Scalable Management of Trajectories and Context Model Descriptions

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

vorgelegt von

Ralph Markus Lange

aus Böblingen

Hauptberichter:	Prof. Dr. rer. nat. Dr. h.c. Kurt Rothermel
Mitberichter:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Tag der mündlichen Prüfung:	29. November 2010

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart

2010

Acknowledgments

Thanks to the unfailing mercy of my Heavenly Father, it is now time to write the last lines of this thesis ...

I would like to thank my supervisor Prof. Dr. Kurt Rothermel, for inspiring and guiding me during the research described in this thesis and for giving me the opportunity to work in his group. Special thanks also go to Dr. Frank Dürr, most importantly for the many fruitful discussions. I have always been able to count on both of them to help transform my fledgling ideas into focused research endeavors as well as to improve the clarity and precision of my writing.

My sincere thanks are also given to Prof. Dr. Bernhard Mitschang for accepting the task of the second referee.

Furthermore, I would like to thank my colleagues in the Distributed Systems Department and within the Collaborative Research Center 627 for the many valuable and stimulating research discussions, the joint paper and proposal writing, and the pleasant working atmosphere. Among those people, a special mention deserve Robert Sauter, PD Dr. Wolfgang Blochinger, Dr. Matthias Gauger, Tobias Farrell, Harald Weinschrott, Lars Geiger, Marius Wernke, Prof. Dr. Christian Becker, Dr. Dominique Dudkowski, Dr. Susanne Bürklen, Dr. Gregor Schiele and Steffen Maier.

I also would like to give thanks to the Deutsche Forschungsgemeinschaft (DFG) for their generous funding within the Collaborative Research Center 627.

Last but not least, I wish to thank my wonderful wife Mirjam and my parents for her continuing support, encouragement, and prayers.

Contents

List of Figures

List of Tables

List of Abbreviations

AES	Advanced Encryption Standard
API	Application Programming Interface
CDR	Connection-Preserving Dead Reckoning
CPU	Central Processing Unit
DB	Database
DBMS	Database Management System
DHT	Distributed Hash Table
DOP	Dilution Of Precision
DP	Douglas-Peucker *(algorithm)*
DHT	Distributed Hash Table
DTI	Distributed Trajectory Index
GPS	Global Positioning System
GRTS	Generic Remote Real-Time Trajectory Simplification
GSAlg	Generic Split Algorithm
HIS	Heterogeneous Information System
HTTP	Hypertext Transfer Protocol
IEC	International Electrotechnical Commission
IP	Internet Protocol
IR	Infrared
ISO	International Organization for Standardization
KML	Keyhole Markup Language
KMZ	Zipped KML
LDR	Linear Dead Reckoning
LRC	Long-Range Contact
MBR	Minimum Bounding Rectangle
MOD	Moving Objects Database
NMEA	National Marine Electronics Association
OSM	OpenStreetMap

OWL	Web Ontology Language
PDMS	Peer Data Management System
RAM	Random-Access Memory
RCC	Region Connection Calculus
RDF	Resource Description Framework
RFID	Radio-Frequency Identification
SDC-Tree	Source Description Class Tree
SO	Shadow Object
SOL	Simple Ontology Language
SPARQL	SPARQL Protocol and RDF Query Language[1]
SQL	*(often)* Structured Query Language[2]
UDP	User Datagram Protocol
UTC	Coordinated Universal Time
WKT	Well-Known Text
WLAN	Wireless Local Area Network

[1] A recursive acronym.
[2] According to the SQL standard [ISO-9075], SQL is a name on its own.

Abstract

Context-awareness refers to the idea that applications adapt to their context of use including, for example, location, nearby devices and user habits. In the last years, billions of sensors have been deployed all over the globe, which allow creating comprehensive context models of our physical environment. The availability of such models constitutes a huge potential for context-aware computing as it allows selecting relevant context information from different providers all over the globe. However, such sharing of context information poses a number of challenges. A fundamental problem is how to provide efficient access to the immense amounts of distributed dynamic context information – particularly due to the mobility of devices and other entities.

To enable efficient access to current and past position information about moving objects, we propose a family of trajectory tracking protocols (CDR, GRTS) as well as a distributed indexing scheme (DTI) for trajectories. Given a certain accuracy bound, CDR and GRTS optimize the storage consumption and communication cost for tracking a moving object's trajectory in real-time at some remote database and allow for various trade-offs between computational costs, reduction efficiency, and communication. DTI enables efficient access to trajectory information that is partitioned in space and stored by different servers for scalability reasons. In addition, an extended scheme DTI+S is presented, which optimizes the processing of aggregate queries.

For discovering context information that is relevant for the situation of an application, we propose a powerful formalism for describing context models in a concise manner and a corresponding index structure (SDC-Tree). The formalism considerably extends existing approaches for describing information sources by constraints and permits to adjust between different semantics for matching descriptions against corresponding queries. The SDC-Tree enables to discover relevant context models out of potentially millions of descriptions efficiently using multidimensional indexing capabilities.

Deutsche Zusammenfassung

Skalierbare Verwaltung von Trajektorien und Kontextmodellbeschreibungen

1 Einleitung

Kontextbezogene Anwendungen und Systeme sind in der Lage sich an die physische Umgebung in der sie verwendet werden anzupassen. Beispiele reichen von kleinen Funktionsmerkmalen, wie der Anpassung der Klingeltonlautstärke eines Mobiltelefons an Hintergrundgeräusche, bis hin zu komplexen Anwendungsfällen, wie der Navigation sehbehinderter Personen [HKBE07]. Wichtige Aspekte des Kontextes einer Anwendung sind unter anderem der Ort, Benutzergewohnheiten und Geräte in räumlicher Nähe.

Die Entwicklung des zugehörigen Paradigmas der Informatik ist seit den Anfängen in den neunziger Jahren eng mit der Miniaturisierung und dem Preisverfall von Sensoren, Rechnern und Kommunikationshardware verbunden. Seither wurden auf der Erde Milliarden von Sensoren wie Kameras, Satelliten, Wetterstationen, drahtlose Sensornetze, Smartphones und GPS-Empfänger ausgebracht [CRHPP09,Rid07]. Die enormen Datenmengen dieser Sensoren ermöglichen die Erstellung von umfangreichen Modellen unserer Umgebung, wie zum Beispiel detaillierten Straßenkarten mit Echtzeitdaten zum Verkehrsaufkommen, 3D-Modellen von Gebäuden und feingranularen Wetterkarten.

Die Verfügbarkeit solcher Modelle bietet ein großes Potential für kontextbezogene Anwendungen, da diese nicht länger an vorgegebene Sensoren gebunden sind, sondern die für sie relevanten Kontextinformationen dynamisch von verschiedenen Anbietern wählen können. In Zukunft werden Millionen solcher *Kontextmodelle* erwartet, die von einer Vielzahl stationärer und mobiler *Kontextanbieter* bereitgestellt und von unterschiedlichsten Anwendungen genutzt werden [RDD+03, LCG+09].

Die Bereitstellung und gemeinsame Verwendung von Kontextinformationen wirft eine Reihe komplexer Fragestellungen auf, insbesondere zur Integration heterogener Daten, zur Auflösung von Inkonsistenzen und zur skalierbaren Verwaltung von Kontextinformationen. In den vergangenen Jahren wurden mehrere Ontologien für Kontextinformationen entwickelt (z.B. [RMCM03, CPFJ04, GWPZ04, BCQ+07, YCDN07]) und Rahmenwerke und Systeme zur Kontextverwaltung für mobile Endgeräte und speziell ausgestattete Räume oder Gebäude vorgestellt (z.B. [RHC+02, KMK+03, BC04, CFJ+04]).

Für weiträumige oder globale Szenarien mit Millionen von Kontextmodellen werden *verteilte* Konzepte zur Kontextverwaltung benötigt, die eine Reihe neuer Fragestellungen aufwerfen. Ein fundamentales Problem ist der effiziente Zugriff auf die immensen Mengen verteilter und dynamischer Kontextinformationen. In erster Näherung wird ein Discovery-Mechanismus benötigt, der es ermöglicht, jene Kontextmodelle aufzufinden, die für die aktuelle Situation einer Anwendung relevant sind. Dies umfasst zwei Teilprobleme:

1. *Formale Beschreibung von Kontextmodellen:* Zur Entscheidung ob ein Kontextmodell für die Situation einer Anwendung relevant ist, wird ein Formalismus zur prägnanten und präzisen Beschreibung von Kontextmodellen und zum Matching mit entsprechenden Anfragen benötigt.

2. *Indexierung von Kontextmodellbeschreibungen:* Damit die relevanten Modelle unter Millionen von Beschreibungen rasch aufgefunden werden können, ist weiterhin eine geeignete Struktur zur Indexierung und Suche der Beschreibungen erforderlich.

Informationen über die Bewegungen mobiler Objekte (d.h. deren Trajektorien) sind für die Kontextverwaltung von besonderer Bedeutung. Sie werden nicht nur für den Zugriff auf den *Primärkontext* der Objekte – Ort, Zeit und Identität [DA00, RDD+03, YCDN07] – benötigt, sondern auch für den Zugriff auf weitere, damit verbundene Kontextinformationen. Dies gilt in besonderem Maße für mobile Sensoren, aber auch für alle anderen beweglichen Entitäten im Sinne der Definition von Kontext [Dey00, DA00]. Zur Erfassung und Verwaltung von Trajektorien werden spezielle Datenbanken für bewegliche Objekte (engl. *Moving Objects Database*, MOD) verwendet [MGA03, GS05].

Für den effizienten Zugriff auf verteilte, dynamische Kontextinformationen in weiträumigen oder gar globalen Szenarien werden folglich auch skalierbare Verfahren und Strukturen zur Erfassung und Verwaltung von Trajektorien benötigt. Die Notwendigkeit die Trajektorien in Echtzeit zu erfassen – um den Zugriff auf aktuelle *und* frühere Positionsdaten zu ermöglichen – und die große Zahl an Objekten verursachen dabei zwei Teilprobleme, die von existierenden Ansätzen nicht abgedeckt werden:

3. *Effiziente Echtzeiterfassung von Trajektorien:* Bei sehr vielen Positionsbestimmungssystemen (insbesondere GPS) erfolgt die eigentliche Bestimmung mit entsprechenden Sensoren direkt bei den mobilen Objekten. Zur Echtzeiterfassung der Trajektorien in einer MOD müssen die Daten regelmäßig per Funk übertragen werden. Zur Optimierung des Speicherbedarfs und der Kommunikationskosten werden Protokolle benötigt, die es ermöglichen zwischen diesen Kosten und der Genauigkeit der Daten abzuwägen.

4. *Verteilte Indexierung räumlich partitionierter Trajektorien:* Übersteigt die Zahl der zu verwaltenden Objekte die Kapazität eines einzelnen Datenbankservers, so müssen die Trajektoriendaten auf mehrere Server verteilt werden. Räumliche Partitionierung ist hierfür besonders geeignet, da sie sowohl die Verarbeitung von Anfragen zu Abschnitten einer einzelnen Trajektorie als auch zu allen Trajektorienabschnitten in einem gewissen Gebiet unterstützt. Die effiziente Verarbeitung von Anfragen des erstgenannten Typs erfordert allerdings eine geeignete Indexstruktur zum raschen Zugriff auf die Partitionen einer Trajektorie.

Der Schichtung kontextbezogener Systeme folgend (von den Sensoren unten, bis hin zu den Anwendungen oben), werden zunächst in Kapitel 2 die Protokolle *Connection-Preserving Dead Reckoning* (CDR) und *Generic Remote Real-Time Trajectory Simplification* (GRTS) als Lösung für das dritte Teilproblem präsentiert. Beide ermöglichen es zwischen der räumlichen Genauigkeit mit der eine Trajektorie erfasst wird und dem Speicherbedarf sowie den Kommunikationskosten abzuwägen. CDR minimiert dabei die Kommunikati-

onskosten während GRTS den Speicherbedarf bis auf wenige Prozent über dem theoretischen Minimum reduzieren kann.

Für das vierte Teilproblem wird anschließend in Kapitel 3 der *Distributed Trajectory Index* (DTI) vorgeschlagen, welcher das effiziente Routing von Anfragen entlang einer räumlich partitionierten Trajektorie ermöglicht. Die erweiterte Variante DTI+S ermöglicht darüber hinaus, durch Verwendung sogenannter *Summaries*, die besonders rasche Verarbeitung von Aggregationsanfragen, wie zum Beispiel die Maximalgeschwindigkeit innerhalb eines Zeitintervalls.

Schließlich werden in Kapitel 4 für die ersten beiden Teilprobleme ein Formalismus zur Beschreibung von Kontextmodellen basierend auf *Defined Classes* [BCM+03] vorgestellt und der *Source Description Class Tree* (SDC-Tree) zur Indexierung solcher Beschreibungen präsentiert. Der Formalismus unterscheidet sich von verwandten, existierenden Ansätzen vor allem durch seine Mächtigkeit: Er ermöglicht geschachtelte Restriktionen über beliebige Relationen, alternative Beschreibungen und die Abstimmung zwischen zwei grundlegenden Semantiken zum Matching von Beschreibungen mit Anfragen nach Kontextmodellen.

Die wesentlichen Beiträge dieser Arbeit wurden auf internationalen Konferenzen publiziert [LDR08a, LDR08b, LFDR09, LDR10b]. Weitere im Rahmen dieser Arbeit entstandene Publikationen sind [FLR07, DPG+08, LCG+09, LWG+09, LDR10a] und beschreiben unter anderem den praktischen Einsatz der vorgestellten Konzepte und Algorithmen.

In den folgenden drei Abschnitten werden die wichtigsten Punkte dieser Beiträge kapitelweise zusammengefasst.

2 Effiziente Echtzeiterfassung von Trajektorien

Die Trajektorie eines mobilen Objekts wird üblicherweise als räumlich-zeitlicher Polygonzug dargestellt, dessen Ecken die mit einem entsprechenden Sensor gemessenen Positionen sind [LFG+03, MGA03, GS05]. Bei vielen Systemen, insbesondere GPS, erfolgt die Positionsbestimmung direkt bei dem mobilen Objekt. Würde zur Verwaltung der Trajektorie jede gemessene Position an eine MOD übermittelt und von dieser gespeichert, so wäre dies mit großem

Speicherbedarf und hohen Kommunikationskosten verbunden. Letzteres gilt besonders, wenn die MOD in Echtzeit über die Trajektorie informiert werden soll, was für viele Anwendungen erforderlich ist.

Daher wird ein Protokoll zur effizienten Echtzeiterfassung von Trajektorien benötigt, welches es ermöglicht, diese Kosten gegenüber der Genauigkeit der Trajektorieninformation seitens der MOD abzuwägen.

Eine einfache Lösung ist es, alle gemessenen Positionen an die MOD zu übermitteln und deren Menge dort geeignet zu reduzieren. Dazu existieren verschiedene Algorithmen zur *Simplifizierung* von Polygonzügen – auch *Kurvenglättung* genannt. Ein bekanntes Beispiel ist der Douglas-Peucker-Algorithmus [DP73]. Allerdings werden bei diesem Vorgehen sehr viele Positionen an die MOD übermittelt, die anschließend überhaupt nicht gespeichert werden.

Zur effizienten Echtzeiterfassung der aktuellen Position eines mobilen Objekts wurde vielfach die Verwendung von *Dead Reckoning* (wörtl. *Koppelnavigation*) vorgeschlagen [WSCY99, LR01, CJP05]. Bei Dead Reckoning sendet das mobile Objekt in einer entsprechenden Aktualisierungsnachricht neben der zuletzt gemessenen Position auch eine mathematische Funktion zur Vorhersage der künftigen Bewegung an die MOD. Eine erneute Aktualisierungsnachricht ist erst dann erforderlich, wenn das Objekt droht, sich von der vorhergesagten Position um mehr als eine tolerierte Abweichung ϵ zu entfernen. Als besonders einfach und trotzdem effektiv hat sich lineares Dead Reckoning (LDR) erwiesen, das zur Vorhersage lediglich einen Geschwindigkeitsvektor verwendet. Dead Reckoning optimiert zwar die Zahl der Nachrichten, liefert aber keinen zusammenhängenden Polygonzug.

Für LDR wurde in [TCS+06] gezeigt, dass der durch die Ursprünge der Vorhersagen gegebene Polygonzug die tatsächliche Bewegung mit einer Genauigkeit von 2ϵ annähert. Zur effizienten Echtzeiterfassung von Trajektorien wird daher die Verwendung von LDR mit $\epsilon' := \frac{1}{2}\epsilon$ vorgeschlagen – im Folgenden mit $\mathrm{LDR}_{\frac{1}{2}}$ abgekürzt.

In Kapitel 2 wird dieser Ansatz eingehend analysiert und gezeigt, dass er viele unnötige Aktualisierungsnachrichten verursacht. Aus diesem Grund werden anschließend zwei maßgeschneiderte Protokolle namens *Connection-Preserving Dead Reckoning* (CDR) und *Generic Remote Real-Time Trajectory Simplifi-*

cation (GRTS) vorgeschlagen. Beide ermöglichen den Speicherbedarf und den Kommunikationsaufwand gegenüber einer tolerierten Abweichung ϵ, mit der die MOD über die tatsächliche Bewegung informiert wird, abzuwägen. Beide Protokolle berücksichtigen dabei auch mögliche Abweichungen zwischen der tatsächlichen Bewegung des Objekts und dem gemessenen Polygonzug.

CDR basiert auf LDR, ergänzt dieses aber um eine weitere Bedingung für das Senden einer Aktualisierungsnachricht. Und zwar sendet CDR nicht nur eine Aktualisierung wenn das Objekt droht um mehr als ϵ von der Vorhersage abzuweichen, sondern auch wenn die aktuell gemessene Position nicht als weitere Ecke für den simplifizierten Polygonzugs geeignet ist. Letzteres ist der Fall, wenn die Strecke zwischen der zuletzt übermittelten Position und der aktuell gemessenen Position um mehr als ϵ von der tatsächlichen Bewegung dazwischen abweicht oder abweichen könnte. Tritt dieser Fall ein, so sendet CDR die zuvor gemessene Position – welche noch als Ecke für den simplifizierten Polygonzugs geeignet war – zusammen mit einem neuen Geschwindigkeitsvektor an die MOD. Zur Überprüfung der zweiten Bedingung speichert CDR alle gemessenen Positionen seit der letzten Aktualisierungsnachricht in der *Sensing History* $\mathbb{S}$.

Es zeigt sich, dass viele Positionen nach gewisser Zeit – noch vor der nächsten Aktualisierungsnachricht – aus $\mathbb{S}$ entfernt werden können. Diese Optimierung reduziert den Speicherbedarf auf dem mobilen Objekt um durchschnittlich 49%. Allerdings ist die Größe von $\mathbb{S}$ trotz dieser Optimierung theoretisch unbegrenzt – und damit auch die Rechenzeit je Positionsbestimmung. Daher wird anschließend die Variante CDR_m vorgestellt, welche die Größe von $\mathbb{S}$ auf m Positionen begrenzt. In ausführlichen Simulationen mit realen GPS-Traces hat sich gezeigt, dass CDR_m mit $m = 500$ dieselben Reduktionsraten wie CDR erreicht und gleichzeitig die Rechenzeit mit einem 1 GHz-Prozessor auf rund 0.08 ms begrenzt.

GRTS verwendet zwar auch LDR zur Echtzeiterfassung der aktuellen Position, trennt diese aber soweit als möglich von der Simplifizierung der zeitlich zurückliegenden Trajektorie. Anders als CDR verwendet es zur Simplifizierung daher nicht Dead Reckoning, sondern kann mit beliebigen Simplifizierungsalgorithmen realisiert werden.

Dazu speichert GRTS die gemessenen Positionen ebenfalls in einer Sensing History $\mathbb{S}$. Droht das mobile Objekt um mehr als ϵ von der Vorhersage abzuweichen, so wendet GRTS den entsprechenden Simplifizierungsalgorithmus auf $\mathbb{S}$ an und fügt die so berechneten neuen Ecken der simplifizierten Trajektorie der Aktualisierungsnachricht hinzu.[3] Die Simplifizierung kann allerdings auch zuvor übermittelte Ecken der simplifizierten Trajektorie abändern, d.h. ersetzen. Dies gilt insbesondere für die letzte Ecke, die den Ursprung der bisherigen Vorhersage bildet. Zusätzlich zu den neuen Ecken und dem Geschwindigkeitsvektor für LDR wird daher in jeder Aktualisierungsnachricht angegeben, wie viele der bisherigen Ecken – beginnend mit der letzten – zunächst zu entfernen sind. Bei einem großen Teil der Aktualisierungen bleibt die Zahl der Ecken der simplifizierten Trajektorie unverändert, und damit auch der Speicherbedarf seitens der MOD. GRTS erzielt daher bessere Reduktionsraten als CDR, welches mit jeder Aktualisierung eine weitere Ecke hinzufügt.

