) ∩NN(S_{i+k+1});
SL = SL \ S_{COVER}
Füge $s_{i'}$:neuer Segmentpunkt am Schnitt zwischen q und ⊥(u, p) mit NN($s_{i'}$) =p in SL ein
Füge $s_{i+1'}$: neuer Segmentpunkt am Schnitt zwischen q und⊥(v, p) mit NN($s_{i+1'}$)=p in SL ein
update maxNNdist(SL) falls notwendig
Falls N Directory Node:
Für alle p in N sortiert nach MINDIST(p,q):
Falls MINDIST(p,q) > maxNNdist(SL): continue; //kann immer upgedated werden, deshalb billig
Falls ∄sl in SL: MINDIST(p,sl) < dist(sl, NN(sl)): continue;
SL = CNN-depth(p, SL, maxNNdist(SL))
return SL

Ergebnis: alle Tupel ([s_i, s_{i+1}], {NN(s_i)| NN(s_i)=NN(s_j)})

- Algorithmus (Best-First):

Function CNN-best(R-Tree-Node N, q=[s,e]):

Setzte SL_0= {s,e} //Splitpunkte; SL sei sortiert nach aufsteigender Distanz zu s

Setzte aktuelle NN‘s: NN(e) = NN(s) = Ø //Aktuelle NN‘s sind noch nicht bekannt

APL = [N]: LIST of R-Tree-Node ORDERED by ASCENDING MINDIST to q

maxNNdist(SL) = ∞

while APL ≠ Ø

Hole ersten Entry F aus APL

Falls F is Leaf Entry (=Datenpunkt):

S_{COVER}: {sl ∈ SL| dist(sl, NN(sl)) > dist(sl,p)} // S_{COVER} ={s_i,…, s_{i+k}}

u=NN(s_{i-1})∩NN(S_i); v=NN(s_{i+k}) ∩NN(S_{i+k+1});

SL = SL \ S_{COVER}

Füge $s_{i'}$:neuer Segmentpunkt am Schnitt zwischen q und ⊥(u, p) mit NN($s_{i'}$) =p in SL ein

Füge $s_{i+1'}$: neuer Segmentpunkt am Schnitt zwischen q und⊥(v, p) mit NN($s_{i+1'}$)=p in SL ein

update maxNNdist(SL) falls notwendig

Falls N Directory Node:

Für alle p in N:

Falls MINDIST(p,q) > maxNNdist(SL): prune p;

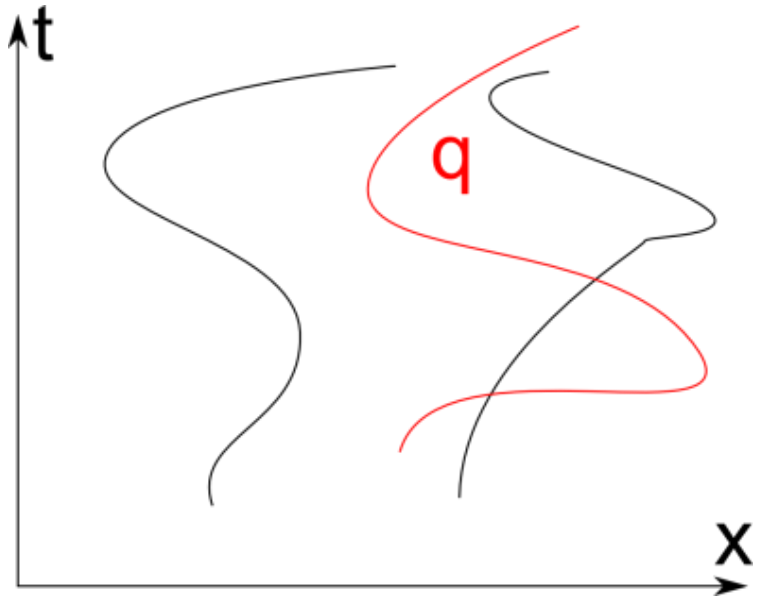
Falls ∄sl in SL: MINDIST(p,sl) < dist(sl, NN(sl)): prune p;