Die simplifizierte Trajektorie kann bei GRTS in drei Abschnitte unterteilt werden: (1.) Einen unveränderlichen Abschnitt, der nur von der MOD gespeichert wird. (2.) Einen veränderlichen Abschnitt, welcher an der ersten Position in $\mathbb{S}$ beginnt und durch künftige Aktualisierungsnachrichten ganz oder teilweise ersetzt werden kann. (3.) Den durch LDR vorhergesagten Abschnitt.

Der eben skizzierte Basisalgorithmus von GRTS legt weder fest, wie viele Ecken der zweite, veränderliche Abschnitt umfasst, noch wie viele Positionen in $\mathbb{S}$ zwischengespeichert werden. Dementsprechend sind zwei grundlegende Varianten möglich: GRTS_k begrenzt den veränderlichen Abschnitt auf k Ecken. Selbst für $k = 1$ können Reduktionsraten nur 10% unter der theoretisch bestmöglichen erreicht werden. Allerdings impliziert k keine strikte Begrenzung für die Größe von $\mathbb{S}$, welche die zur Simplifizierung benötigte Rechenzeit maßgeblich beeinflusst. GRTS_m hingegen begrenzt die Größe von $\mathbb{S}$ auf m Positionen und damit auch die Anzahl der Ecken des zweiten Abschnitts. Erreicht $|\mathbb{S}|$ den Parameter m, so wird $\mathbb{S}$ simplifiziert, die simplifizierten Ecken für die nächste Aktualisierungsnachricht zwischengespeichert und alle Positionen bis zur ersten neuen Ecke aus $\mathbb{S}$ entfernt. Folglich erzeugt GRTS_m nach spätestens m Positionsbestimmungen eine weitere Ecke. Um diesen Nachteil gegenüber

[3] Die erste so berechnete Ecke entspricht der ersten gemessenen Position in $\mathbb{S}$ und wird nicht weiter berücksichtigt, weil sie der MOD in jedem Fall bereits bekannt ist.

GRTS_k zu mindern, wird in Abschnitt 2.5.3 die Variante GRTS_mc vorgestellt, welche die Größe von $\mathbb{S}$ durch eine Technik zur Kompression von Trajektorienabschnitten reduziert.

Danach wird besprochen, wie GRTS mit dem optimalen Simplifizierungsalgorithmus von Imai und Iri [II88] und einer einfachen aber wirkungsvollen Heuristik [MdB04, AHPMW05, HGNM08] realisiert werden kann. Bei letzterer handelt es sich um einen Online-Algorithmus, so dass die Simplifizierung inkrementell – nach jeder Positionsbestimmung – ausgeführt werden kann. Für diese Heuristik wird überdies eine Optimierung vorgestellt, welche zur weiteren Simplifizierung nicht mehr benötigte Positionen aus $\mathbb{S}$ vorzeitig entfernt.

Anschließend werden die Ergebnisse ausführlicher Simulationen von CDR und GRTS mit mehreren hundert realen GPS-Traces präsentiert und erklärt. Beispielsweise zeigt sich, dass die Kompressionstechnik von GRTS_mc bei Realisierung mit der genannten Heuristik überflüssig ist – dank der vorgestellten Optimierung dieser Heuristik.

Zum Vergleich mit der theoretisch bestmöglichen Reduktionsrate wurde ein optimaler Simplifizierungsalgorithmus nachträglich auf die gesamten GPS-Traces angewendet. Der Vergleich ergibt, dass GRTS_mc mit dem optimalen Algorithmus von Imai und Iri und $m = 500$ durchschnittlich 97% der bestmöglichen Reduktionsrate erzielt und dazu mit einem 1 GHz-Prozessor höchstens 21 ms Rechenzeit je Positionsbestimmung benötigt.

Aus diesen Ergebnissen werden drei Schlussfolgerungen für die Wahl eines bestimmten Verfahrens gezogen: Ist der Speicherbedarf für die simplifizierten Trajektorien von wichtigster Bedeutung und verfügen die mobilen Objekte über ausreichend Rechenleistung, so sollte GRTS_mc mit dem optimalen Simplifizierungsalgorithmus von Imai und Iri verwendet werden. Sollen die Kommunikationskosten minimiert werden, so ist CDR_m wegen der etwas geringeren Datenraten zu verwenden. Für alle anderen Anwendungsszenarien (und damit wohl die meisten) ist GRTS_m mit der genannten Simplifizierungsheuristik wegen seiner guten Reduktionsraten und dem geringen Rechenaufwand die beste Wahl.

In Kapitel 2 werden ferner physikalische Modelle zur Bestimmung der möglichen Abweichung zwischen der tatsächlichen Bewegung eines mobilen Ob-

jekts und der gemessenen Trajektorie erörtert und bewertet. Außerdem wird eine prototypische Implementierung von GRTS für Smartphones und Subnotebooks, mit der die Trajektorien dieser Geräte in Google Earth in Echtzeit verfolgt werden können, beschrieben und von entsprechenden Experimenten berichtet.

3 Verteilte Indexierung räumlich partitionierter Trajektorien

Wie eingangs erläutert können Anfragen über Trajektorien oder zu Kontextinformationen, die mit Trajektorien verknüpft sind, in zwei wesentliche Klassen eingeteilt werden: (1.) Trajektorienbasierte Anfragen, die sich auf Abschnitte einer vorgegebenen Trajektorie beziehen und (2.) koordinatenbasierte Anfragen, die alle Trajektorienabschnitte in einem gewissen Gebiet betreffen [PJT00, GS05].

In ähnlicher Weise lassen sich zwei grundlegende Schemata zur Partitionierung von Trajektoriendaten auf mehrere Datenbankserver unterscheiden: (1.) Bei objektbasierter Partitionierung werden die Trajektorien als vollständige Datensätze verschiedenen Servern zugeteilt. (2.) Bei räumlicher Partitionierung wird jedem Server ein disjunkter Ausschnitt des gesamten Raums zugeordnet, die Trajektorien entsprechend dieser Ausschnitte in Partitionen zerlegt und auf den zugehörigen Servern gespeichert.

Beide Schemata weisen sehr unterschiedliche Charakteristika hinsichtlich der Anfrageverarbeitung auf: Die objektbasierte Partitionierung ermöglicht die effiziente Verarbeitung trajektorienbasierter Anfragen, da jede Anfrage genau einem Server zugeordnet werden kann. Koordinatenbasierte Anfragen hingegen müssen von potentiell allen Servern verarbeitet werden. Umgekehrt lässt sich bei räumlicher Partitionierung eine koordinatenbasierte Anfrage einem oder wenigen Servern zuordnen. Für die räumliche Partitionierung spricht, dass dies auch für trajektorienbasierte Anfragen gilt, da ein gewisser Abschnitt einer vorgegebenen Trajektorie stets von einigen wenigen benachbarten Servern gespeichert wird. Allerdings sind diese Server nur indirekt durch die Anfrage und den Verlauf der Trajektorie gegeben. Zur effizienten Verarbeitung trajektorienbasierter Anfragen wird folglich eine geeignete Struktur zum raschen Zugriff auf die Partitionen einer Trajektorie benötigt.

Als weiterer Vorteil räumlicher Partitionierung sei angemerkt, dass die Server nahe der ihnen zugeordneten Ausschnitte des Raums platziert werden können. Auf diese Weise lassen sich die Kommunikationswege zwischen den mobilen Objekten und den Servern minimieren [PS01, LR02].

Zur effizienten Verarbeitung trajektorienbasierter Anfragen in räumlich partitionieren MODs wird in Kapitel 3 der *Distributed Trajectory Index* (DTI) vorgestellt. Dieser ermöglicht eine Anfrage entlang einer Trajektorie rasch zu jenen Servern zu leiten, welche die angefragten Partitionen der Trajektorie speichern.

Zunächst wird ein generelles Verfahren zur Verarbeitung einer trajektorienbasierten Anfrage eingeführt: In der ersten Phase dieses Verfahrens wird die Anfrage mit Hilfe des *Home-Server-Pointers* an einen beliebigen Server geleitet, der eine Partition der angefragten Trajektorie speichert. In der zweiten Phase wird die Anfrage von Server zu Server entlang der Trajektorie in Richtung des angefragten Zeitintervalls weitergeleitet. Wurde das Ende (oder auch der Anfang) des angefragten Abschnitts erreicht, so wird die Anfrage in der dritten Phase entsprechend verarbeitet und – falls notwendig – wiederum von Server zu Server zusammen mit möglichen Teilergebnissen weitergeleitet.

Speichert ein Server mehrere Partitionen derselben Trajektorie, so kann die Anfrage in der zweiten Phase zwischen diesen "abkürzen". Die grundlegende Idee von DTI ist, ähnliche Abkürzungen zu entfernten Partitionen der Trajektorie durch entsprechende Pointer explizit bereitzustellen. Ein solcher *DTI-Pointer* ist einfach eine Kopie einer anderen Position, d.h. der geographischen Koordinaten und des zugehörigen Zeitstempels. Er kann dazu verwendet werden eine Nachricht direkt an den Server zu leiten, der die entsprechende Position speichert. Ob die Nachricht tatsächlich unmittelbar an den Server übermittelt werden kann oder ob dazwischen geographisches Routing erforderlich ist, hängt davon ab, wie die Server miteinander vernetzt sind und ist für DTI irrelevant. Daher kann DTI als temporales Routing-Overlay über einem geographischen Overlay aufgefasst werden.

Die DTI-Pointer werden so gewählt und mit ausgewählten Positionen der Trajektorie verknüpft, dass sie eine perfekte, bidirektionale Skip-Liste [Pug90] bilden. Dazu wird periodisch die aktuelle Position als *DTI-Node* deklariert, mit

aufsteigender Sequenznummer versehen und Pointer zu älteren DTI-Nodes gemäß der Skip-Liste angelegt. Zum Anlegen der Pointer wird eine entsprechende Nachricht an die Server der betroffenen älteren DTI-Nodes gesendet.

Dank der Skip-Liste hängt die Anzahl der zum Routing entlang eines Zeitintervalls benötigten DTI-Pointer logarithmisch von der Länge des Intervalls ab. Da die DTI-Pointer nicht direkt auf die Server verweisen, sondern lediglich auf geographische Positionen, kann die zu Grunde liegende räumliche Partitionierung ohne Beeinträchtigung von DTI geändert werden. Die DTI-Nodes müssen lediglich zusammen mit den verknüpften Positionen migriert werden.

DTI ermöglicht das effiziente Routing von Anfragen in der zweiten Phase der Verarbeitung. Durch Erweiterung des DTI um *Summaries* (DTI+S) kann aber auch die eigentliche Verarbeitung bestimmter Aggregationsanfragen in der dritten Phase beschleunigt werden. Dies gilt für alle Aggregate, die auf Basis existierender Aggregate kleinerer Zeitintervalle ermittelt werden können. Exemplarisch seien die Länge, die Maximalgeschwindigkeit und das minimal umgebende Rechteck eines Trajektorienabschnitts genannt.

DTI+S speichert mit jedem DTI-Pointer eine *Summary*, welche die entsprechenden Aggregate des von diesem Pointer überspannten bzw. abgekürzten Trajektorienabschnitts enthält. Die Summaries werden jeweils mit den DTI-Pointern erstellt. In der dritten Phase der Anfrageverarbeitung wird abhängig vom Anfragetyp und bereits vorliegenden Teilergebnissen entschieden, welche Summaries zur Verarbeitung herangezogen werden können. Aus diesen wird die nützlichste Summary ausgewählt und die Anfrage entlang des zugehörigen DTI-Pointers weitergeleitet.

Zur Beurteilung der Effektivität von DTI und DTI+S wurde eine räumlich partitionierte MOD bestehend aus 1000 Servern in verschiedenen Experimenten simuliert. Die Server speicherten die Bewegung von 100 mobilen Objekten während eines Jahres in einem rechteckigen Gebiet von der Größe der USA. Nach rund acht Monaten simulierter Zeit wurden in jedem Experiment 10^7 Anfragen verschiedenen Typs über unterschiedliche Abschnitte der Trajektorien gestellt. Die Messungen ergaben, dass die Verwendung von DTI die Zeit zum Routing von Anfragen in der zweiten Phase um bis zu 69% gegenüber Routing ohne DTI reduziert. Bei entsprechenden Anfragetypen kann DTI+S

die Gesamtzeit zur Anfrageverarbeitung sogar um mehr als 95% verkürzen – unter Berücksichtigung der lokalen Indexstrukturen der Server und der entsprechenden Festplattenzugriffs- und Rechenzeiten.

In der abschließenden Diskussion verwandter Arbeiten in Abschnitt 3.7 wird die Einzigartigkeit des Ansatzes gezeigt und erörtert, wie sich DTI und existierende Algorithmen zur Verarbeitung koordinatenbasierter Anfragen in räumlich partitionierten MODs ergänzen.

4 Beschreibung und Indexierung von Kontextmodellen

Hinsichtlich des Auffindens von Kontextmodellen kann ein verteiltes Kontextmanagementsystem als heterogenes Informationssystem (HIS) aufgefasst werden, welches aus einer großen Zahl von Datenquellen besteht. Die meisten der existierenden Ansätze zum Auffinden von Datenquellen in HIS verwenden die Abbildungen zwischen den Schemata der Datenquellen oder zu einem gemeinsamen Schema, um jene Quellen auszuschließen, die zu einer bestimmten Relation oder Klasse keine Informationen bereitstellen. Diese Vorgehensweise ist vergleichsweise grobgranular und skaliert nicht mit der Anzahl der Datenquellen. Beispielsweise könnte es tausende von Kontextanbietern geben, die ähnliche Schemata verwenden um Pläne oder 3D-Modelle verschiedener Gebäude bereitzustellen.

Daher wird ein dedizierter Formalismus zur Beschreibung von Datenquellen in HIS benötigt. Das mögliche Spektrum reicht von Listen mit Schlagworten bis hin zu logikbasierten Ansätzen, die Restriktionen wie Preis $\geq 200\,\$$ zum Ausschluss von Quellen für eine gegebene Anfrage mit Preis $\leq 30\,\$$ verwenden.

Dieses Spektrum zeigt gleichzeitig auf, dass unterschiedliche Semantiken zum Matching mit Anfragen denkbar sind. Schlagworte implizieren eine *positive* Semantik, da eine Beschreibung eine gegebene Anfrage nur dann erfüllt, wenn die Anfrage die "richtigen" Schlagworte enthält, die auch zur Beschreibung der Quellen verwendet wurden. Die angedeutete Verwendung von Restriktionen hingegen impliziert eine *negative* Semantik, da die Restriktionen nur dem Ausschluss von nachweislich irrelevanten Quellen dienen.

In Kapitel 4 wird daher für HIS bei denen die Quellen eine gemeinsame Ontologie verwenden – eine Annahme, die insbesondere für Kontextmanagementsysteme gilt [RMCM03, CPFJ04, GWPZ04, BCQ+07, YCDN07] – ein erweiterter logikbasierter Formalismus vorgestellt. Dieser ermöglicht nicht nur geschachtelte Restriktion über Relationen der Ontologie, sondern unterscheidet auch zwischen alternativen Beschreibungen derselben Quelle. Außerdem bietet er die Möglichkeit in den Anfragen zwischen positiver und negativer Matching-Semantik abzuwägen.

Dazu wird jede Quelle durch ein oder mehrere *Defined Classes* [BCM+03] beschrieben, das heißt durch eine Klasse der Ontologie gefolgt von Restriktionen, wie zum Beispiel

$$D_1 = \langle \textsf{BuildingPart} : \textsf{partOf} \in \langle \textsf{Museum} : \textsf{name} \in \{\text{“British Museum”}\} \rangle \rangle$$

zur Beschreibung eines Gebäudemodells des Britischen Museums. Da der Name dieses Museums unzweideutig das berühmte Museum in London bezeichnet, ist keine Ortsangabe notwendig. Stattdessen kann die Defined Class

$$D_2 = \langle \textsf{BuildingPart} : \textsf{location} \in \{44 \text{ Gt Russell St, London, UK}\} \rangle$$

alternativ zur Beschreibung des Gebäudemodells verwendet werden. Entsprechend kann das Gebäudemodell über den Namen oder den Ort aufgefunden werden. Anfragen nach Kontextmodellen werden ebenfalls als Defined Classes formuliert. Während eine Beschreibung aber mehrere, alternative Defined Classes $\{D_1, \ldots, D_n\}$ umfassen kann, besteht eine Anfrage aus nur einer Defined Class Q, die alle der Anwendung hinsichtlich ihres Informationsbedarfs bekannten Restriktionen enthält.

Zur Auswertung der Defined Classes $\{D_1, \ldots, D_n\}$ mit Q werden zwei Prädikate definiert: Das *Query Matching Predicate* $\rightsquigarrow_{\text{Q}}$ stellt eine notwendige Bedingung für das Matching einer Beschreibung mit einer Anfrage dar. Damit $D_i \rightsquigarrow_{\text{Q}} Q$ erfüllt ist, muss für jede Restriktion von D eine überlappende Restriktion in Q definiert sein. Das *Query Dismatching Predicate* $/\!\!/_{\text{Q}}$ ist eine hinreichende Bedingung für Nicht-Matching. $D_j /\!\!/_{\text{Q}} Q$ ist erfüllt, wenn in D_j und Q Restriktionen auf demselben Attribut a oder derselben Relation

r definiert sind, deren Wertbereiche sich nicht überlappen. Durch Hinzufügen von Pseudo-Restriktionen $a \in *$ bzw. $r \in *$ zu Q kann die Bedeutung von $\rightsquigarrow_{\text{Q}}$ gegenüber $/\!\!/_{\text{Q}}$ abgestuft werden und so zwischen positiver oder negativer Matching-Semantik gewählt werden.

Da die Konzeption einer Indexstruktur für Beschreibungen von Datenquellen eng vom entsprechenden Formalismus abhängt, wird im selben Kapitel der *Source Description Class Tree* (SDC-Tree) zur Indexierung von Defined Classes vorgestellt. Jeder Knoten repräsentiert eine erweitere Defined Class N_i, welche *Node Class* genannt wird. Die Node Class eines jeden Knoten subsumiert die Node Classes der Kindknoten gemäß dem *Index Subsumption Predicate* $\succeq_{\text{I}}$, welches $\rightsquigarrow_{\text{Q}}$ impliziert. Die Node Classes bilden also eine Halbordnung, welche die Struktur des Baums widerspiegelt. Die Wurzel repräsentiert die Node Class $\langle C_\top, \textsc{True} : \rangle$, wobei $C_\top$ die Wurzelklasse der IS-A-Hierarchie der gemeinsamen Ontologie ist.

Eine Defined Class D einer zu indexierenden Beschreibung wird beginnend von der Wurzel an jene Blätter weitergereicht, deren Node Classes ein passend definiertes *Index Matching Predicate* $\rightsquigarrow_{\text{I}}$, welches $\rightsquigarrow_{\text{Q}}$ impliziert, mit D erfüllen. D wird also an jedem Blatt mit Node Class N_i $\rightsquigarrow_{\text{I}}$ D gespeichert. Eine gegebene Anfrage Q wird analog unter der Bedingung $N_i \rightsquigarrow_{\text{Q}} Q$ an die zutreffenden Blätter geleitet. Auf diese Weise werden ganze Teilbäume ausgeschlossen, deren Node Classes und Defined Classes $\rightsquigarrow_{\text{Q}}$ für Q nicht erfüllen.

Defined Classes können sich in drei Aspekten voneinander unterscheiden: (1.) Der Klasse und Subklassen der Ontologie, auf die sie sich beziehen, (2.) der Existenz von Restriktionen für ein gewisses Attribut oder eine gewisse Relation und (3.) dem Wertebereich der Restriktionen. Dies gilt auch für geschachtelte Defined Classes wie $\langle \textsf{Museum} : \textsf{name} \in \{\text{“British Museum”}\} \rangle$ im obigen Beispiel.

Die Node Classes sind so definiert, dass Defined Classes nach allen drei Aspekten unterschieden und indexiert werden können. Entsprechend kann ein Blattknoten und dessen Node Class auf drei Arten in zwei oder mehr Kindknoten und Node Classes aufgeteilt werden.

Zur automatischen Aufteilung von Blattknoten dient der *Generic Split Algorithm* (GSAlg): Überschreitet die Anzahl indexierter Defined Classes eines Blattknotens einen gewissen Schwellwert, so ermittelt GSAlg alle möglichen

Aufteilungen unter Berücksichtigung aller Attribute und Relationen, bewertet deren Nützlichkeit für die vorliegenden Defined Classes und wählt schließlich die beste Aufteilung aus.

Zur Erprobung und Bewertung des SDC-Tree wurde dieser mit potentiellen Beschreibungen von Kontextmodellen simuliert, welche unter Verwendung der OpenStreetMap-Datenbank [OSM] generiert wurden. Die Simulationsergebnisse mit bis zu 10^5 Defined Classes zeigen, dass der Aufwand für die Suche nach Beschreibungen logarithmisch von der Anzahl indexierter Defined Classes abhängt und der SDC-Tree daher die angestrebte effiziente Discovery von Kontextmodellen und anderen Datenquellen ermöglicht.

Zum Abschluss werden in Abschnitt 4.6 verwandte Arbeiten diskutiert. Es wird gezeigt, dass der Formalismus zur Beschreibung von Kontextmodellen und anderen Datenquellen deutlich über existierende Ansätze hinausgeht. Gleiches gilt für die Konzepte zur Indexierung der Beschreibungen im SDC-Tree.

1 Introduction

Context-awareness refers to the idea that applications and systems adapt to their context of use including location, noise, nearby devices, user habits and the social environment amongst others. Examples for context-awareness range from simple software features, e.g. adjusting the ring volume of a mobile phone for the ambient noise, to complex application tasks such as navigation of blind persons [HKBE07] or emergency assistance to elderly people [BCM06].

From the beginning in the 1990s, advancements in context-aware computing are closely connected to the miniaturization of sensing, computing, and communication hardware. For example, the Active Badge Location System [WHFG92], which is considered one of the first context-aware applications [BDR07], bases on wearable tags (*badges*) with infrared (IR) emitters and stationary IR sensors. Each badge periodically emits a unique signal to be received by the IR sensors. The sightings of the IR sensors allow tracking the position of a badge and its wearer. In its initial experimental application, the system was used for forwarding incoming calls to the phone closest to the callee [WHFG92].

Since these days, billions of sensors have been deployed all over the globe, including cameras, satellites, weather stations, wireless sensor networks, global positioning system (GPS) receivers, and radio-frequency identification (RFID) sensors. By 2008, the Earth supported several billion RFID sensors and GPS receivers [CRHPP09]. A significant number of the more than three billion mobile phones on earth [Rid07] are smartphones, equipped with a variety of sensors as well as powerful processors and high data rate Internet connection.

Consider, for example, the Apple iPhone 3GS [iPhone3GS]: It features a GPS receiver, proximity and ambient light sensors, a 3-axis accelerometer, a digital compass and a 3.0 megapixel camera besides the requisite microphone. These sensors not only enable to search, present, and enrich information based on the user's location, but also to infer the user's activities (sitting, walking,

meeting friends), habits (at the gym, coffee shop, at work), and surroundings (noisy, hot, bright) by sensor data fusion and analysis [MLF+08].

The immense amounts of data acquired by the billions of mobile and stationary sensors allow creating comprehensive models of our physical environment. A number of research projects from different areas, including wireless sensor networks, public sensing, environmental monitoring, and geographic information systems, tackle the technical challenges for acquisition and creation of such models. Examples for such projects are CitySense [MMR+08] to monitor weather conditions and air pollutants using an urban-scale sensor network, SenseWeb [LKNZ08] to share data streams from sensors all over the globe, and CarTel [HBZ+06] to collect and analyze traffic data from mobile sensors in cars or smartphones.

In addition, there already exist a number of online community projects for the acquisition and creation of models of the physical environment. For example, the OpenStreetMap project [OSM, HW08] provides a world map created by users all over the globe using GPS traces, whereas the Automatic Weather Map System [AWEKAS] and the Citizen Weather Observer Program [CWOP] are projects that collect and provide data from private weather stations.

The availability of such models constitutes a huge potential for context-aware computing. Context-aware applications are no longer bound to predefined sensors but can dynamically select relevant context information from different providers. Therefore, these models and their providers are referred to as *context models* and *context providers*, respectively [RDD+03, LCG+09].

In future, millions of context models can be expected, which are shared by a wide variety of applications. The models not only cover different aspects and areas of the physical world but may also differ in granularity, topological and topographical nature, dynamism, and temporal extent. Some models only provide information on the current point in time, whereas others provide historical information such as the trajectories of containers or vehicles. Context models may even provide predictions on the future state of the physical world such as weather forecasts.

The sharing of context information poses a number of important challenges to context-aware computing research – including schema integration, privacy protection, resolving of inconsistencies, and the scalable management of con-

text information in particular. In the last years, a number of context management frameworks and systems have been proposed for mobile devices (e.g. [HSP+03, KMK+03, BC04]) as well as for rooms or buildings equipped with a number of networked devices and sensors (e.g. [RHC+02, CFJ+04, WDC+04]).

Context management in large-scale or global scenarios, however, is tackled by few works only. One approach is the Nexus platform (cf. Section 1.3), in which this thesis was carried out.

A fundamental problem for context management in large-scale scenarios is how to provide efficient access to the immense amounts of distributed dynamic context information. In the first instance, an approach for discovering context models that are relevant for the current situations of applications and their users is required. The mentioned mobility of users, sensors, and other physical entities, however, reveals another dimension of this problem: There exist large quantities of relevant entities that are bound neither to certain locations nor to single context models. In addition, there exist billions of mobile sensors that can be used for the acquisition of context models or that even provide moving context models. To allow for efficient access to information about the primary context (i.e. location, time, and identity [DA00, RDD+03, YCDN07]) of such objects or to other context information associated with their movement, scalable approaches for tracking and management of trajectories are needed.

In the following, we investigate this fundamental problem and give comprehensive solutions to the mentioned aspects.

1.1 Brief Introduction to Context-Aware Computing

Before we render the problem statement and our contributions more precisely in Section 1.2, we give a brief introduction to the history and developments in context-aware computing. Expert readers may skip this section.

The beginnings. The term 'context-aware computing' was introduced in 1994 by Bill N. Schilit and other researchers from the Xerox Palo Alto Research Center (PARC). They describe context-aware computing as *"the ability of a mobile user's applications to discover and react to changes in the environment they are situated in"* [ST94] and characterize respective context-aware software

as follows [SAW94]:

> *"Such context-aware software adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time. A system with these capabilities can examine the computing environment and react to changes to the environment."*

Relation to ubiquitous computing. Context-aware computing is closely connected to the integration of information processing into everyday objects and activities, generally referred to as *ubiquitous* or *pervasive computing*. This is also suggested by the fact that context-aware computing was introduced within the ubiquitous computing research program at Xerox PARC, under the direction of Mark Weiser, who is widely considered to be the father of ubiquitous computing.

In his seminal article *The Computer of the 21st Century* [Wei91] Weiser envisions how information technology indistinguishably weaves into the fabric of everyday life – in consequence of the seamless integration of networked and mostly miniaturized computers throughout the physical environment. In the resulting *embodied virtuality*, articles can be digitally copied from ordinary newspapers, electronic bookcases provide texts for download to portable computers, and drawings can be composed jointly – across the Atlantic Ocean – using digital flip charts. Of course, several of these ideas have been realized as consumer products since that time such as digital reading devices for electronic books.

Since context plays a vital role in many of these scenarios, context-awareness is generally called a core concept [BC04] or enabling technology [Sch02] for ubiquitous computing.

First applications. The above-mentioned Active Badge Location System [WHFG92] has been also used intensively at Xerox PARC for experimental applications [SAW94]: *Watchdog*, for instance, executes predefined Unix commands when the wearer of a badge enters a certain location or meets a specific person. *Contextual Reminders* permits to pop up messages depending on time, location, nearby persons, and proximate devices.

Context-aware computing attracted more and more research interest in the following years. One popular application area to this day is tourist guidance in cities and museums. For example, the GUIDE system [CDMF00] allows creating tailored city tours using portable computers and presents information on sights depending on the visitors' locations. More than ten other mobile guides are surveyed in [KB03] and [RTA05].

Another popular application area – which emerged in the last ten years, but also advances ideas from the beginnings of context-aware computing – are *smart environments*, i.e. rooms or buildings equipped with a number of networked devices and sensors. For example, the Gator Tech Smart House [HMEZ+05] provides an assistive environment for elderly people based on the occupants' locations and preferences as well as sensor data from thermostats, water sensors, RFID readers at power outlets, and many more.

Definition of context. A crucial issue is the understanding and definition of *context* in this area of research [Dou04]. From the beginning it was pointed out that location is only one facet of context, although it is of particular importance for most applications. Schilit et al. [SAW94] give lighting, noise level, network connectivity, communication costs, communication bandwidth, and the social situation as further examples. Besides, they state that three important aspects of context are: where you are, who you are with, and what resources are nearby. Similarly, Schmidt et al. [SBG99] distinguish between human factors and the physical environment. Human factors are categorized into information on the user, the user's social environment, and the user's tasks. The physical environment comprises location, infrastructure, and physical conditions.

Despite these examples and classifications, it took several years until a definition of context, proposed by Anind K. Dey, became widely accepted [Dey00, DA00]:

> *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."*[1]

[1] An extended version of this definition is given in [DAS01]: "[...] of entities (i.e., whether

Context management. According to this definition, context is always bound to entities of the physical world. Hence, information about context can be shared by different applications – despite the application-centric nature of the definition. Based on this property, various frameworks and systems for managing context information have been proposed. Although each of these approaches aims at supporting a multitude of applications, they differ in terms of intended system structures and application domains.

For example, the Hydrogen Context-Framework [HSP+03] is tailored for resource-constrained mobile devices. It allows applications to retrieve context information from different embedded sensors based on a common object-oriented data model. A similar framework for the Symbian platform is presented in [KMK+03]. It particularly features a Bayes classifier to recognize higher-level context information such as driving noise and music styles.

The CASS middleware [FC04] is a server-based middleware but likewise targets mobile context-aware applications. The context information is managed at a central database server and transmitted to the mobile devices as needed. A similar approach is taken by the Context Broker Architecture (CoBrA) [CFJ+04], the Service-Oriented Context-Aware Middleware (SOCAM) [GPZ05b], and the Semantic Space Infrastructure [WDC+04] – three context middleware systems for smart environments. Each of them stores the context information in a central knowledge base in order to provide a consistent context model of the environment including all users and devices. The Gaia metaoperating system [RHC+02], on the contrary, uses a distributed approach for context management in smart environments. It assumes a number of context providers and components for inferring higher-level context information. A registry maintains a list of all providers and allows applications to discover the context information they need.

As stated initially, only few works tackle large-scale or global scenarios with millions of context models and providers. Nimbus [Rot03] is a scalable platform for managing geographic information. It is based on a hierarchical location

a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects."

model, ranging from countries and cities to buildings and rooms. The location model is constituted by a large number of servers, where each server provides a clipping of the hierarchy, e.g. the location information for a city or a university. Semantic Context Space (SCS) [GPZ05a] is a scalable lookup service for context providers using an overlay network. It assumes an ontology that is shared by all context providers and allows retrieving those providers that have information about a given ontology class. Nexus [LCG+09], which is further explained in Section 1.3, aims for a global context management platform using a scalable federation approach.

Context ontologies. An important characteristic of each of these frameworks and systems is the assumption of a shared ontology for semantic interoperability between context providers and applications. One example is the Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) [CPFJ04], which has been proposed with CoBrA [CFJ+04]. It is expressed in the Web Ontology Language (OWL) [OWL] and references several well-established ontologies and calculi such as the OpenCyc Spatial Ontology [OpenCyc] and the Region Connection Calculus (RCC) [RCC92]. Surveys about ontologies and modeling approaches for context-aware computing are given in [YCDN07] and [BCQ+07].

Importance of location. As aforementioned, location information is a very important type of context information. Together with time and identifiers of entities, it may be used as an index to access other context information [RDD+03]. For this reason those three are referred to as *primary context* [DA00, RDD+03, YCDN07]. Location-based services such as friend-finding [OCS06] can be considered as special context-aware applications that are solely based on location information.

Therefore, location-sensing techniques have been always an integral part of ubiquitous and context-aware computing research (e.g. [SAW94, HHS+99, WDC+04, CL08]). The same applies to location services and moving objects databases (MODs) for efficient management of position or trajectory information (e.g. [ST94, DAS01, NGS+01, DKM+03]).

Depending on the application domain, different location or positioning sys-

tems[2] are used, ranging from global navigation satellite systems such as GPS [ME01] to a variety of indoor systems. Examples for the latter are RADAR [BP00], Cricket [SBGP04], and Smart Floor [OA00], which are based on signal strength measurements in WLANs, ultrasonic ranging, and pressure-sensitive floor tiles, respectively. See [HB01] for a survey of location systems and position sensing techniques.

Location services originated from distributed computing whereas MODs are an important topic in database research. Location services are particularly intended for managing the current positions of a large number of moving objects. Examples are the Globe Location Service [vSHHT98], the Nexus Location Service [LR02] and GeoGrid [ZZL07] – besides the location registers for mobile communications networks [PS01].

MODs, on the contrary, typically store entire trajectories. Two important research problems that have been addressed in the last decade are data modeling (e.g. [SWCD98, FGNS00, GdAD06]) and spatiotemporal indexing (e.g. [PJT00, HKTG06, PSJ06, NR07]). Research on location services and MODs converged in the last years. Protocols for tracking an object's current position, for instance, have been studied with location services as well as MODs (e.g. [WSCY99, LR01, LNR02, CJNP04]).

1.2 Problem Statement and Contributions

For economic and technical reasons, it is desirable to share context information by a variety of applications. For this purpose, several ontologies for semantic interoperability between providers of context information and applications have been developed (e.g. [RMCM03, CPFJ04, GWPZ04, BCQ+07, YCDN07]). Furthermore, a number of frameworks and systems for context management in mobile devices and smart environments have been proposed (e.g. [HSP+03, KMK+03, RHC+02, CFJ+04, WDC+04, GPZ05b]).

The proliferation of sensing technology, however, will afford millions of context models and providers to be shared by a wide variety of applications, as

[2] According to the ISO/IEC standard 19762-5:2008 [ISO-19762-5] such systems are called 'locating systems' but the terms 'positioning systems' and 'location systems' are much more popular, e.g. see [WHFG92, BP00, HB01, SBGP04, LCC+05, CL08].

discussed above. Obviously, these models have to be managed in a *distributed* fashion, to cope with the immense amounts of context information and their dynamism. Such context management poses a number of new challenges. A fundamental problem is how to provide efficient access to this distributed dynamic context information.

In the first instance, a scalable discovery approach for context models is needed, to be able to efficiently access context information about those aspects and areas of the physical world that are relevant for the situation of an application. Such discovery of context models comprises two subproblems:

1. *Formal describing of context models:* To decide whether a context model is relevant for the situation of an application, an adequate formalism for describing context models and for matching these descriptions against compatible queries is needed. The formalism has to be expressive enough to enable context providers to describe their offers of information in a concise manner – taking into consideration that there may exist thousands of context models providing information about a class or attribute of a given context ontology. Also the formalism has to facilitate alternative descriptions since many context models may be discovered by several characteristics such as location and identity.

2. *Indexing of context model descriptions:* For scalability, an appropriate index structure for context model descriptions is essential. It has to ensure that those descriptions matching a given query can be retrieved efficiently out of potentially millions of descriptions.

For the first subproblem, we propose a powerful formalism based on *defined classes*, i.e. logical expressions specifying sets of objects. This formalism is applicable not only to arbitrary context management systems but to any heterogeneous information system that is based on a shared ontology. It extends existing approaches for describing information sources by constraints considerably and permits to adjust between different semantics for matching descriptions against queries. For the second subproblem, we present the *Source Description Class Tree* (SDC-Tree). The SDC-Tree is a novel index structure and exploits the expressiveness of the proposed formalism completely – unlike

existing indexing approaches for ontology-based source discovery, which only consider the IS-A hierarchy of the shared ontology (e.g. [GPZ05a, KHM08]). Our evaluations show that the cost for searching one or few context models in den SDC-Tree depends logarithmically on the number of indexed descriptions.

Location information is of particular importance for context-aware computing since it may be used as index to access other context information [RDD+03]. The latter applies not only to mobile sensors but also to many other moving objects since context information is always bound to entities of the physical world [Dey00, DA00]. Therefore, moving objects databases (MODs) are used in context management systems to track and manage trajectory information about mobile objects such as sensors, hand-held devices, and vehicles.

This reveals another dimension of the considered fundamental problem. To provide efficient access to primary context information about moving objects or to other context information associated with their trajectories, scalable approaches for tracking and management of trajectories are needed. For this purpose, a number of tracking protocols and spatiotemporal index structures have been proposed (e.g. [WSCY99, PJT00, LR01, CJP05, GS05, HKTG06, NR07]). However, the need to manage the trajectories in real-time at one or more MODs – to enable access to current *and* past position information – and the large number of objects raises two subproblems beyond existing solutions:

3. *Efficient real-time trajectory tracking:* Many positioning systems (e.g. GPS) are based on sensors that are attached to the moving objects. Tracking the trajectories of such objects in real-time requires transmitting the position data over a wireless network to the MOD regularly. To minimize storage consumption and communication cost, efficient tracking protocols are needed that allow trading these costs off against data accuracy. Most existing protocols are designed for tracking the current position only and therefore neglect the storage consumption or do not generate a connected simplified trajectory at all.

4. *Distributed indexing of space-partitioned trajectories:* Managing a large number of trajectories may require partitioning the data to multiple database servers. A promising scheme is spatial partitioning since it

supports scalable processing of queries about single trajectories as well as of queries about all trajectory segments at a certain location. For efficient processing of the former class of queries, however, a distributed index structure over the partitions of each trajectory is needed.

In this thesis, we propose the *Connection-Preserving Dead Reckoning* (CDR) and *Generic Remote Real-Time Trajectory Simplification* (GRTS) protocols to address the third subproblem. CDR and GRTS optimize the storage consumption and communication cost for tracking a moving object's trajectory at a remote MOD under the constraint that the actual movement and the simplified trajectory known to the MOD do not deviate by more than a predefined accuracy bound. GRTS outperforms the only existing approach [TCS+06] by more than factor five in terms of reduction efficiency. Furthermore, it allows trading computational costs off against reduction efficiency as it can be combined with different line simplification algorithms (e.g. [DP73, II88, MdB04]). Even with a simple heuristic, the reduction performance of GRTS is only 10 to 14% worse than offline, retrospective reduction with an optimal line simplification algorithm.

For the fourth subproblem, we propose the *Distributed Trajectory Index* (DTI). It facilitates efficient routing of queries about single trajectories to the queried segments in space-partitioned MODs. Given that the database servers form a fully meshed geographic overlay network, the number of hops with DTI-based routing logarithmically depends on the temporal routing distance, i.e. the time span between the segment stored by the server that received the query and the queried segment. The extended scheme DTI+S further accelerates the processing of queries on aggregates of dynamic attributes (like the maximum speed during a time interval) by augmenting the index pointers with summaries of trajectory segments.

The major contributions of this thesis have been published at various international scientific conferences [LDR08a, LDR08b, LFDR09, LDR10b]. Accompanying publications are [FLR07, DPG+08, LCG+09, LWG+09, LDR10a], describing amongst others the practical use of the proposed concepts and algorithms.

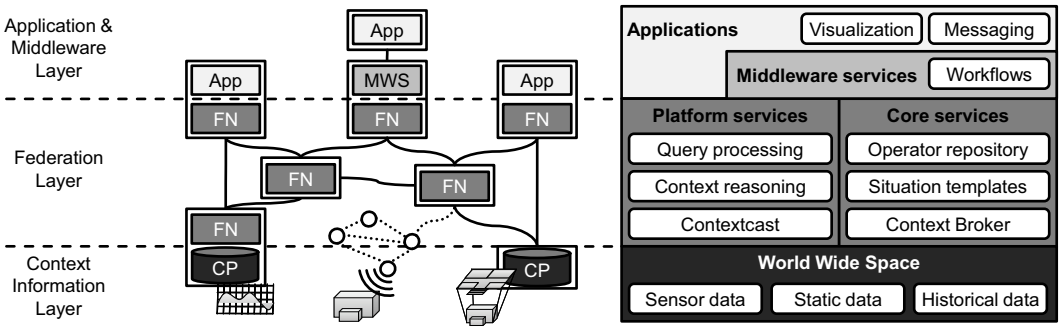


Figure 1.1: Architecture of the Nexus platform.