Füge p in APL ein

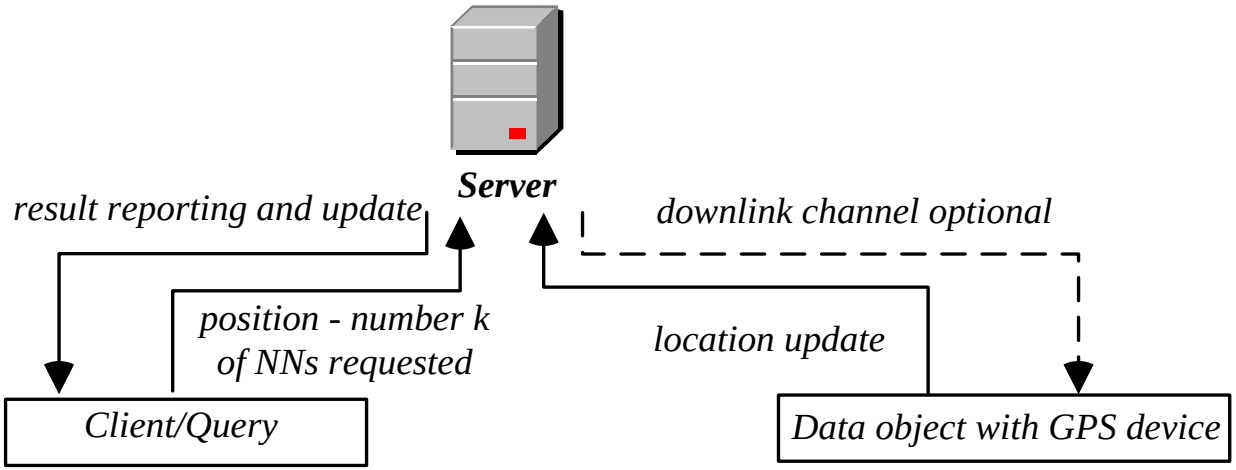
Ergebnis: alle Tupel ([s_i, s_{i+1}], {NN(s_i)| NN(s_i)=NN(s_j)})

3.3.1 Monitoring-Anfragen

- Query wird initialisiert und soll dann über einen längeren Zeitraum eine Ergebnismenge aktuell halten
- Oft Client/Server-basiert
- Oft keine Annahmen bzgl. der Beweglichkeit der Objekte (z.B. keine Annahme einer linearen Bewegung)

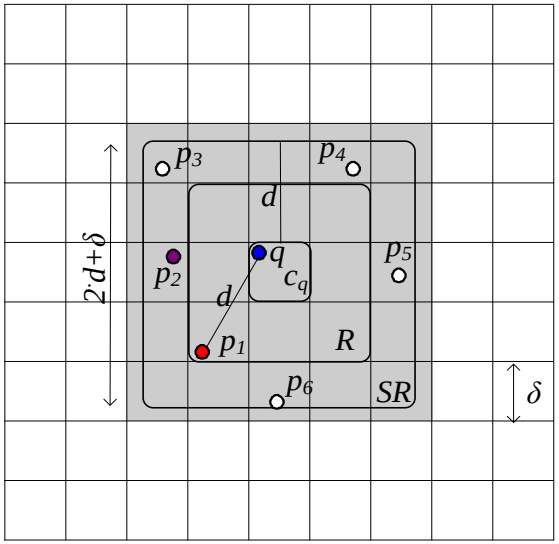


– Continuous NN Monitoring:
  - Annahmen:
    – Queries sowie Objekte in der Datenbank bewegen sich
    – Bewegung zufällig und damit unvorhersehbar
    – Zentraler Server überwacht die kNN jeder Query und sendet sie an einen Client
  - Ziele:
    – CPU-Overhead des Servers minimieren
    – Netzwerklast für Ortsupdates minimieren
  - Oft keine Verwendung komplexer Indices, da Updates meist sehr teuer

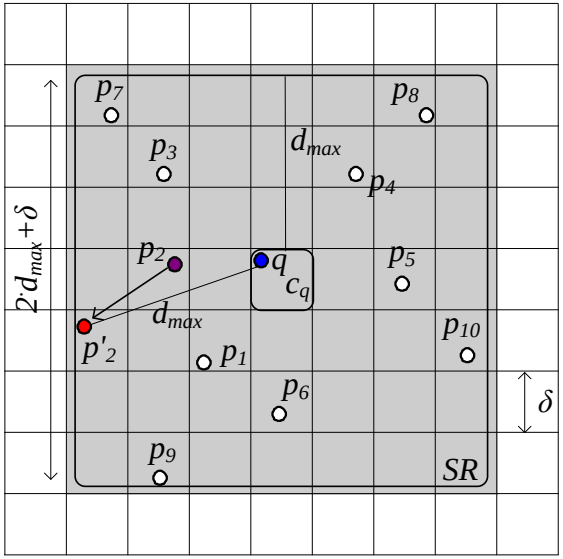


- YPK-CNN [YPK05]
  - Einfacher Grid-Index zur Überwachung der kNN einer oder mehrerer Queries $\{q_0, \ldots, q_n\}$
  - Querying the Now
  - Algorithmen für …
    - … wenige Queries: indexiere die Queries
    - … viele Queries: indexiere die Datenbank
    - … gleichverteilte Queries: berechne die kNN jede Runde neu
    - … unbekannte Verteilung der Queries: inkrementeller Ansatz
  - Hier: inkrementell, indexierte Datenbank, kNN für alle Queries, sequentiell