1.3 Background: The Nexus Platform

This thesis has been carried out within the Collaborative Research Center 627 *Nexus – Spatial World Models for Mobile Context-Aware Applications*, which traces back to [HKL+99].

Within the research center, the conceptual and technological framework for integrating and sharing context models of commercial and non-commercial providers at a global scale is researched. The Nexus platform federates these local context models and offers consistent views on the context information demanded by applications.

Nexus uses a three-layer architecture as illustrated in Figure 1.1, slightly adapted from [LCG+09]. Applications and middleware services are located on the top layer, while the context providers form the bottom layer. The distributed federation layer in the middle allows stream-based processing of complex spatial queries [CEB+09]. It additionally features situation recognition techniques [RHD+10] and context-aware communication [GDR09]. An important core service is the *Context-Broker* to discover those context providers that are relevant for a certain query or recognition task. The Context-Broker is based on the SDC-Tree proposed in Chapter 4.

Context information is expressed using the *Augmented World Modeling Language* based on a shared extendible ontology named *Standard Class Schema* [NM04]. This language is compatible with Web Ontology Language (OWL) [OWL] as shown in [NGMW08].

Within the research center, a scalable location service for managing the

current positions of moving objects has been proposed [LR02] together with protocols for efficient position tracking [LR01, LNR02]. The service consists of a number of location servers where each server manages a partition of the overall service area. These works are extended by the GRTS protocol and DTI, presented in the Chapters 2 and 3, for scalable tracking and management of trajectories.

1.4 Structure of the Thesis

The remainder of the thesis is structured following the general layering of context management systems – from the sensors at the bottom up to the applications at the top (e.g. [RHC+02,WDC+04,GPZ05b,LWG+09]). For example, in the Nexus platform, tracking protocols and moving objects databases belong to the Context Information Layer at the bottom, whereas the Context-Broker for discovering context models is located in the Federation Layer in the middle (cf. Figure 1.1). Therefore, we propose solutions for the third and fourth subproblems before we tackle the first two subproblems.

Chapter 2 first gives a formal description of the real-time trajectory tracking problem and an analysis of the only existing approach, based on the dead reckoning mechanism. We then present *Connection-Preserving Dead Reckoning* (CDR), which outperforms the existing approach by factor two in terms of reduction efficiency and communication cost, although it is also purely based on dead reckoning. Subsequently, we present the *Generic Remote Real-Time Trajectory Simplification* (GRTS) protocol, which clearly separates between tracking of the current position and simplification of the past trajectory. We propose several optimizations of GRTS and discuss realizations with different line simplification algorithms. We then give results from extensive simulations with real GPS traces and draw conclusions for the selection of a specific CDR or GRTS variant for a given application scenario. Finally, experiences with a prototypical implementation of GRTS and related work are discussed.

In Chapter 3, we present the *Distributed Trajectory Index* (DTI) together with the extended scheme DTI+S. We give the algorithms and protocols for the construction of DTI and DTI+S as well as for query routing and processing. We show the efficiency of the proposed scheme in simulative evaluations and

discuss related work.

In Chapter 4, we introduce the formalism for describing context models and for specifying compatible queries. We give the predicates for matching descriptions against queries before we present the *Source Description Class Tree* (SDC-Tree) and the corresponding algorithms. Furthermore, evaluation results from simulations of the SDC-Tree as well as related work are discussed.

Finally, a summary of the contributions and an outlook to promising future research topics are given in Chapter 5.

2 Efficient Real-Time Trajectory Tracking

In this chapter, we address the third subproblem of how to provide efficient access to distributed dynamic context information, namely the efficient real-time tracking of moving objects' trajectories.

We formalize this problem and discuss the two major aspects, namely to approximate an object's movement over time by a simplified trajectory with as few storage consumption as possible and to optimize the costs for communicating the trajectory information from the remote object to a MOD in real-time. Next, we present a first solution named *Connection-Preserving Dead Reckoning* (CDR), which is solely based on linear dead reckoning – a protocol designed for tracking the current positions of moving objects at low communication costs. Subsequently, we discuss the advantages of separating tracking of the current position from simplification of the past trajectory to gain flexibility and better reduction efficiency. Based on this finding, we propose the *Generic Remote Real-Time Trajectory Simplification* (GRTS) protocol, which can be combined with different line simplification algorithms to trade computational costs off against reduction efficiency. Different variants of GRTS as well as realizations with two well-known line simplification algorithms are discussed. We compare the different protocols and algorithms in a number of simulations with real GPS traces, draw conclusions for the selection of a concrete approach for a given application scenario, and describe a prototypical implementation of GRTS. Finally, related work is discussed and the chapter is concluded with a summary.

2.1 Preliminaries

As explained in Chapter 1, MODs are used in context-aware systems for managing trajectory information about mobile devices and other moving entities, which is a prerequisite for supporting information about their primary con-

text as well as for accessing other context information associated with their trajectories.

Generally, a moving object's trajectory is represented by a polyline in time and space where the vertices are the timestamped positions acquired by a suitable positioning system [LFG+03, MGA03, GS05]. Many of these systems (including GPS) are based on sensors that are attached to the moving objects. Tracking the trajectories of such objects therefore requires communicating the position data to the MOD using wireless communication.

Transmitting and storing every sensed position of an object's trajectory, however, causes high communication costs and generally consumes too much storage capacity. The former particularly applies if the MOD has to be informed in real-time about the object's movement, as required for many context-aware applications. For example, an ordinary GPS receiver may generate more than 30 million position data records per year, and in large-scale context-aware systems there may be thousands or millions of objects to be tracked.

Therefore, a tracking protocol is needed that allows trading these costs off against the accuracy of the trajectory information known to the MOD. With such a protocol, the MOD manages only a simplified trajectory that does not deviate by more than a certain accuracy bound ϵ from the actual movement, given by a subset of the sensed positions. We refer to this algorithmic problem as efficient (real-time) *trajectory tracking*. A formal definition is given below, at the end of Section 2.2.

Real-time trajectory tracking is related to line simplification, on the one hand, and protocols for tracking an object's current position, on the other hand. For clarity, we refer to the latter as (real-time) *position tracking* in the following.

Line simplification refers to a multitude of algorithmic problems on approximating a given polyline by a simplified one with fewer vertices. In the terminology of line simplification, trajectory tracking is referred to as min-# problem in the case of Hausdorff distance under the (time-)uniform distance metric in $\mathbb{R}^{1+d}$ with $d = 2$ or 3 [AV00, AHPMW05]. A straightforward approach for trajectory tracking using line simplification is to transmit the sensed position data from the tracked objects to the MOD and perform the simplification entirely on the MOD, for instance using the Douglas-Peucker algorithm [DP73]

as explained in [CWT06]. Obviously, such a solution does not optimize the communication cost since also those positions dropped later by simplification are transmitted over the wireless network.

Position tracking protocols, in contrast, aim to minimize the communication cost for informing the MOD about the current position of an object, but do not necessarily generate a simplified trajectory of the past movement. The latter particularly applies to the most efficient position tracking protocols, which are based on dead reckoning. With this technique, a tracked object initially transmits a function predicting its future movement to the MOD. This prediction function is updated at the MOD only if the object's locally sensed position impends to deviate from the predicted one by more than some accuracy bound ϵ. Consequently, only those sensing operations that require an adjustment of the prediction cause an update message to be sent. The most simple but nevertheless efficient variant is linear dead reckoning (LDR) [WSCY99, LR01, CJP05]. It uses a linear prediction given by a timestamped position and a velocity vector.

Dead reckoning does not generate a simplified trajectory since it describes the object's movement by a sequence of disconnected line sections – one for each prediction. Computing a connected trajectory on the basis of the linear predictions of LDR results in a simplified trajectory that may deviate from the actual movement by up to 2ϵ [TCS+06]. For trajectory tracking with accuracy bound ϵ, Trajcevski et al. therefore propose to use LDR with $\epsilon' := \frac{1}{2}\epsilon$, which we refer to as $\text{LDR}_{\frac{1}{2}}$. This approach, however, again causes unnecessary communication cost and consumes a lot of storage capacity.

In this chapter, we propose the *Generic Remote Real-Time Trajectory Simplification* (GRTS) protocol [LFDR09, LDR10a], which clearly separates tracking the current position from simplifying the past trajectory. GRTS also applies dead reckoning for tracking the current position, to optimize the number of messages sent over the wireless network, but can be combined with any line simplification algorithm suited for trajectories to reduce the past trajectory data.

There exist different solutions for line simplification, varying in reduction efficiency and computational overhead [DP73, II88, HS94, MdB04, AdBHZ07]. For example, an optimal line simplification algorithm provides the best re-

duction efficiency but causes the highest overhead, while solutions based on heuristics lower the computational overhead at the cost of smaller reduction rates. This flexibility allows applications to trade computational complexity off against reduction efficiency. To limit the computing time at the moving objects, we also present a space- and time-bounded variant named GRTS_{mc}.

We investigate two realizations of GRTS with different line simplification algorithms, namely the optimal line simplification algorithm introduced in [II88] and an efficient simplification heuristic [MdB04].

Our evaluations show that GRTS outperforms $\text{LDR}_{\frac{1}{2}}$ by a factor of five in terms of reduction efficiency. They also show that the number of vertices of the simplified trajectory obtained by $\text{GRTS}_{\text{mc}}^{\text{Opt}}$, i.e. the space- and time-bounded variant of GRTS realized with the optimal line simplification algorithm, may be less than 1% greater than the number of vertices of the best possible simplification computed by an optimal offline algorithm.

The separation of tracking of the current position from simplification of the past trajectory interestingly shows that the two goals of the trajectory tracking problem – to minimize communication cost and to minimize storage consumption – also contradict to some degree: With GRTS, the simplified trajectory may be revised over time, i.e. vertices may be replaced by later update messages, causing larger message sizes and thus communication cost. This effect increases, the higher the reduction rate achieved by the simplification algorithm.

Prior to GRTS, we therefore also propose *Connection-Preserving Dead Reckoning* (CDR) [LDR08a], which is solely based on dead reckoning as $\text{LDR}_{\frac{1}{2}}$, but outperforming $\text{LDR}_{\frac{1}{2}}$ by factor two. Although the reduction performance of CDR is significantly lower than the performance of GRTS, it transmits slightly less data than GRTS since it creates the simplified trajectory in an append-only fashion. Furthermore, the idea of a sensing history introduced with CDR is essential for GRTS.

Note already that Figure 2.26 in Section 2.7.5 gives an overview of all trajectory tracking approaches proposed in this thesis and depicts their relations.

2.2 Assumptions and Notation

We consider a collection of mobile objects with embedded positioning sensors (e.g. GPS receivers) whose trajectories are managed by a remote MOD. The objects and the MOD are connected by a wireless network. The overall number of trajectories stored by the MOD is of no relevance here.

An object's movement over time describes a continuous spatiotemporal function $\vec{a} : \mathbb{R} \mapsto \mathbb{R}^d$ from time to plane ($d = 2$) or space ($d = 3$) called the object's *actual trajectory*. Let t_C denote the current time, then $\vec{a}(t)$ is defined up to t_C and $\vec{a}(t_\mathrm{C})$ is the object's current actual position.

The positioning sensor periodically senses the object's current position with period T_S, referred to as *sensing period*. It results in a sequence of *sensed positions* $(s_1, s_2, \ldots, s_\mathrm{R})$, where s_1 denotes the first and s_R the most recent sensed position. Each s_i is a data record consisting of the sensing time t and the sensed position $\vec{p}$, denoted by $s_i.t$ and $s_i.\vec{p}$, respectively.

The sensed positions define the *sensed trajectory* $\vec{s}(t)$, a continuous, piecewise linear function, as follows: Two consecutive positions s_i and s_{i+1} define a *spatiotemporal line section* $\overline{s_i\, s_{i+1}}$ on the domain $[s_i.t, s_{i+1}.t]$ as

$$\overline{s_i\, s_{i+1}} : t \mapsto \frac{(s_{i+1}.t - t)\, s_i.\vec{p} + (t - s_i.t)\, s_{i+1}.\vec{p}}{s_{i+1}.t - s_i.t} \ .$$

Then, $\vec{s}(t)$ is defined by the sequence $(s_1, s_2, \ldots, s_\mathrm{R})$ on the domain $[s_1.t, s_\mathrm{R}.t]$ as

$$\vec{s} : t \mapsto \overline{s_i\, s_{i+1}}(t) \text{ where } s_i.t \leq t \leq s_{i+1}.t \ .$$

Geometrically, $\vec{s}(t)$ is a time-monotonous polyline in $\mathbb{R}^{1+d}$ given by the sequence of vertices $(s_1, s_2, \ldots, s_\mathrm{R})$.

Note that the domain $[s_1.t, s_\mathrm{R}.t]$ does not continuously increase over time but periodically by T_S, with each sensing operation. For current time t_C, it holds $t_\mathrm{C} - T_\mathrm{S} < s_\mathrm{R}.t \leq t_\mathrm{C}$.

$\vec{s}(t)$ generally deviates from $\vec{a}(t)$ due to inaccuracies of the positioning sensor and the time-discrete sensing. The former are generally described by stochastic means such as probability density functions or percentiles, which allow deriving a *maximum sensor inaccuracy* σ that holds with high probability. Inaccuracies

beyond σ – typically indicated by erratic positions – are considered as errors, which have to be treated separately. Regarding the time discretization by position sensing, the movement between two sensing operations is subject to physical constraints like the maximum speed or acceleration.

Therefore, we assume that the deviation between $\vec{s}(t)$ and $\vec{a}(t)$ is bounded by a certain *maximum sensing deviation* δ, i.e. $\forall\, t' \in [s_1.t, s_\mathrm{R}.t]$ it holds $|\vec{s}(t') - \vec{a}(t')| \leq \delta$. For example, given a maximum speed v_max, we can conclude that

$$|\vec{a}(t') - \vec{s}(t')| \leq \sigma + v_\mathrm{max}\frac{T_\mathrm{S}}{2} =: \delta \ ,$$

since the object cannot move more than $v_\mathrm{max} \cdot T_\mathrm{S}/2$ and then return to its origin during a sensing period T_S.

The use of speed-based movement constraints to estimate the deviation between $\vec{s}(t)$ and $\vec{a}(t)$ is discussed in detail in [PJ99]. In Section 2.6, we show how to incorporate acceleration-based constraints, which typically afford smaller values of δ.

The sensor inaccuracy may depend on dynamic technical conditions such as the satellite constellation of GPS described by the dilution of precision (DOP). Therefore, σ may be time-dependent, and thus δ. For simplicity, we assume σ and δ to be fixed at first. In Section 2.5.4, we discuss how to account for time-dependent values of σ and δ, given with each sensed position.

Note that the physical movement constraints also allow estimating $\vec{a}(t')$ for $t' > s_\mathrm{R}.t$. For example, given a maximum speed v_max, the actual position $\vec{a}(t')$ is known to be inside a circle with radius $\sigma + v_\mathrm{max}(t' - s_\mathrm{R}.t)$ around $s_\mathrm{R}.\vec{p}$. This property is utilized by dead reckoning as explained in Section 2.3.

In this regard, we also assume that the time for processing and transmitting an update message to the MOD is bounded by the *update time* T_U. Exceptional transmission delays and connection breaks are considered as errors and have to be detected by subsidiary mechanisms such as heartbeat messages.

The MOD describes the object's trajectory by a continuous, piecewise linear function $\vec{u} : t \mapsto \mathbb{R}^d$ called *simplified trajectory*. Geometrically, $\vec{u}(t)$ is a time-monotonous polyline in $\mathbb{R}^{1+d}$ given by a sequence of vertices $(u_1, u_2, \ldots)$ like $\vec{s}(t)$. Each vertex u_i is a data record with attributes t and $\vec{p}$, just as a sensed position.

We refer to any clipping of $\vec{a}(t)$, $\vec{s}(t)$, or $\vec{u}(t)$ given by an arbitrary time interval or a subsequence of vertices as *trajectory segment*.

With this notation, the algorithm problem of tracking a moving object's trajectory efficiently in real-time can be formally stated as follows:

Definition 1 (Efficient real-time trajectory tracking): The goals are to minimize the number of vertices of the simplified trajectory $\vec{u}(t)$ as well as the amount of data transmitted over the wireless network under the following two constraints, where t_C denotes the current time:

1. *Simplification constraint:* For a certain *accuracy bound* ϵ known by the moving object and the MOD, it holds

$$\forall\, t' \in [s_1.t, t_C] \,:\, |\vec{u}(t') - \vec{a}(t')| \leq \epsilon \ .$$

2. *Real-time constraint:* At t_C, position $\vec{u}(t)$ is available at the MOD for every $t \in [s_1.t, t_C]$.

2.3 Analysis of Linear Dead Reckoning

Before we present our approaches CDR and GRTS, we analyze the use of linear dead reckoning (LDR) for trajectory tracking – also because CDR and GRTS make use of LDR.

As explained above, LDR is an efficient mechanism for tracking the current position of a moving object with low communication costs [WSCY99, LR01, CJP05]. With LDR, the moving object and the MOD share a linear prediction function $\vec{l}(t)$ for determining the object's current position. $\vec{l}(t)$ is defined by a previously sensed position l_O called *prediction origin* and a *velocity vector* $\vec{l}_V$ as

$$\vec{l} : t \mapsto l_O.\vec{p} + (t - l_O.t)\vec{l}_V \ .$$

For a given accuracy bound ϵ, the LDR protocol guarantees that $\vec{l}(t)$ known by the MOD approximates the objects' current actual position by ϵ. Formally, at current time t_C, it guarantees that $|\vec{l}(t_C) - \vec{a}(t_C)| \leq \epsilon$. For this purpose, LDR has to send a new prediction $(l_O, \vec{l}_V)$ to the MOD as soon as $|\vec{l}(t_C) - \vec{a}(t_C)|$

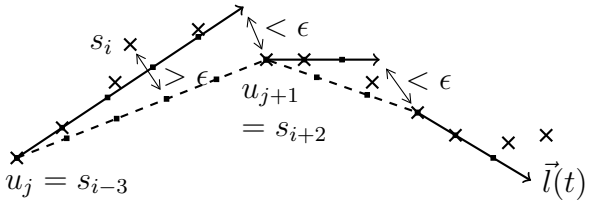


Figure 2.1: Example of a violation of ϵ for trajectory tracking by LDR.

impends to reach ϵ, taking into account the inaccuracy σ of the positioning sensor and the possible movement during the next sensing period T_S and the update time T_U for processing and transmitting an update message. For example, assuming a maximum speed v_max, the object has to send an update if

$$|s_\mathrm{R}.\vec{p} - \vec{l}(s_\mathrm{R}.t + T_\mathrm{S} + T_\mathrm{U})| + \sigma + v_\mathrm{max}(T_\mathrm{S} + T_\mathrm{U}) > \epsilon$$

since $s_\mathrm{R}.\vec{p}$ may deviate by up to σ from the actual position at $s_\mathrm{R}.t$ and the object may move by up to $v_\mathrm{max}(T_\mathrm{S}+T_\mathrm{U})$ until an update sent after the subsequent sensing operation would have been received by the MOD.[1]

LDR is not a trajectory tracking protocol but only a position tracking protocol because of the discontinuities between the different predictions. More precisely, it represents the object's movement by a sequence of disconnected spatiotemporal line sections instead of a continuous polyline. Figure 2.1 illustrates such discontinuities. The solid arrows denote the linear predictions – and thus the line sections – whereas the small crosses indicate the sensed positions.

However, the distance between the end point of such a line section and the start point of the subsequent one is bounded by ϵ.

In [TCS+06], Trajcevski et al. utilize this property for trajectory tracking. They analyze the spatiotemporal polyline given by the origins of the linear predictions and prove that it approximates the actual movement by 2ϵ. Figure 2.1 illustrates this polyline by a dashed line. Based on this finding, they conclude that LDR allows for trajectory tracking with accuracy bound ϵ as

[1] Note that many works do not clearly state whether they account for the sensing period and update time, or not. Some works even ignore the sensor inaccuracy σ since it can be initially offset against ϵ, as long as it does not vary over time.

follows:

1. The moving object reports its current position using LDR with the accuracy bound $\epsilon' := \frac{1}{2}\epsilon$.

2. The MOD not only stores the current prediction, but also the origins of all previous predictions as vertices of the simplified trajectory $\vec{u}(t)$.

In the following, we refer to this approach as $\text{LDR}_{\frac{1}{2}}$. We argue that $\text{LDR}_{\frac{1}{2}}$ is very conservative, which can be seen from the polyline $\vec{u}(t)$ given by the prediction origins of LDR with accuracy bound ϵ. This polyline may deviate by more than ϵ from the actual trajectory $\vec{a}(t)$, as just explained. However, such violations occur seldom and rarely get close to 2ϵ for the following reasons:

1. The actual position $\vec{a}(t')$ at time t' can only deviate by more than ϵ from the corresponding line section $\overline{u_j\, u_{j+1}}$ if $\vec{a}(t')$ and u_{j+1} are located in opposite directions from the predicted movement. This requires the object to first deviate in one direction and then in the opposite direction from the same prediction, as illustrated in Figure 2.1 with the sensed positions s_i and s_{i+2}. Such opposite directions, however, do not occur with typical movement patterns like turning off or stopping.

2. A violation $|\vec{a}(t') - \overline{u_j\, u_{j+1}}| > \epsilon$ can be close to 2ϵ only if the opposite deviations from the prediction occur promptly one after another, which implies $t' \approx u_{j+1}.t$. Thus, for violations close to 2ϵ, the object has to move in a very fast and irregular fashion.

To support this argumentation, we analyzed the potential of such violations by simulating LDR with a GPS trace of a 4-hour bicycle tour from the OpenStreetMap [OSM] project. For the simulation, we used $\sigma = 7.8\,\text{m}$, $v_{\max} = 10\,\text{m/s}$, $T_\text{S} = 1\,\text{s}$, $T_\text{U} = 0.2\,\text{s}$, and $\epsilon = 50\,\text{m}$.

With these assumptions and parameters, LDR generates 396 position updates, where the last one indicates the end of the tour. Thus, the resulting simplified trajectory $\vec{u}(t)$ consists of 395 line sections given by 396 vertices. Figure 2.2 illustrates the deviations between the sensed positions s_i and $\vec{u}(t)$ depending on time. Each cross denotes a sensed position s_i with

$$|s_i.\vec{p} - \vec{u}(s_i.t)| > \epsilon - \delta = 37.2\,\text{m}\ ,$$

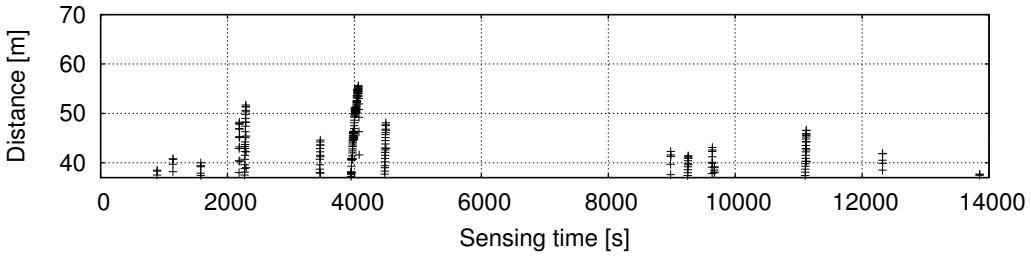


Figure 2.2: Potential violations of ϵ during a 4-hour bicycle tour for trajectory tracking by LDR.

i.e. where the deviation between $\vec{a}(t)$ and $\vec{u}(t)$ may violate or certainly violates the accuracy bound ϵ. Obviously, the violations are not distributed uniformly over time but appear at few line sections of $\vec{u}(t)$ only, namely at 15 of the 395 line sections. Furthermore, the deviations are well below 2ϵ.

We conclude that using LDR with $\epsilon' := \frac{1}{2}\epsilon$, i.e. $\text{LDR}_{\frac{1}{2}}$, is generally too strict for preventing violations of ϵ. It generates needless position updates and hence simplified trajectories with unnecessary large numbers of vertices.

Next, we present an approach for trajectory tracking that extends LDR to prevent such violations before we present the GRTS protocol, which clearly separates tracking of the current position from simplification of the past trajectory.

2.4 Connection-Preserving Dead Reckoning

In this section, we propose *Connection-Preserving Dead Reckoning* (CDR), which extends LDR such that the prediction origins make up the vertices of a simplified trajectory $\vec{u}(t)$ that approximates $\vec{a}(t)$ by the accuracy bound of LDR. First, we present the basic CDR algorithm executed at the moving object before we discuss an optimization of this algorithm. Then, we present a space- and time-bounded variant of this algorithm named CDR_m.

2.4.1 Basic Version of CDR

As just explained in Section 2.3, the prediction origins of LDR with accuracy bound ϵ make up a simplified trajectory $\vec{u}(t)$ that approximates $\vec{a}(t)$ by ϵ for most of the time. However, some line sections of $\vec{u}(t)$ may violate ϵ.

CDR is based on the observation that the moving object has all information for detecting such violations in real-time:

1. It knows the current prediction given by the prediction origin l_{O} and the velocity vector $\vec{l}_{\mathrm{V}}$.

2. It knows the most recent sensed position s_{R}.

3. Thus, it also knows the resulting line section $\overline{l_{\mathrm{O}}\, s_{\mathrm{R}}}$ of $\vec{u}(t)$, in case s_{R} is used as origin of the next prediction.

4. It can store the *sensing history* since the last update message, i.e. the positions that have been sensed after $l_{\mathrm{O}}.t$, and thus check whether one of these positions deviates from $\overline{l_{\mathrm{O}}\, s_{\mathrm{R}}}$ by more than $\epsilon - \delta$. If so, the line section $\overline{l_{\mathrm{O}}\, s_{\mathrm{R}}}$ may deviate by more than ϵ from $\vec{a}(t)$. In the following we refer to the sensing history as $\mathbb{S} := \{s_i : s_i.t > l_{\mathrm{O}}.t\}$.

The basic idea of CDR is that the moving object not only sends a new position update if caused by LDR, but also if one of the sensed positions since the last update message deviates from $\overline{l_{\mathrm{O}}\, s_{\mathrm{R}}}$ by more than $\epsilon - \delta$.

Figure 2.3 shows the pseudo code of the algorithm executed by moving object. A crucial difference to LDR is that CDR maintains a dynamic array that stores the sensing history (line 5). Another, subtle difference to LDR is that CDR does not use the most recent sensed position s_{R} as origin of a new prediction, but the one before s_{R}, denoted by $s_{\mathrm{R}'}$ (line 10). The use of $s_{\mathrm{R}'}$ has negligible or no influence on LDR since the predicted velocity $\vec{l}_{\mathrm{V}}$ generally is determined by means of the last sensed positions and particularly $(s_{\mathrm{R}}.\vec{p} - s_{\mathrm{R}'}.\vec{p})/(s_{\mathrm{R}}.t - s_{\mathrm{R}'}.t)$. Yet, it is essential for $\vec{u}(t)$ as explained below.

Initially the moving object transmits its current position and the zero vector as velocity prediction to the MOD. Then it executes the while loop (lines 7 to 17) as long as it wants to report its movement to the MOD.

```
 1: s_R ← sense position                                   ▷ Most recent sensed position.
 2: l_O ← s_R                                              ▷ Prediction origin.
 3: v⃗_V ← 0                                               ▷ Predicted velocity.
 4: send update message (l_O, v⃗_V) to MOD
 5: 𝕊 ← {}                                                 ▷ Sensing history since last update.
 6: s_R' ← s_R                                             ▷ Second last sensed position.
 7: while report movement do
 8:     s_R ← sense position
 9:     if LDR causes update or ∃ s_i ∈ 𝕊 : |s_i − l_O s_R(s_i.t)| > ε − δ then
10:         l_O ← s_R'
11:         v⃗_V ← compute new predicted velocity ...
12:         send update message (l_O, v⃗_V) to MOD
13:         𝕊 ← {}                                        ▷ Clear the sensing history.
14:     end if
15:     𝕊 ← 𝕊 ∪ {s_R}                                     ▷ Add s_R to sensing history.
16:     s_R' ← s_R
17: end while
18: send final update message (s_R) to MOD
```

Figure 2.3: Basic version of CDR algorithm.

During each iteration, it first senses its current position (line 8) and then checks whether LDR causes an update or whether the *section condition*, given as

$$\forall\, s_i \in \mathbb{S} : |s_i - \overline{l_\mathrm{O}\, s_\mathrm{R}}(s_i.t)| \leq \epsilon - \delta \ ,$$

is violated (line 9). The section condition simply states that none of the sensed positions s_i since the last update should deviate from $\overline{l_\mathrm{O}\, s_\mathrm{R}}$ by more than $\epsilon - \delta$, as discussed above. If there exists an s_i deviating by more than $\epsilon - \delta$, a new update message with prediction origin $l_\mathrm{O} = s_{\mathrm{R}'}$ is sent to the MOD such that the corresponding line section of the simplified trajectory $\vec{u}(t)$ fulfills the section condition.

After sending an update message, $\mathbb{S}$ is cleared to remove the sensed positions before $l_\mathrm{O}.t$ (line 13).

If the moving object wants to stop reporting its movement, it sends a fi-

nal update message with the most recent sensed position and terminates the algorithm (line 18).

The simplified trajectory $\vec{u}(t)$ managed by the MOD consists of two parts: the spatiotemporal polyline given by the vertices $(u_1, \ldots, u_m)$, as described in Section 2.2, and the prediction function $\vec{l}(t)$ of LDR. On receiving an update message $(l_\mathrm{O}, \vec{l}_\mathrm{V})$, the MOD simply updates $\vec{l}(t)$ with the new origin l_O and the new velocity vector $\vec{l}_\mathrm{V}$ and appends l_O to the sequence of vertices as $(m+1)$th vertex.

Given a query for the moving object's position at time t', the MOD answers as follows:

- $t' \leq l_\mathrm{O}.t$ *:* The MOD calculates $\vec{u}(t')$ as described in Section 2.2 and returns the result to the query issuer.
- $t' > l_\mathrm{O}.t$ *:* It calculates the predicted position at time t' using $\vec{l}(t') = l_\mathrm{O}.\vec{p} + (t' - l_\mathrm{O}.t)\vec{l}_\mathrm{V}$ and returns the result to the query issuer.

If the MOD receives the final update message (s_R) it removes $\vec{l}(t)$ and completes $\vec{u}(t)$ by appending s_R as final vertex to $(u_1, \ldots, u_m)$.

2.4.2 Optimization of Sensing History

CDR differs from LDR regarding the space requirements at the moving object. While LDR only stores the current prediction and the most recent sensed position, the basic version of CDR stores the whole sensing history $\mathbb{S}$ since the last update. Theoretically, the size of $\mathbb{S}$ is unbounded.

However, this problem can be alleviated. For every sensed position $s_i \in \mathbb{S}$ there exists a certain point in time from that on it cannot violate the section condition without s_R causing LDR to send an update. After this time, s_i can be removed from $\mathbb{S}$, even before the next update. This significantly reduces the space consumption of CDR as well as the computing time per position fix. For example, in case of the bicycle tour mentioned in Section 2.3 and $\epsilon = 50\,\mathrm{m}$ the maximum size of $\mathbb{S}$ is reduced from 405 to 214 positions. The maximum computing time per position fix – here measured in processor ticks, i.e. clock cycles of the CPU (cf. Section 2.7) – is reduced from 312390 to 182060 ticks.

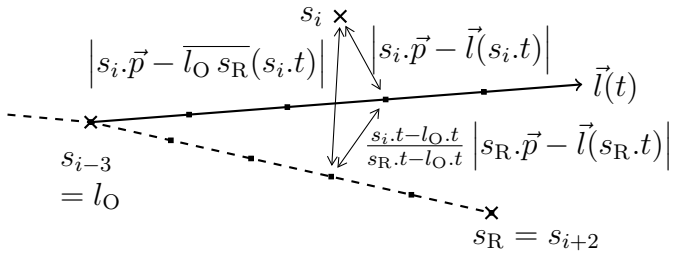


Figure 2.4: Geometric illustration of triangle inequality for optimization of $\mathbb{S}$.

To determine this point in time for a given $s_i \in \mathbb{S}$, we analyze the state of the basic version of CDR (cf. Figure 2.3) right after having sensed a new position s_R (line 8) that does not cause an update by LDR.

We consider the line section $\overline{l_O\, s_R}$, which is going to be the next line section $\overline{u_m\, u_{m+1}}$ of $\vec{u}(t)$ if an update is sent in the subsequent iteration of the while loop.

Since the prediction function $\vec{l}(t)$ and $\overline{l_O\, s_R}$ are linear functions in t with identical origin l_O, we conclude that regarding time $s_i.t$ they deviate by

$$\frac{s_i.t - l_O.t}{s_R.t - l_O.t}\left|s_R.\vec{p} - \vec{l}(s_R.t)\right| \ .$$

As illustrated in Figure 2.4, it follows the triangle inequality

$$\left|s_i.\vec{p} - \overline{l_O\, s_R}(s_i.t)\right| \leq \left|s_i.\vec{p} - \vec{l}(s_i.t)\right| + \frac{s_i.t - l_O.t}{s_R.t - l_O.t}\left|s_R.\vec{p} - \vec{l}(s_R.t)\right| \ . \qquad (2.1)$$

LDR certainly sends an update if $|s_R.\vec{p} - \vec{l}(s_R.t)|$ is equal to ϵ. Under the above assumption that s_R does not cause an update, we safely conclude that $|s_R.\vec{p} - \vec{l}(s_R.t)| \leq \epsilon$ and finally estimate $\left|s_i.\vec{p} - \overline{l_O\, s_R}(s_i.t)\right|$ by

$$\left|s_i.\vec{p} - \overline{l_O\, s_R}(s_i.t)\right| \leq \left|s_i.\vec{p} - \vec{l}(s_i t.)\right| + \frac{s_i.t - l_O.t}{s_R.t - l_O.t}\epsilon \ .$$

Clearly, this estimate decreases over time, i.e. with increasing values of $s_R.t$. We now derive the point in time from that on it falls below $\epsilon - \delta$, as required

```
1: [...]
2: while report movement do
3:     s_R ← sense position
4:     while |𝕊| > 0 and s_R.t ≥ κ(peek(𝕊)) do
5:         pop(𝕊)                                  ▷ Remove root of heap.
6:     end while
7:     [...]
8: end while
9: send final update message (s_R) to MOD
```

Figure 2.5: Optimization of $\mathbb{S}$ in the CDR algorithm.

to fulfill the section condition definitely:

$$\epsilon - \delta \geq \left| s_i.\vec{p} - \vec{l}(s_i.t) \right| + \frac{s_i.t - l_\mathrm{O}.t}{s_\mathrm{R}.t - l_\mathrm{O}.t}\epsilon$$

$$\Leftrightarrow \frac{\epsilon - \delta - \left| s_i.\vec{p} - \vec{l}(s_i.t) \right|}{s_i.t - l_\mathrm{O}.t} \geq \frac{\epsilon}{s_\mathrm{R}.t - l_\mathrm{O}.t} \tag{2.2}$$

$$\Leftrightarrow s_\mathrm{R}.t \geq \underbrace{\frac{s_i.t - l_\mathrm{O}.t}{\epsilon - \delta - \left| s_i.\vec{p} - \vec{l}(s_i.t) \right|}\epsilon + l_\mathrm{O}.t}_{=:\kappa(s_i)} \tag{2.3}$$

Thus, s_i cannot violate the section condition once time $s_\mathrm{R}.t$ fulfills the inequation (2.3).

So far, we assumed that s_R does not cause an update by LDR. In the general case, it holds that, once $s_\mathrm{R}.t$ fulfills (2.3), s_i cannot violate the section condition without s_R causing an update by LDR. Therefore, CDR can remove s_i from $\mathbb{S}$ at this point in time without affecting its future decisions on a new update.

For this purpose, CDR organizes $\mathbb{S}$ as a min-heap according to the right-hand side of (2.3), i.e. $\kappa(s_i)$. After position sensing, it first removes the root of $\mathbb{S}$ one by one, as long as this sensed position fulfills (2.3). Figure 2.5 shows the corresponding additional pseudo code to the basic version of CDR.

2.4.3 Space- and Time-bounded CDR

With the optimization presented above, CDR tries to reduce the sensing history $\mathbb{S}$ after each position fix. Nevertheless, the space consumption of CDR is not bounded, which can be critical for resource-constrained mobile devices.

In the following, we present the CDR_m algorithm whose space consumption is bounded by a predefined parameter m. CDR_m guarantees that $|\mathbb{S}| \leq m$ at every point in time. This also limits the computing time per position fix.

CDR_m is based on the following idea: Besides a heap of fixed size m for storing $\mathbb{S}$, it maintains a floating-point variable $d_\mathbb{S}$ providing aggregated information on all sensed positions that could not be stored in $\mathbb{S}$ due to the space constraint. More precisely, $d_\mathbb{S}$ defines a time-dependent bound for $\left|s_\text{R}.\vec{p} - \vec{l}(s_\text{R}.t)\right|$. Each time $|\mathbb{S}|$ is going to exceed m, the CDR_m algorithm removes a sensed position from $\mathbb{S}$ and updates $d_\mathbb{S}$ accordingly.

If $\left|s_\text{R}.\vec{p} - \vec{l}(s_\text{R}.t)\right|$ is below the bound defined by $d_\mathbb{S}$, none of the sensed positions that could not be stored in $\mathbb{S}$ violates the section condition for the most recent sensed position.

For this purpose, the section condition is split into two subconditions, respectively. The first subcondition is evaluated on the sensed positions currently stored in $\mathbb{S}$, just as with CDR. The second subcondition is evaluated on $d_\mathbb{S}$.

We now give the mathematical basis for $d_\mathbb{S}$ and derive the inequation for the second subcondition. First, we reconsider the triangle inequality (2.1), given in Section 2.4.2,

$$\left|s_i.\vec{p} - \overline{l_\text{O}\, s_\text{R}}(s_i.t)\right| \leq \left|s_i.\vec{p} - \vec{l}(s_i.t)\right| + \frac{s_i.t - l_\text{O}.t}{s_\text{R}.t - l_\text{O}.t}\left|s_\text{R}.\vec{p} - \vec{l}(s_\text{R}.t)\right| \ .$$

With it, we conclude that

$$\left|s_i.\vec{p} - \vec{l}(s_i.t)\right| + \frac{s_i.t - l_\text{O}.t}{s_\text{R}.t - l_\text{O}.t}\left|s_\text{R}.\vec{p} - \vec{l}(s_\text{R}.t)\right| \leq \epsilon - \delta \tag{2.4}$$

implies $\left|s_i.\vec{p} - \overline{l_\text{O}\, s_\text{R}}(s_i.t)\right| \leq \epsilon - \delta$.

Inequation 2.4 can be rewritten as

$$\left|s_{\mathrm{R}}.\vec{p} - \vec{l}(s_{\mathrm{R}}.t)\right| \leq \underbrace{\frac{\epsilon - \delta - \left|s_i.\vec{p} - \vec{l}(s_i.t)\right|}{s_i.t - l_{\mathrm{O}}.t}}_{=:\,\varphi(s_i)}(s_{\mathrm{R}}.t - l_{\mathrm{O}}.t) \ .$$

Thus, $\left|s_{\mathrm{R}}.\vec{p} - \vec{l}(s_{\mathrm{R}}.t)\right| \leq \varphi(s_i) \cdot (s_{\mathrm{R}}.t - l_{\mathrm{O}}.t)$ implies that s_i does not violate the section condition. This result is used for the $\mathrm{CDR_m}$ algorithm as follows:

1. The minimum $\varphi(s_j)$ of all sensed positions s_j that had to be removed from $\mathbb{S}$ due to the space constraint is stored in the variable $d_{\mathbb{S}}$.
2. Inequation $\left|s_{\mathrm{R}}.\vec{p} - \vec{l}(s_{\mathrm{R}}.t)\right| \leq d_{\mathbb{S}} \cdot (s_{\mathrm{R}}.t - l_{\mathrm{O}}.t)$ is used as second subcondition of the section condition. Thus, an update is sent if $\left|s_{\mathrm{R}}.\vec{p} - \vec{l}(s_{\mathrm{R}}.t)\right|$ exceeds $d_{\mathbb{S}} \cdot (s_{\mathrm{R}}.t - l_{\mathrm{O}}.t)$.