- Berechnung des initialen Ergebnisses:
  - Wandere so lange entlang einer rechteckigen Spirale um q, bis k Objekte gefunden wurden. Die Objekte befinden sich innerhalb einer Entfernung d von q.
  - Durchsuche alle Zellen im Query-Rechteck SR mit einer Seitenlänge von 2d+δ um ggf. false misses zu eliminieren
  - δ≥0 vergrößert SR, so dass jede Grid-Zelle ganz oder gar nicht in RS liegt

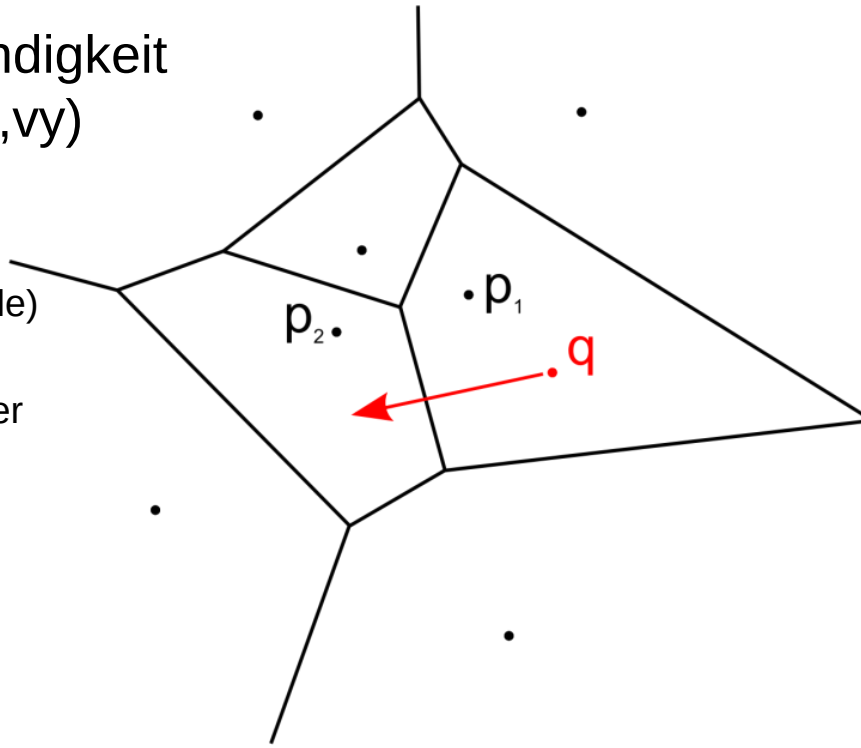


- Update des Ergebnisses:
  - d_{max} der letzten NN zu q berechnen
  - Suche die kNN in $2d_{max}$ +δ