Therefore, as long as the second subcondition is fulfilled, each removed position s_j fulfills the section condition. Thus, as long as both subconditions are fulfilled, every sensed position since the last position update fulfills the section condition. After an update, $\mathbb{S}$ is cleared – just as with CDR – and $d_{\mathbb{S}}$ is reset to ∞.

A crucial question is which sensed position to remove from $\mathbb{S}$ once $|\mathbb{S}|$ is going to exceed m. Clearly, for small values of $d_{\mathbb{S}}$, the most recent sensed position s_{R} violates the second subcondition more likely. Therefore, $\mathrm{CDR_m}$ always removes the $s_j \in \mathbb{S}$ with maximum $\varphi(s_j)$. For this purpose it stores $\mathbb{S}$ as a max-heap according to $\varphi(s_i)$.

Since $d_{\mathbb{S}}$ aggregates all previously sensed positions with $\varphi(s_j) \geq d_{\mathbb{S}}$, the most recent sensed position s_{R} need not be added to $\mathbb{S}$ if $\varphi(s_{\mathrm{R}}) \geq d_{\mathbb{S}}$. Hence, the invariant $\forall s_i \in \mathbb{S} : \varphi(s_i) \leq d_{\mathbb{S}}$ holds for $\mathbb{S}$. For this reason, $\mathrm{CDR_m}$ can directly assign $\varphi(\mathrm{pop}(\mathbb{S}))$ to $d_{\mathbb{S}}$ when removing the root of $\mathbb{S}$. It does not need to determine the minimum of $\varphi(\mathrm{pop}(\mathbb{S}))$ and $d_{\mathbb{S}}$ explicitly.

Note that the max-heap order by $\varphi(s_i)$ of $\mathrm{CDR_m}$ is identical to the min-heap order by $\kappa(s_i)$ of CDR since $\kappa(s_i) = l_{\mathrm{O}}.t + \epsilon/\varphi(s_i)$.

This can be seen by comparing the inequations (2.2) and (2.3) in Section 2.4.2. Moreover, CDR's condition $s_{\mathrm{R}}.t \geq \kappa(\mathrm{peek}(\mathbb{S}))$ for removing a s_i from $\mathbb{S}$ can be rewritten as $\varphi(\mathrm{peek}(\mathbb{S})) \cdot (s_{\mathrm{R}}.t - l_{\mathrm{O}}.t) \geq \epsilon$.

1: $s_{\mathrm{R}} \leftarrow$ sense position ▷ Most recent sensed position.
2: $l_{\mathrm{O}} \leftarrow s_{\mathrm{R}}$ ▷ Prediction origin.
3: $\vec{l}_{\mathrm{V}} \leftarrow 0$ ▷ Predicted velocity.
4: send update message $(l_{\mathrm{O}}, \vec{l}_{\mathrm{V}})$ to MOD
5: $\mathbb{S} \leftarrow \{\}$ ▷ Sensing history since last update.
6: $d_{\mathbb{S}} \leftarrow \infty$ ▷ Indicates empty aggregation.
7: $s_{\mathrm{R}'} \leftarrow s_{\mathrm{R}}$ ▷ Second last sensed position.
8: **while** report movement **do**
9: $s_{\mathrm{R}} \leftarrow$ sense position
10: **while** $|\mathbb{S}| > 0$ **and** $\varphi(\mathrm{peek}(\mathbb{S})) \cdot (s_{\mathrm{R}}.t - l_{\mathrm{O}}.t) \geq \epsilon$ **do**
11: pop($\mathbb{S}$) ▷ Remove root of heap.
12: **end while**
13: **if** LDR causes update **or** $\exists\, s_i \in \mathbb{S} : |s_i - \overline{l_{\mathrm{O}}\, s_{\mathrm{R}}}(s_i.t)| > \epsilon - \delta$ **or** $|s_{\mathrm{R}}.\vec{p} - \vec{l}(s_{\mathrm{R}}.t)| > d_{\mathbb{S}} \cdot (s_{\mathrm{R}}.t - l_{\mathrm{O}}.t)$ **then**
14: $l_{\mathrm{O}} \leftarrow s_{\mathrm{R}'}$
15: $\vec{l}_{\mathrm{V}} \leftarrow$ compute new predicted velocity ...
16: send update message $(l_{\mathrm{O}}, \vec{l}_{\mathrm{V}})$ to MOD
17: $\mathbb{S} \leftarrow \{\}$ ▷ Clear the sensing history.
18: $d_{\mathbb{S}} \leftarrow \infty$ ▷ Reset the bound.
19: **end if**
20: **if** $|\mathbb{S}| = m$ **and** $\varphi(s_{\mathrm{R}}) < d_{\mathbb{S}}$ **then**
21: $d_{\mathbb{S}} \leftarrow \varphi(\mathrm{pop}(\mathbb{S}))$ ▷ Remove and aggregate the root.
22: **end if**
23: **if** $\varphi(s_{\mathrm{R}}) < d_{\mathbb{S}}$ **then**
24: insert s_{R} into $\mathbb{S}$
25: **end if**
26: $s_{\mathrm{R}'} \leftarrow s_{\mathrm{R}}$
27: **end while**
28: send final update message (s_{R}) to MOD

Figure 2.6: $\mathrm{CDR_m}$ algorithm.

From an algorithmic perspective, $\mathrm{CDR_m}$ is thus an extension of CDR. Figure 2.6 gives the pseudo code of $\mathrm{CDR_m}$. The additional statements compared to CDR are the following ones:

- *Lines 6 and 18:* Initialize or reset $d_{\mathbb{S}}$, respectively.

- *Lines 20 to 22:* Remove the sensed position with maximum $\varphi(s_i)$ from $\mathbb{S}$ and aggregate it in $d_{\mathbb{S}}$ if $|\mathbb{S}| = m$ and s_{R} has to be added to $\mathbb{S}$.

- *Lines 23 to 25:* Insert the most recent sensed position s_{R} into the heap $\mathbb{S}$ if $\varphi(s_{\mathrm{R}}) < d_{\mathbb{S}}$.

The computing time per position fix of $\mathrm{CDR_m}$ is dominated by two inner loops: First, removing those sensed positions from $\mathbb{S}$ that cannot any more violate the section condition without s_{R} causing LDR to send an update (lines 10 to 12). Second, checking the validity of the first subcondition of the section condition, indicated by the existential quantifier in line 13. The computing time of all other statements either is constant or depends logarithmically on m, such as the pop operation on the heap $\mathbb{S}$.

The computing time for checking the validity of the first subcondition is a linear function in m, i.e. it is in $\mathcal{O}(m)$.

The removal of positions from $\mathbb{S}$ according to the optimization of CDR, however, is more complex: Under normal circumstances, only few positions can be removed from $\mathbb{S}$. Often the root of the heap $\mathbb{S}$ cannot be removed at all and the next position fix has to be awaited.

However, in rare cases, there may exist several sensed positions with equal or similar values $\varphi(s_i)$ that are removed from $\mathbb{S}$ all together at a certain point in time. By analyzing

$$\kappa(s_i) := \frac{\epsilon - \delta - \left| s_i.\vec{p} - \vec{l}(s_i.t) \right|}{s_i.t - l_{\mathrm{O}}.t} \ ,$$

one can see that this particularly may occur if the object first deviates far from the predicted movement and then linearly returns to it. The reason is that during the linear movement back to predicted one, the numerator and the denominator of $\kappa(s_i)$ both increase linearly.

Since the time for removing the root of a heap is a function in $\mathcal{O}(\log(m))$, the worst case computing time for the removal of positions from $\mathbb{S}$ according

to the optimization of CDR is a function in

$$\begin{aligned}\mathcal{O}(\log(m) + \log(m-1) + \ldots + 1) &= \mathcal{O}(\log(m!)) \\ &= \mathcal{O}(m \log(m)) \ .\end{aligned}$$

Hence, the overall computing time per position fix of $\mathrm{CDR_m}$ is bounded by a function in $\mathcal{O}(m \log(m))$.

2.5 Generic Remote Real-Time Trajectory Simplification

$\mathrm{LDR}_{\frac{1}{2}}$ and CDR use dead reckoning for two different problems, namely the tracking of the current position and the simplification of the past trajectory. While this leads to simple solutions, the efficiency of simplification depends on the quality of dead reckoning, which has been designed for position tracking only.

On the other hand, there exists a variety of efficient line simplification algorithms that could be used for this purpose. Therefore, it is a good idea to separate position tracking from simplification issues as far as possible to gain flexibility.

In this section, we propose the *Generic Remote Real-Time Trajectory Simplification* (GRTS) protocol, which clearly separates tracking of the current position from simplification of the past trajectory and which can be combined with any line simplification algorithm suited for trajectories.

First, we present the basic protocol and algorithm and prove its correctness. Then, we discuss how to bound the space consumption and computing time for line simplification in GRTS and how to incorporate time-dependent maximum sensing deviations $\delta(t)$. Finally, we present two realizations of GRTS, with the optimal line simplification algorithm by Imai and Iri [II88] as well as with an efficient simplification heuristic [MdB04] referred to as section heuristic.

2.5.1 Basic Protocol and Algorithm

Although, it is a good idea to separate position tracking from simplification issues as far as possible, the simplification process must be synchronized with

position tracking to make sure that the simplified data arrives in time at the MOD. The GRTS protocol follows a synchronization pattern, which we call *per-update simplification*.

With this pattern, simplification is performed whenever the position tracking mechanism decides to send an update message. For this purpose, the moving object stores a partial history of sensed positions, which serves as input for the simplification process. Based on this input, the simplification algorithm generates a sequence of vertices for updating the simplified trajectory $\vec{u}(t)$, which is included in the update message. In many cases, the generated sequence replaces one or few vertices of $\vec{u}(t)$ only – without increasing their number. Therefore, GRTS has better reduction efficiency than $\mathrm{LDR}_{\frac{1}{2}}$ and CDR, which always generate one additional vertex per update.

Depending on the line simplification algorithm used with GRTS, the simplification process may be prepared with each sensed position, to reduce the computing time for line simplification when the position tracking mechanism decides to send an update message. If GRTS is realized with an online algorithm, the simplification even can be performed with each sensing operation – resulting in *per-sense simplification* – as explained in Section 2.5.6.

In the following, we consider LDR for position tracking in GRTS as LDR is the most efficient, general applicable position tracking protocol [WSCY99, LR01, CJP05]. However, GRTS can be realized with any position tracking protocol based on a polygonal prediction function $\vec{l}(t)$.

GRTS divides the simplified trajectory $\vec{u}(t)$ into three parts as depicted in Figure 2.7:

1. *Stable part:* This part generally comprises a large number of vertices, stored by the MOD only – except for the last vertex u_{m-k}, which is also known to the moving object, as explained below.

2. *Variable part:* It generally comprises few vertices only, known to the MOD *and* the moving object. The last vertex u_m also composes the prediction origin for the next part.

3. *Predicted part:* This part is given by the prediction function $\vec{l}(t)$, i.e. the origin u_m and the vector $\vec{l}_{\mathrm{V}}$, and is known to the MOD and the moving

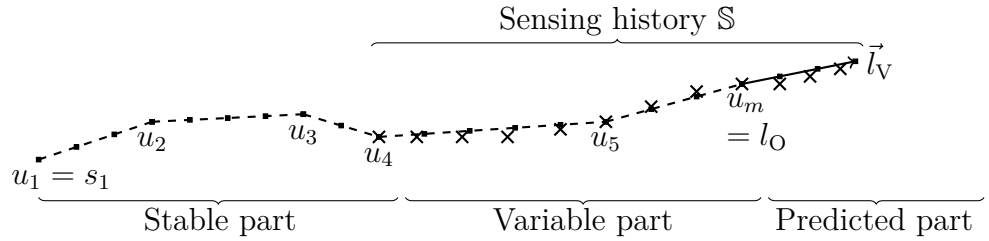


Figure 2.7: Three parts of $\vec{u}(t)$ and corresponding $\mathbb{S}$ for GRTS.

object.

The moving object not only stores the vertices of the variable part and the predicted velocity but also the *sensing history* $\mathbb{S}$ for those two parts. Note that $\mathbb{S}$ further includes the last sensed position of the stable part – i.e. the vertex u_{m-k} – as required for simplifying $\vec{s}(t)$ for $t > u_{m-k}.t$. Formally, $\mathbb{S}$ is the sequence of chronologically ordered sensed positions with $\text{first}(\mathbb{S}) = u_{m-k}$ and $\text{last}(\mathbb{S}) = s_R$.

The MOD is not aware of the differentiation between stable and variable part, but only the moving object. Once the moving object decides that a vertex u_i belongs to the stable part, this vertex will not be changed by future updates. Hence, the stable part grows in an append-only fashion. The variable part, in contrast, may be changed by future updates. An update message therefore consists of three elements:

1. The number of vertices to remove from $\vec{u}(t)$, starting backwards at u_m.
2. The new vertices $\mathbb{U} := (u_j, \ldots, u_m)$ to append to $\vec{u}(t)$, which may increase but (rarely) even decrease the overall number of vertices.
3. The new velocity vector $\vec{l}_V$, which replaces the previous prediction.

Figure 2.8 shows the pseudo code of the basic GRTS algorithm executed by the mobile object. Initially, the moving object transmits its most recent sensed position $s_R = s_1$ as first vertex u_1 to the MOD (line 4). Thus, u_1 also serves as prediction origin until the next update. Then, the object executes the while loop (lines 8 to 24) as long as it wants to report its trajectory to the MOD.

1: $s_{\mathrm{R}} \leftarrow$ sense position ▷ Most recent sensed position.
2: $\mathbb{U} \leftarrow (s_{\mathrm{R}})$ ▷ New vertices for $\vec{u}(t)$.
3: $\vec{l}_{\mathrm{V}} \leftarrow 0$ ▷ Predicted velocity.
4: send update message $(0, \mathbb{U}, \vec{l}_{\mathrm{V}})$ to MOD
5: $\mathbb{S} \leftarrow (s_{\mathrm{R}})$ ▷ Sensing history.
6: $\mathbb{V} \leftarrow ()$ ▷ Vertices of variable part of $\vec{u}(t)$.
7: $\mathbb{U} \leftarrow ()$
8: **while** report movement **do**
9: $s_{\mathrm{R}} \leftarrow$ sense position
10: $\mathbb{S} \leftarrow \mathbb{S} \parallel (s_{\mathrm{R}})$ ▷ Append s_{R} to sensing history.
11: **if** LDR causes update **then**
12: $\mathbb{U} \leftarrow$ line simplification with bound $\epsilon - \delta$ on $\mathbb{S}$
13: $\mathbb{U} \leftarrow \mathbb{U} \setminus (\mathrm{first}(\mathbb{U}))$ ▷ $\mathrm{first}(\mathbb{U}) = \mathrm{first}(\mathbb{S})$ and belongs to stable part.
14: $\vec{l}_{\mathrm{V}} \leftarrow$ compute new predicted velocity ...
15: send update message $(|\mathbb{V} \setminus \mathbb{U}|, \mathbb{U} \setminus \mathbb{V}, \vec{l}_{\mathrm{V}})$ to MOD
16: $\mathbb{V} \leftarrow \mathbb{U}$
17: $\mathbb{U} \leftarrow ()$
18: **if** $\mathbb{V}$ and $\mathbb{S}$ should be reduced **then**
19: $\mathbb{V}' \leftarrow$ some prefix of $\mathbb{V}$ that should belong to stable part ...
20: $\mathbb{S} \leftarrow (s \in \mathbb{S} \mid s.t \geq \mathrm{last}(\mathbb{V}').t)$ ▷ Clear $\mathbb{S}$ up to last stable vertex.
21: $\mathbb{V} \leftarrow \mathbb{V} \setminus \mathbb{V}'$ ▷ Set new variable part of $\vec{u}(t)$.
22: **end if**
23: **end if**
24: **end while**
25: $\mathbb{U} \leftarrow$ line simplification with bound $\epsilon - \delta$ on $\mathbb{S}$
26: $\mathbb{U} \leftarrow \mathbb{U} \setminus (\mathrm{first}(\mathbb{U}))$
27: send update message $(|\mathbb{V} \setminus \mathbb{U}|, \mathbb{U} \setminus \mathbb{V})$ to MOD ▷ Final update, no $\vec{l}_{\mathrm{V}}$.

Figure 2.8: Basic GRTS algorithm.

During each iteration, the object first senses its current position (line 9) and appends it to the sensing history (line 10). Then, it checks whether it has to send an update message to the MOD according to LDR (line 11).

If so, the object computes a simplified trajectory segment for the movement of the variable and predicted part using $\mathbb{S}$. Since the segment of $\vec{s}(t)$ given by $\mathbb{S}$ does not deviate by more than the maximum sensing deviation δ from $\vec{a}(t)$,

it executes the line simplification algorithm with *simplification bound* $\epsilon - \delta$ and stores the resulting vertices of the simplified segment in $\mathbb{U}$ (line 12). Hence, the simplified segment given by $\mathbb{U}$ approximates $\vec{a}(t)$ on $(\text{first}(\mathbb{S}).t, s_{\text{R}}.t]$ according to ϵ. Note that $\text{first}(\mathbb{U})$ can be safely removed from $\mathbb{U}$ since it always is $\text{first}(\mathbb{S})$ and thus the last vertex u_{m-k} of the stable part. Then, the object computes a new velocity vector for LDR (line 14) and creates a corresponding update message.

To minimize the size of the update message, only those vertices of $\mathbb{U}$ that actually change the variable part are included. For this purpose, the algorithm maintains the vertices of the variable part in an array $\mathbb{V}$ (lines 6 and 16). To create the update message, the number of vertices that have to be removed from the end of the variable part (expressed by $|\mathbb{V} \setminus \mathbb{U}|$) and the new vertices that have to be added to it (expressed by $\mathbb{U} \setminus \mathbb{V}$) are computed. This information is sent together with the new predicted velocity to the MOD (line 15).

Subsequently, the moving object stores the vertices $\mathbb{U}$ as new vertices $\mathbb{V}$ of the variable part (line 16) and clears $\mathbb{U}$ (line 17).

Finally, the object may decide to reduce the size of the variable part by removing some prefix of $\mathbb{V}$ and $\mathbb{S}$, respectively (lines 18 to 22). A possible policy is to limit the size of $\mathbb{V}$ to some given parameter k. We refer to this variant as GRTS_{k} in the following.[2] However, GRTS_{k} does not limit the size of the sensing history $\mathbb{S}$, which has important impact on the space consumption and computing time per position fix. Therefore, we propose the variants GRTS_{m} and GRTS_{mc} in Section 2.5.3, which limit $|\mathbb{S}|$ to a given parameter m.

Once the moving object wants to stop reporting its movement, it computes a last simplification of $\mathbb{S}$ (line 25) and sends a corresponding final update message to the MOD (line 27).

The algorithm executed by the MOD is rather simple. On receiving a message $(|\mathbb{V} \setminus \mathbb{U}|, \mathbb{U} \setminus \mathbb{V}, \vec{l}_{\text{V}})$, it removes the $|\mathbb{V} \setminus \mathbb{U}|$ last vertices from $\vec{u}(t)$, appends the vertices $\mathbb{U} \setminus \mathbb{V}$ to $\vec{u}(t)$ and replaces the current predicted velocity with the new vector $\vec{l}_{\text{V}}$.

To determine the object's position at time t', the MOD has to distinguish two cases, similar to CDR:

[2] According to this nomenclature, the GRTS algorithm proposed in [LFDR09] is GRTS_{k} with $k = 1$.

1. $t' \leq u_m.t$: The MOD calculates $\vec{u}(t')$ by linear interpolation between the vertices u_j and u_{j+1} with $u_j.t \leq t' \leq u_{j+1}.t$ as described in Section 2.2.
2. $t' > u_m.t$: The MOD calculates $\vec{u}(t')$ by means of the prediction function $\vec{l}(t)$ given by $l_{\mathrm{O}} = u_m$ and $\vec{l}_{\mathrm{V}}$.

2.5.2 Proof of Correctness

To show the correctness of GRTS, we prove that GRTS satisfies the simplification and the real-time constraint, cf. Section 2.2. Both constraints are to be fulfilled within the time interval $[s_1.t, t_{\mathrm{C}}]$, where t_{C} denotes the current time. Clearly, the simplified trajectory $\vec{u}(t)$ is always defined on $[s_1.t, t_{\mathrm{C}}]$ since the prediction by $\vec{l}_{\mathrm{V}}$ is not limited in time.

Theorem 1 (Correctness of GRTS): At every point in time t_{C}, the simplified trajectory $\vec{u}(t')$ known to the MOD satisfies $|\vec{u}(t) - \vec{a}(t)| \leq \epsilon$ for every $t' \in [s_1.t, t_{\mathrm{C}}]$.
Proof: For any t_{C}, consider the three parts of $\vec{u}(t)$ illustrated in Figure 2.7:

- *Stable part:* It is $s_1.t = u_1.t \leq t' \leq u_{m-k}.t$ for some $k > 0$. Let s_i and s_{i+1} be the sensed positions that enclose t', i.e. $s_i.t \leq t' \leq s_{i+1}.t$.

  According to line simplification (line 12), it holds that $|\overline{s_i\, s_{i+1}}(t') - \vec{u}(t')| \leq \epsilon - \delta$. Using the maximum sensing deviation δ defined in Section 2.2, we conclude the triangle inequality $|\vec{u}(t') - \vec{a}(t')| \leq |\vec{u}(t') - \overline{s_i\, s_{i+1}}(t')| + |\overline{s_i\, s_{i+1}}(t') - \vec{a}(t')| \leq \epsilon - \delta + \delta = \epsilon$.

- *Variable part:* It is $\mathrm{first}(\mathbb{S}).t = u_{m-k}.t < t' \leq u_m.t$. At the time of the most recent execution of the line simplification algorithm the last vertex of the variable part u_m was equal to $\mathrm{last}(\mathbb{S})$.

  Hence, the vertices $(u_{m-k}, \ldots, u_m)$ compose a simplification of the sensed trajectory segment within the time interval $[u_{m-k}.t, u_m.t]$ according to the bound $\epsilon - \delta$. Analogous to the stable part, we conclude $|\vec{u}(t') - \vec{a}(t')| \leq \epsilon$ using triangle inequality.

- *Predicted part:* It is $u_m.t < t' \leq t_{\mathrm{C}}$. As explained above, the LDR protocol guarantees $|\vec{l}(t_{\mathrm{C}}) - \vec{a}(t_{\mathrm{C}})| \leq \epsilon$ for $\vec{l}(t)$ given by $l_{\mathrm{O}} = u_m$ and $\vec{l}_{\mathrm{V}}$ known to the MOD.

Since the MOD has not received a new prediction up to current time t_C, we conclude that $\forall\, t' \in (u_m.t, t_C] : |\vec{l}(t') - \vec{a}(t')| \leq \epsilon$.

□

2.5.3 Space- and Time-bounded Simplification

The basic GRTS algorithm does not define a policy when to reduce the variable part and to what extent. A possible approach is to limit $|\mathbb{V}|$ to some parameter k (e.g. $k = 1$ or 2), as explained in Section 2.5.1. This variant GRTS_k, however, does not limit the size of the sensing history $\mathbb{S}$, which has important impact on the space consumption and computing time per position fix, depending on the line simplification algorithm used with GRTS.

Therefore, we propose two space- and time-bounded variants named GRTS_m and GRTS_mc in the following, limiting $|\mathbb{S}|$ to a given m (e.g. $m = 100$ or 500). Since the latter variant builds on the former, we first explain GRTS_m and then GRTS_mc.

GRTS_m. Figure 2.9 shows the pseudo code of the GRTS_m algorithm executed by the moving object. The most important difference to the basic algorithm is that $\mathbb{U}$ is created incrementally, each time $|\mathbb{S}|$ reaches m. Thus, $\mathbb{S}$ is not only simplified if LDR causes an update, but also if $|\mathbb{S}| = m$.

For this purpose, $|\mathbb{S}|$ is checked after each sensing operation (line 11). If it reaches m, a simplification $\mathbb{U}'$ for $\mathbb{S}$ is computed (line 12), the first vertex is removed as in the basic algorithm (line 13), and the subsequent vertex of $\mathbb{U}'$ is added to $\mathbb{U}$ (line 14). Thereafter, the sensing history is cleared up to this vertex (line 15). The clearing determines that this vertex is considered to belong to the stable part – although $\mathbb{U}$ is not sent to the MOD until LDR causes an update.

If LDR causes an update, a simplification $\mathbb{U}'$ for the current sensing history is computed and appended to $\mathbb{U}$ (line 18 to 20). Then, $\mathbb{U}$ is sent to the MOD as in the basic algorithm (line 22). Next, $\mathbb{U}'$ is stored as new variable part in $\mathbb{V}$ (line 23), consistent with $\mathbb{S}$. Finally, $\mathbb{U}$ is cleared for the next update (line 24).

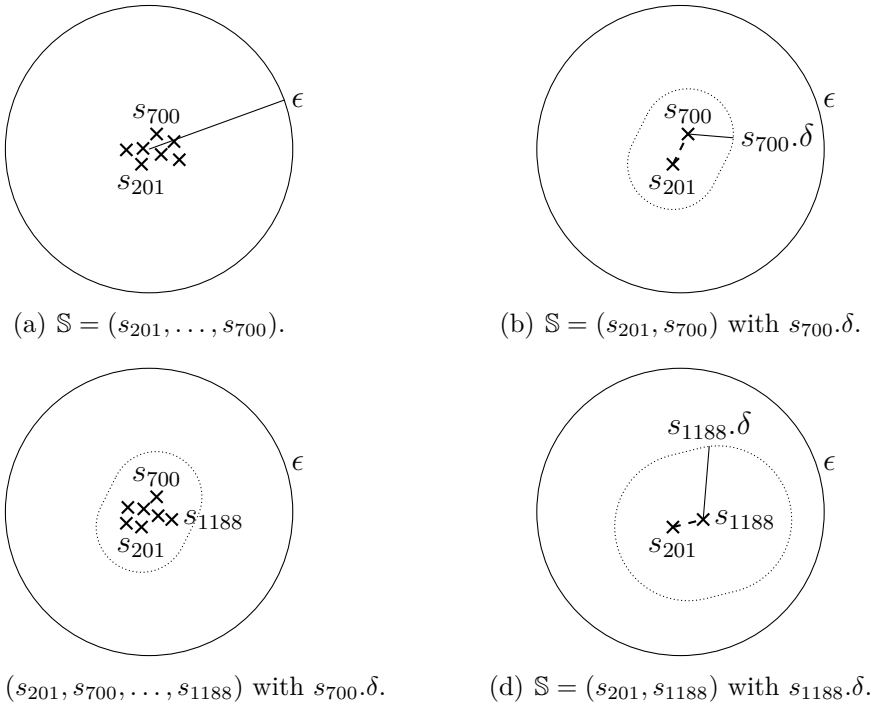
GRTS_m limits the input for line simplification to m sensed positions and thus bounds the space consumption and the computing time per position fix,

1: $s_\mathrm{R} \leftarrow$ sense position ▷ Most recent sensed position.
2: $\mathbb{U} \leftarrow (s_\mathrm{R})$ ▷ New vertices for $\vec{u}(t)$.
3: $\vec{l}_\mathrm{V} \leftarrow 0$ ▷ Predicted velocity.
4: send update message $(0, \mathbb{U}, \vec{l}_\mathrm{V})$ to MOD
5: $\mathbb{S} \leftarrow (s_\mathrm{R})$ ▷ Sensing history.
6: $\mathbb{V} \leftarrow ()$ ▷ Vertices of variable part of $\vec{u}(t)$.
7: $\mathbb{U} \leftarrow ()$
8: **while** report movement **do**
9: $s_\mathrm{R} \leftarrow$ sense position
10: $\mathbb{S} \leftarrow \mathbb{S} \parallel (s_\mathrm{R})$ ▷ Append s_R to sensing history.
11: **if** $|\mathbb{S}| = m$ **then**
12: $\mathbb{U}' \leftarrow$ line simplification with bound $\epsilon - \delta$ on $\mathbb{S}$
13: $\mathbb{U}' \leftarrow \mathbb{U}' \setminus (\mathrm{first}(\mathbb{U}'))$
14: $\mathbb{U} \leftarrow \mathbb{U} \parallel (\mathrm{first}(\mathbb{U}'))$ ▷ Add one vertex for future stable part of $\vec{u}(t)$.
15: $\mathbb{S} \leftarrow (s \in \mathbb{S} \mid s.t \geq \mathrm{last}(\mathbb{U}).t)$ ▷ Clear $\mathbb{S}$ up to this new vertex in $\mathbb{U}$.
16: **end if**
17: **if** LDR causes update **then**
18: $\mathbb{U}' \leftarrow$ line simplification with bound $\epsilon - \delta$ on $\mathbb{S}$
19: $\mathbb{U}' \leftarrow \mathbb{U}' \setminus (\mathrm{first}(\mathbb{U}'))$
20: $\mathbb{U} \leftarrow \mathbb{U} \parallel \mathbb{U}'$
21: $\vec{l}_\mathrm{V} \leftarrow$ compute new predicted velocity ...
22: send update message $(|\mathbb{V} \setminus \mathbb{U}|, \mathbb{U} \setminus \mathbb{V}, \vec{l}_\mathrm{V})$ to MOD
23: $\mathbb{V} \leftarrow \mathbb{U}'$
24: $\mathbb{U} \leftarrow ()$
25: **end if**
26: **end while**
27: [...]
28: send update message $(|\mathbb{V} \setminus \mathbb{U}|, \mathbb{U} \setminus \mathbb{V})$ to MOD ▷ Final update, no $\vec{l}_\mathrm{V}$.

Figure 2.9: GRTS_m algorithm.

depending on the line simplification algorithm being used. For example, with the simplification algorithm by Imai and Iri [II88] and $m = 500$, it bounds the computing time to 21 ms on a 1 GHz processor – compared to unacceptable computing times of up to 2.6 s with GRTS_k, cf. Section 2.7.4.

On the downside, GRTS_m generates an additional vertex for $\vec{u}(t)$ at least



(a) $\mathbb{S} = (s_{201}, \ldots, s_{700})$.

(b) $\mathbb{S} = (s_{201}, s_{700})$ with $s_{700}.\delta$.

(c) $\mathbb{S} = (s_{201}, s_{700}, \ldots, s_{1188})$ with $s_{700}.\delta$.

(d) $\mathbb{S} = (s_{201}, s_{1188})$ with $s_{1188}.\delta$.

Figure 2.10: Example for compression of $\mathbb{S}$ by GRTS_{mc}.

every m sensing operations – even if the object does not move for a long period of time $\gg m{\cdot}T_{\text{S}}$, where T_{S} denotes the sensing period. This drawback, however, can be alleviated effectively by a compression approach for the sensing history, resulting in the variant GRTS_{mc}.

GRTS_{mc}. The fundamental idea of this compression technique is to keep the results of a simplification caused by m in $\mathbb{S}$ – instead of adding the vertex s_b of the first simplified line section $\overline{\text{first}(\mathbb{S})\, s_b}$ to $\mathbb{U}$ – to be able to revise and improve this simplification later on. For this purpose, s_b is extended by a special attribute δ representing the sensed positions between $\text{first}(\mathbb{S})$ and s_b so that these positions can be removed from $\mathbb{S}$. If the simplified vertex s_b is later replaced by another sensed position s_{b+x}, the resulting line section $\overline{\text{first}(\mathbb{S})\, s_{b+x}}$ may span more than m sensed positions.

In detail, the compression technique works as follows: Once $|\mathbb{S}| = m$, GRTS_{mc} computes a simplification of $\mathbb{S}$, just as GRTS_{m}, where $\overline{\text{first}(\mathbb{S})\, s_b}$ de-

notes the first simplified line section. Then, the sensed positions between first($\mathbb{S}$) and s_b are removed from $\mathbb{S}$, but s_b is kept in $\mathbb{S}$ and extended by the attribute δ that gives the maximum deviation between the removed positions and the line section $\overline{\text{first}(\mathbb{S})\, s_b}$, i.e.

$$s_b.\delta := \max_{\text{first}(\mathbb{S}).t < s_i.t < s_b.t} |\overline{\text{first}(\mathbb{S})\, s_b}(s_i.t) - s_i.\vec{p}| \ .$$

Therefore, we refer to such a sensed position s_b as *compressed* position in the following. It may be removed from $\mathbb{S}$ during a subsequent simplification if another line section $\overline{\text{first}(\mathbb{S})\, s_{b+x}}$ that approximates s_b by $\epsilon - (\delta + s_b.\delta)$ is found. The reason is that the property

$$|\overline{\text{first}(\mathbb{S})\, s_{b+x}}(s_b.t) - s_b.\vec{p}| < \epsilon - (\delta + s_b.\delta)$$

guarantees by triangle inequality that

$$\forall\, s_i \text{ with } \text{first}(\mathbb{S}).t < s_i.t < s_b.t \ : \ |\overline{\text{first}(\mathbb{S})\, s_{b+x}}(s_i.t) - s_i.\vec{p}| < \epsilon - \delta \ .$$

An example for this procedure is illustrated in Figure 2.10. In this example, we assume $m = 500$ and consider an object that does not move for long period of time with sensing history $\mathbb{S} = (s_{201}, \ldots, s_{699})$. When adding s_{700} to $\mathbb{S}$, it is $|\mathbb{S}| = m$ (cf. Figure 2.10a). Since the object is not moving, $\mathbb{S}$ can be approximated by the line section

$$\overline{s_{201}\, s_{700}} = \overline{\text{first}(\mathbb{S})\, \text{last}(\mathbb{S})} \ .$$

Therefore, GRTS_{mc} reduces the sensing history to $\mathbb{S} := (s_{201}, s_{700})$ and extends s_{700} by δ with

$$s_{700}.\delta := \max_{201 < i < 700} |\overline{s_{201}\, s_{700}}(s_i.t) - s_i.\vec{p}| \ ,$$

which is ideally zero in case of standstill. $s_{700}.\delta$ is depicted in Figure 2.10b, where the sensed positions are scattered around the center.

After another 488 position fixes, $|\mathbb{S}|$ reaches m again (cf. Figure 2.10c). Assuming that the object also stood still during $[s_{700}.t, s_{1188}.t]$, the sensing history $\mathbb{S} = (s_{201}, s_{700}, s_{701}, \ldots, s_{1188})$ can be approximated by $\overline{s_{201}\, s_{1188}} =$

$\overline{\text{first}(\mathbb{S})\,\text{last}(\mathbb{S})}$ since

$$|\overline{s_{201}\,s_{1188}}(s_{700}.t) - s_{700}.\vec{p}| \approx 0 \leq \epsilon - (\delta + s_{700}.\delta) \approx \epsilon - \delta \ .$$

Hence, the compressed position s_{700} is removed from $\mathbb{S}$ and not added to $\mathbb{U}$. The sensing history is reduced to $\mathbb{S} := (s_{201}, s_{1188})$, where s_{1188} is a compressed position, as shown in Figure 2.10d. For computing $s_{1188}.\delta$ not only

$$\max_{700<i<1188} |\overline{s_{201}\,s_{1188}}(s_i.t) - s_i.\vec{p}|$$

has to be taken into account, but also $|\overline{s_{201}\,s_{1188}}(s_{700}.t) - s_{700}.\vec{p}| + s_{700}.\delta$.

The compression technique works similarly, if an object moves linearly for a long period of time.

Figure 2.11 shows the additional pseudocode for GRTS_{mc} compared to the GRTS_{m} algorithm. The variable c' counts the compressed positions in $\mathbb{S}$ (line 2). The compressed positions are always at the beginning of $\mathbb{S}$, right after the last vertex of the stable part, either known to the MOD or stored in $\mathbb{U}$ for the next update. Note that first($\mathbb{S}$) can be considered as non-compressed position, even if it was compressed during previous iterations, since the movement before first($\mathbb{S}$).t is no more relevant for simplification.

It is very unlikely that two or more compressed positions can be spanned by a line section during future simplifications. To prevent that $\mathbb{S}$ becomes filled with compressed positions, causing frequent but ineffective simplifications, their number should be kept small. Therefore, GRTS_{mc} moves the first compressed position to $\mathbb{U}$ once c' exceeds a certain number c (e.g. $c = 1$ or 2).

If the number of non-compressed sensed positions in $\mathbb{S}$ exceeds $m - c$ (line 6), GRTS_{mc} computes a simplification for $\mathbb{S}$, taking the δ-values of the compressed positions into account (line 7). Then, it searches for the first simplified line section $\overline{s_a\,s_b}$ (i.e. consecutive vertices s_a and s_b in $\mathbb{U}$) where s_b is *not* compressed (line 8). Note that $c' = 0$ implies $s_a = \text{first}(\mathbb{S})$, as in the above example.

GRTS_{mc} compresses s_b by computing $s_b.\delta$ and removes the s_i spanned by $\overline{s_a\,s_b}$ from $\mathbb{S}$ (lines 9 to 18). During this computation, c' is decreased if $\overline{s_a\,s_b}$ spans another compressed position, i.e. if s_a is not the last compressed position in $\mathbb{S}$.

Since s_b is compressed now, c' is increased by one (line 19) and thus may exceed c. If so, the first compressed position (i.e. the second element of $\mathbb{S}$) is added to $\mathbb{U}$ and first($\mathbb{S}$) is removed from $\mathbb{S}$ – such that first($\mathbb{S}$) = last($\mathbb{U}$) as in the GRTS_m algorithm (lines 21 to 23).

2.5.4 Time-dependent Maximum Sensing Deviation

The GRTS_mc algorithm introduced the δ-attribute to represent the deviation along a simplified line section $\overline{s_a\, s_b}$ at the end vertex s_b. This idea can be generalized to represent time-dependent maximum sensing deviations $\delta(t)$, in particular to allow for varying sensor inaccuracies $\sigma(t)$.

For this purpose, *every* sensed position s_i is extended with an attribute δ that gives the maximum deviation between $\vec{a}(t)$ and $\overline{s_{i-1}\, s_i}(t)$, depending on physical movement constraints and inaccuracies of the positioning sensor.

When compressing a sensed position s_b to represent a line section $\overline{s_a\, s_b}$ in GRTS_mc, δ is set to

$$s_b.\delta := \max_{s_a.t < s_i.t \leq s_b.t} |\overline{s_a\, s_b}(s_i.t) - s_i.\vec{p}| + s_i.\delta$$

independent whether s_i is compressed or not. Thus, there is no difference between compressed and non-compressed positions, except that the former are counted by c'.

As there exists no global constant δ anymore, the line simplification algorithm is called with ϵ only, but has to account for the individual δ-values of the $s_i \in \mathbb{S}$.

Next, we discuss how to realize GRTS with two different line simplification algorithms and how to include the individual δ-values in these algorithms.

2.5.5 GRTS with Optimal Line Simplification Algorithm

Here, we describe how to combine GRTS with the optimal simplification algorithm introduced in [II88]. Although this algorithm has originally been designed for offline usage, we apply it online following the per-update simplification pattern. Thus, whenever LDR decides to send a new update or the

```
1: [...]
2: c' ← 0                                   ▷ Counts the compressed positions in 𝕊.
3: while report movement do
4:     s_R ← sense position
5:     𝕊 ← 𝕊 ‖ (s_R)                        ▷ Append s_R to sensing history.
6:     if |𝕊| − c' = m − c then
7:         𝕌' ← line simplification with bound ϵ − δ on 𝕊
8:         search first line section s_a s_b in 𝕌' where s_b is not compressed
9:         s_b.δ ← 0
10:        for all s_i ∈ 𝕊 with s_a.t < s_i.t < s_b.t do
11:            if s_i is compressed then     ▷ s_a s_b spans compressed position.
12:                s_b.δ ← max(s_b.δ, |s_a s_b(s_i.t) − s_i.p⃗| + s_i.δ)
13:                c' = c' − 1              ▷ Because s_i is removed from 𝕊.
14:            else                         ▷ Non-compressed position.
15:                s_b.δ ← max(s_b.δ, |s_a s_b(s_i.t) − s_i.p⃗|)
16:            end if
17:            𝕊 ← 𝕊 \ (s_i)                ▷ Reduce 𝕊.
18:        end for
19:        c' = c' + 1                      ▷ Because s_b is compressed now.
20:        if c' > c then   ▷ To decrease c', move first compressed position to 𝕌.
21:            𝕊 ← 𝕊 \ (first(𝕊))
22:            𝕌 ← 𝕌 ‖ (first(𝕊))
23:            c' = c' − 1
24:        end if
25:    end if
26:    if LDR causes update then
27:        [...]
28:    end if
29: end while
30: [...]
```

Figure 2.11: Compression approach for $\mathbb{S}$ in the GRTS_{mc} algorithm.

sensing history gets too large (in case of GRTS_{m} and GRTS_{mc}), the algorithm is initiated with input $\mathbb{S}$.