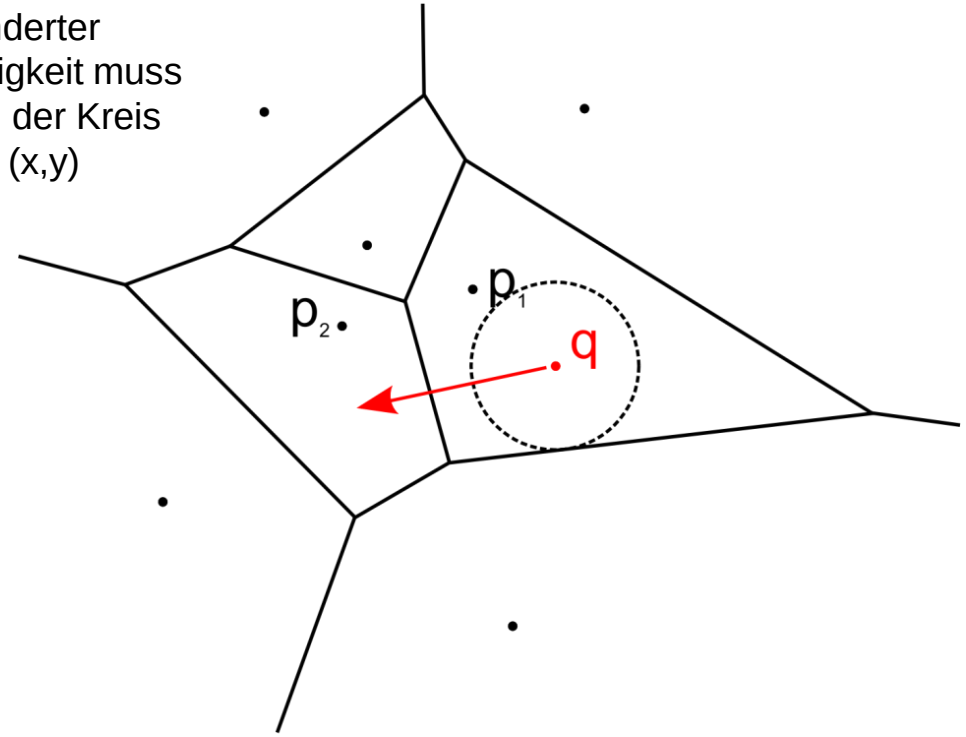


– NN-Queries mit Validitätsinformation [ZL01]

- Ziel: NN-Anfrage eines beweglichen Objektes mit der zusätzlichen Information, wie lange das Ergebnis gültig bleiben wird
- Querying the Presence/Future
- Voronoi-Diagramm der DB wird im Voraus berechnet
- Query q: Position und Geschwindigkeit des Query-Objektes: q = (x,y,vx,vy)
- Antwort: Tupel (p1, t, p2) mit
  - » p1: Punkt aus der DB, der aktuell q am nächten ist (über Voronoizelle)
  - » t: Validitätszeitraum
  - » p2: Punkt aus der DB, der nach der Zeit t q am nächtsten sein wird



- Problem: Validitätszeitraum erfordert Annahmen über das zukünftige Verhalten von Q
  - Lösung 1: nimm lineare Bewegung und konstante Geschwindigkeit an, sende Query erneut falls sich Richtung oder Geschwindigkeit ändert
  - Lösung 2:
    - » Berechne kürzeste Distanz d von q zu einer der Kanten der umgebenden Voronoizelle -> Minimale Distanz, die auch bei Richtungsänderung zurückgelegt werden muss, um die Voronoizelle zu verlassen
    - » Eine neue Query bei geänderter Richtung oder Geschwindigkeit muss erst gestellt werden, wenn der Kreis mit Radius d und Zentrum (x,y) verlassen wurde



[EGSV99]: M. Erwig, R. H. Güting, M. Schneider, M. Vizirgiannis. An Approach to Modeling and Querying Moving Objects Databases, In GeoInformatica 3(3), 1999.

[S99]: T. Sellis. Research Issues in Spatio-temporal Database Systems. In Proc. of SSD, 1999

[MGA03] M. F. Mokbel, T. M. Ghanem, W. G. Aref. Spatio-temporal Acces Methods. IEEE Data Engineering Bulletin, 2003.

[KGT99.pdf] G. Kollios, D. Gunopulos, V. J. Tsotras. On Indexing Mobile Objects. In Proc. of PODS, 1999.

[TP02]: Y. Tao, D. Papadias. Time-Parameterized Queries in Spatio-Temporal Databases. In Proc. of SIGMOD, 2002.

[TPS02]: Y. Tao, D. Papadias, Q. Shen. Continuous Nearest Neighbor Search. In Proc. of VLDB, 2002.

[ZL01] B. Zheng, D. L. Lee. Semantic Caching in Location-Dependent Query Processing. In Proc. of SSTD, 2001.

[PJT00]: D. Pfoser, C. S. Jensen, Y. Theodoridis. Novel Approaches to the Indexing of Moving Object Trajectories. In Proc. of VLDB, 2000.

[TFPL04]: Y. Tao, C. Faloutsos, D. Papadias, B. Liu. Prediction and Indexing of Moving Objects with Unknown Motion Patterns. In Proc. of SIGMOD, 2004.

[TPR00]: S. Saltenis, C. S. Jensen, S. T. Leutenegger, M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In Proc. of MOD, 2000.

[YPK05]:X. Yu, K. Q. Pu, N. Koudas. Monitoring k-Nearest Neighbor Queries Over Moving Objects. n.d.

[XHL90]: X. Xu, J. Han, and W. Lu. RT-Tree: An Improved R-Tree Indexing Structure for Temporal Spatial Databases. In Proc. of the Intl. Symp. on Spatial Data Handling, SDH, pages 1040–1049, July 1990.

[NS98]: M. A. Nascimento and J. R. O. Silva. Towards historical R-trees. In Proc. of the ACM Symp. on Applied Computing, SAC, pages 235–240, Feb. 1998.

[TP01]: Y. Tao and D. Papadias. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In Proc. of the Intl. Conf. on Very Large Data Bases, VLDB, pages 431–440, Sept. 2001.

Auf die Veröffentlichungen kann vom Uni-Netzwerk aus zugegriffen werden.