In detail, the algorithm first considers the sensed positions in $\mathbb{S}$ as vertices

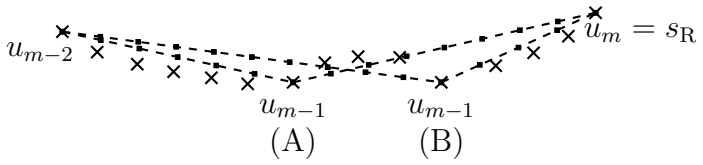


Figure 2.12: Two possible simplifications (A) and (B) with minimal number of vertices $\mathbb{U}' = (u_{m-2}, u_{m-1}, u_m)$.

of an unweighted, directed graph and adds an edge for each pair of sensed positions (s_i, s_{i+x}), where the line section $\overline{s_i\, s_{i+x}}$ approximates every sensed position $s_j \in \mathbb{S}$ with $i < j < i + x$ by ϵ, taking the global δ or the individual $s_i.\delta$ into account. This particularly applies to every pair (s_i, s_{i+1}).

Second, it computes a shortest path between the first vertex first($\mathbb{S}$) and the last vertex last($\mathbb{S}$) = s_R. The vertices $\mathbb{U}'$ of the shortest path compose a simplified trajectory that approximates $\vec{a}(t)$ within the time interval $[\text{first}(\mathbb{S}).t, s_R.t]$ by ϵ.

Due to the segment-wise simplification of the sensed trajectory, induced by the choice of the variable part, the corresponding realizations $\text{GRTS}_\text{k}^\text{Opt}$, $\text{GRTS}_\text{m}^\text{Opt}$, and $\text{GRTS}_\text{mc}^\text{Opt}$ generally do not achieve the optimal, best possible reduction rate as it would be achieved with the optimal line simplification algorithm being applied offline to the overall sequence of sensed positions.

Certainly, the optimal reduction could be achieved by setting k or m to ∞, i.e. by removing the stable part of $\vec{u}(t)$. However, this causes unacceptable computing times and requires very large amounts of space, which can be already seen from the evaluation of GRTS_k with $k = 1$ and 3 (cf. Section 2.7.4). Moreover, it may cause very large update messages.

The underlying reason for the suboptimal reduction is that there may exist several simplifications with minimum number of vertices $\mathbb{U}'$ for a given $\mathbb{S}$. Figure 2.12 illustrates an example of two possible simplifications $\mathbb{U}' = (u_{m-2}, u_{m-1}, u_m)$, implying two possible sequences of vertices to be sent to the MOD or stored for future updates. Generally, choosing the simplification with maximum $u_{m-1}.t$ – here (B) – is a good heuristic since it minimizes the number of sensed positions spanned by the last line section $\overline{u_{m-1}\, u_m}$, which is likely to be revised by future simplifications. Nevertheless, there may be also cases,

where choosing another simplification would yield a better overall reduction efficiency.

Note that the construction of the graph can be performed incrementally, after each sensing operation, despite the per-update simplification pattern. Such an implementation reduces the computing time after those sensing operations that cause a simplification.

2.5.6 GRTS with Section Heuristic

The section heuristic is a simple online line simplification algorithm, which has been proposed in various works including [MdB04][3], [AHPMW05], and [HGNM08].

For simplifying a sequence of sensed positions $(s_1, s_2, \ldots)$ by bound ϵ, the section heuristic works as follows: First, it sets s_1 as vertex u_1 of the simplified trajectory. Then, it iteratively probes the line sections $\overline{s_1\, s_2}, \overline{s_1\, s_3}, \ldots$ until it finds the first section $\overline{s_1\, s_x}$ that would violate ϵ, i.e. where

$$\exists\, s_i \in (s_1, \ldots, s_x) : |\overline{s_1\, s_x}(s_i.t) - s_i.\vec{p}| > \epsilon - \delta$$

or, with individual $s_i.\delta$,

$$\exists\, s_i \in (s_1, \ldots, s_x) : |\overline{s_1\, s_x}(s_i.t) - s_i.\vec{p}| + s_i.\delta > \epsilon\ .$$

In this case, the section heuristic chooses the previous sensed position s_{x-1} as vertex of the simplification. Next, it repeats the above procedure starting at s_{x-1}, and so on.

Since this online algorithm processes the sensed positions iteratively, it allows for *per-sense simplification* by executing the section heuristic for the most recent sensed position s_R after each sensing operation. The advantage of per-sense simplification is that the simplification is performed as early as possible, resulting in a smaller sensing history $\mathbb{S}$ on average. Moreover, the computing

[3] The authors of [MdB04] refer to the section heuristic as Opening-Window algorithm (OPW) and distinguish two variants with different distance metrics. The one with the better reduction efficiency, which corresponds to the section heuristic as explained here, is called BOPW-TR.

```
1: [...]
2: while report movement do
3:     s_R ← sense position
4:     if ∃ s_i ∈ 𝕊 : |first(𝕊) s_R(s_i.t) − s_i.p⃗| + s_i.δ > ε then
5:         𝕌 ← 𝕌 ‖ (last(𝕊))        ▷ Append last sensed position as vertex for u⃗(t).
6:         𝕊 ← (last(𝕊))
7:     end if
8:     𝕊 ← 𝕊 ‖ (s_R)                              ▷ Append s_R to sensing history.
9:     if LDR causes update then
10:        𝕌 ← 𝕌 ‖ (last(𝕊))              ▷ Append s_R as u_m and prediction origin.
11:        l⃗_V ← compute new predicted velocity ...
12:        send update message (|𝕍 \ 𝕌|, 𝕌 \ 𝕍, l⃗_V) to MOD
13:        𝕍 ← (last(𝕌))                          ▷ |𝕍| > 1 would not be exploited.
14:        𝕌 ← ()
15:    end if
16: end while
17: [...]
```

Figure 2.13: $\text{GRTS}_{\text{k}}^{\text{Sec}}$ algorithm with per-sense simplification and fixed $k = 1$.

time for line simplification is distributed over all iterations of GRTS.

This property of the section heuristic, however, is also the reason why it does not exploit variable parts consisting of more than one line section. When realizing GRTS_{k} with the section heuristic the parameter k should be fixed to 1 therefore.

Figure 2.13 shows the corresponding pseudocode of $\text{GRTS}_{\text{k}}^{\text{Sec}}$. For each sensed position s_{R}, the algorithm checks whether the line section $\overline{\text{first}(\mathbb{S})\, s_{\text{R}}}$ approximates the sensed positions in-between by simplification bound ϵ and $\delta(t)$ or not (line 4). If not, it appends the last sensed position – the one before s_{R} – to $\mathbb{U}$ (line 5) and reduces $\mathbb{S}$ accordingly (line 6). When LDR causes a new update, the most recent sensed position is simply appended to $\mathbb{U}$ (line 10) and a corresponding update message is sent to the MOD (line 12). Finally, $\text{GRTS}_{\text{k}}^{\text{Sec}}$ sets $\mathbb{V}$ to (last($\mathbb{U}$)), consistent with $\mathbb{S}$, and clears $\mathbb{U}$ for the next update.

To limit the size of $\mathbb{S}$ to some parameter m, the simplification condition

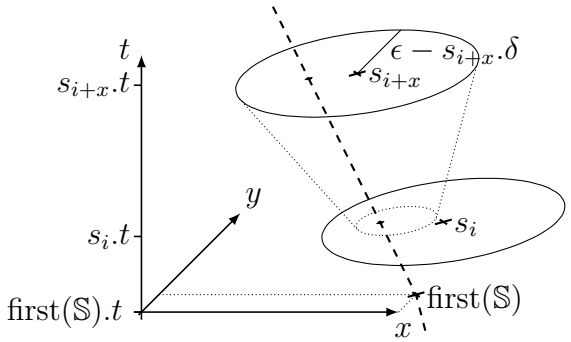


Figure 2.14: Geometric illustration for optimization of the section heuristic.

(line 4) can be extended by

$$\textbf{or } |\mathbb{S}| = m - 1$$

resulting in $\text{GRTS}_{\text{m}}^{\text{Sec}}$. Adding the compression approach described in Section 2.5.3 finally results in $\text{GRTS}_{\text{mc}}^{\text{Sec}}$. Analogous to the parameter k of $\text{GRTS}_{\text{k}}^{\text{Sec}}$, the parameter c of $\text{GRTS}_{\text{mc}}^{\text{Sec}}$ should be fixed to 1 as the section heuristic would never exploit two or more compressed positions for simplification – for the same reason it does not exploit variable parts consisting of more than one line section.

Optimization of $\mathbb{S}$. The average size of $\mathbb{S}$ can be further reduced by a novel optimization of the section heuristic, independent whether $|\mathbb{S}|$ is bounded to some m or not. This optimization particularly supersedes the compression approach of GRTS_{mc}, i.e. $\text{GRTS}_{\text{m}}^{\text{Sec}}$ achieves the same reduction rates as $\text{GRTS}_{\text{mc}}^{\text{Sec}}$ with this optimization (cf. Section 2.7.2).

The basic idea of the optimization is the following: Each sensed position $s_i \in \mathbb{S}$ poses a constraint on the next line section $\overline{\text{first}(\mathbb{S})\, u_m}$ that is going to approximate $\mathbb{S}$. If the constraint given by another sensed position s_{i+x} completely encloses the one given by s_i, then s_i can be removed from $\mathbb{S}$ without affecting the simplification. This reduces the space consumption of the section heuristic by two-thirds on average, cf. Section 2.7.

The constraint defined by a $s_i \in \mathbb{S}$ is that the distance $|\overline{\text{first}(\mathbb{S})\, u_m}(s_i.t) - s_i.\vec{p}| + s_i.\delta$ must not exceed ϵ. This constraint is checked for every potential

line section $\overline{\text{first}(\mathbb{S})\, s_\text{R}}$ (line 4). Geometrically, for each s_i, the line section has to pass the circle with center $s_i.\vec{p}$ and radius $\epsilon - s_i.\delta$ at time $s_i.t$ as illustrated in Figure 2.14. Since the line section's first vertex is known, the circles of two sensed positions s_i and s_{i+x} can be normalized regarding time and compared with each other: The circle of s_{i+x} poses the same constraint like the circle with center $\overline{\text{first}(\mathbb{S})\, s_{i+x}}(s_i.t)$ and radius

$$(\epsilon - s_{i+x}.\delta)\ \frac{s_i.t - \text{first}(\mathbb{S}).t}{s_{i+x}.t - \text{first}(\mathbb{S}).t}$$

at time $s_i.t$. Now, if this circle is contained by the circle of s_i as pictured in Figure 2.14, then s_i can be removed from $\mathbb{S}$. Thus, for each sensed position s_R, the realizations $\text{GRTS}_\text{k}^\text{Sec}$, $\text{GRTS}_\text{m}^\text{Sec}$, and $\text{GRTS}_\text{mc}^\text{Sec}$ can remove every s_i from $\mathbb{S}$ whose circle contains the normalized circle of s_R at $s_i.t$, except $s_i = \text{first}(\mathbb{S})$. In Figure 2.13 this removal should be included between the lines 7 and 8.

2.6 Acceleration-based Movement Constraints

The movement between two sensing operations is bounded by physical constraints such as the maximum speed or acceleration, as explained in Section 2.2. The corresponding values are factored into the maximum sensing deviation δ for trajectory simplification and into the update condition of LDR. It holds: the smaller δ, the higher the possible reduction rate, since δ is to be subtracted from ϵ.

In the previous sections, we exemplarily considered a given maximum speed v_max for simplicity and readability. Yet, for fast objects such as cars or airliners, v_max gives very large values of δ and thus causes bad reduction rates. The reason is that v_max only provides a coarse estimate of the actual movement constraints of such objects. Consider, for example, a sensing period of $T_\text{S} = 1\,\text{s}$ and a car with $v_\text{max} = 50\,\text{m/s}$. According to the resulting speed-based movement constraint, the car is assumed to be able to travel 25 m away and then return to the starting point within one second. This is obviously unrealistic since it requires the car to accelerate (and decelerate) with at least $200\,\text{m/s}^2$.

Next, we therefore explain how to take an object's maximum acceleration

$a_{\max}$ into account. First, we discuss how to compute the maximum sensing deviation δ accordingly. Then, we consider the update condition of LDR.

Given an object with maximum speed $v_{\max}$ and two sensed positions s_{i-1} and s_i, we concluded in Section 2.2 that the object cannot deviate by more than $v_{\max} \cdot T_S/2$ from the line section $\overline{s_{i-1}\, s_i}$.[4] For $a_{\max}$ we analogously conclude that the object cannot deviate by more than

$$\frac{1}{2} a_{\max} \left(\frac{T_S}{2} \right)^2$$

from $\overline{s_{i-1}\, s_i}$ using linear kinematics. Together with the sensor inaccuracy σ, it follows

$$\delta := \sigma + \frac{1}{8} a_{\max} T_S^2 \ .$$

Table 2.1 shows the corresponding values of δ for three typical scenarios of $v_{\max}$ and $a_{\max}$, assuming $\sigma = 7.8\,\text{m}$ and $T_S = 1\,\text{s}$. The scenarios support the argumentation that the ratio between the speed-based value of δ and acceleration-based value increases with the typical speed of the objects.

Incorporating $a_{\max}$ into the update condition of LDR is more complex. For $v_{\max}$ we showed in Section 2.3 that the object has to send an update if

$$|s_R.\vec{p} - \vec{l}(s_R.t + T_S + T_U)| + \sigma + v_{\max}(T_S + T_U) > \epsilon$$

since $s_R.\vec{p}$ the object may move by up to $v_{\max}(T_S + T_U)$ until an update sent after the subsequent sensing operation would have been received by the MOD.

For acceleration-based movement constraints, however, the current velocity has to be taken into account, to be able to estimate the possible deviation between the object's movement and $\vec{l}(t)$ at $s_R.t + T_S + T_U$. Thus, we require an approximation for the *most recent velocity* $\vec{v}_R$ at $s_R.t$. An obvious solution is to use the average velocity between $s_R.t$ and the second last sensed position

[4] In fact, the maximum possible deviation from $\overline{s_{i-1}\, s_i}$ depends on the object's current speed – and thus the distance between $s_{i-1}.\vec{p}$ and $s_i.\vec{p}$ – since the current speed limits the speed for other velocity components to deviate from $\overline{s_{i-1}\, s_i}(t)$. Therefore, δ may depend on time, even if the maximum sensor inaccuracy σ is fixed. This is discussed in detail in [PJ99]. Yet, for significant improvements regarding δ, the object's speed has to be close to $v_{\max}$. Therefore, we neglect this optimization here and focus on $a_{\max}$.

	Human	Car	Airliner
$v_{\max}$	12.0 m/s	50.0 m/s	270.0 m/s
$a_{\max}$	5.0 m/s^2	10.0 m/s^2	20.0 m/s^2
speed-based δ	13.8 m	32.8 m	142.8 m
acceleration-based δ	8.4 m	9.1 m	10.3 m

Table 2.1: Values of δ for typical speed- and acceleration-based movement constraints.

$s_{\mathrm{R}'}$, i.e.

$$\vec{v}_{\overline{\mathrm{R}'\,\mathrm{R}}} := \frac{s_{\mathrm{R}}.\vec{p} - s_{\mathrm{R}'}.\vec{p}}{s_{\mathrm{R}}.t - s_{\mathrm{R}'}.t} = \frac{s_{\mathrm{R}}.\vec{p} - s_{\mathrm{R}'}.\vec{p}}{T_{\mathrm{S}}} \ .$$

The velocity $\vec{v}_{\overline{\mathrm{R}'\,\mathrm{R}}}$ is subject to two approximation errors: First, the object may accelerate (or decelerate) during $[s_{\mathrm{R}'}.t, s_{\mathrm{R}}.t]$ – also sidewards – causing an error of up to $a_{\max} \cdot T_{\mathrm{S}}/2$. Second, $\vec{v}_{\overline{\mathrm{R}'\,\mathrm{R}}}$ is subject to sensor inaccuracies, causing an error of up to $2\sigma/T_{\mathrm{S}}$.

We can distinguish between systematic, time-correlated inaccuracies σ_{sys} and random, noise-like inaccuracies σ_{noise}. For example with GPS, the former are caused by inaccurate ephemeris data and atmospheric effects amongst others, while the latter are caused by the receiver hardware and make only about 10% of the overall sensor inaccuracy σ [Ran94, ME01, Zog09]. Since $\vec{v}_{\overline{\mathrm{R}'\,\mathrm{R}}}$ is computed by two consecutive position fixes, it can be considered to be subject to σ_{noise} only. In sum, $\vec{v}_{\overline{\mathrm{R}'\,\mathrm{R}}}$ may deviate from the actual velocity $\vec{v}_{\mathrm{R}}$ by up to

$$a_{\max}\frac{T_{\mathrm{S}}}{2} + \frac{2\sigma_{\mathrm{noise}}}{T_{\mathrm{S}}} \ .$$

Taking into account the inaccuracy of $s_{\mathrm{R}}.\vec{p}$ and the possible movement and acceleration during the next sensing period T_{S} and the update time T_{U} for processing and transmitting an update message, LDR has to cause an update if

$$\begin{aligned}|s_{\mathrm{R}}.\vec{p} + \frac{s_{\mathrm{R}}.\vec{p} - s_{\mathrm{R}'}.\vec{p}}{T_{\mathrm{S}}}(T_{\mathrm{S}} + T_{\mathrm{U}}) - \vec{l}(s_{\mathrm{R}}.t + T_{\mathrm{S}} + T_{\mathrm{U}})| \\ + \sigma + \frac{1}{2}a_{\max}(T_{\mathrm{S}} + T_{\mathrm{U}})^2 + \left(a_{\max}\frac{T_{\mathrm{S}}}{2} + \frac{2\sigma_{\mathrm{noise}}}{T_{\mathrm{S}}}\right)(T_{\mathrm{S}} + T_{\mathrm{U}}) > \epsilon.\end{aligned}$$

	Human	Car	Airliner
$v_{\max}$	12.0 m/s	50.0 m/s	270.0 m/s
$a_{\max}$	5.0 m/s^2	10.0 m/s^2	20.0 m/s^2
speed-based offset	22.2 m	67.8 m	331.8 m
acceleration-based offset	16.3 m	22.9 m	36.1 m

Table 2.2: Offsets to ϵ for LDR for typical speed- and acceleration-based movement constraints.

The different inaccuracies and the possible movement and acceleration during $T_S + T_U$ can be considered as offset to ϵ similar to δ. Table 2.2 shows the corresponding values for three typical scenarios of $v_{\max}$ and $a_{\max}$, assuming $\sigma = 7.8\,\text{m}$ and $T_S = 1\,\text{s}$. Note that the offsets are significantly greater than the corresponding δ values in Table 2.1 since the latter refer to the movement between two given positions s_{i-1} and s_i. This property is another reason (in addition to the separation of simplification from position tracking) for the significant difference between the number of updates caused by LDR and the number of vertices generated by GRTS or offline simplification – in particular for small values of ϵ.

For instance, for $\epsilon = 25\,\text{m}$, LDR causes more than 1000 updates per hour, while $\text{GRTS}_{\text{m}}^{\text{Sec}}$ with $m = 500$ generates only about 65 vertices. A possible countermeasure is to relax the real-time constraint of trajectory tracking by introducing some temporal tolerance, which has to be subtracted from $T_S + T_U$ in the above formulas. Note that such a temporal tolerance can be only introduced if position tracking is clearly separated from simplification as with GRTS.

If $v_{\max}$ and $a_{\max}$ are both given, the two kinds of movement constraints can be considered simultaneously, simply by choosing the smaller δ value and offset for the update condition of LDR.

2.7 Evaluation

For significant results on the performance of CDR and GRTS, we simulated the different variants and realizations with hundreds of real GPS traces and compared them to $\text{LDR}_{\frac{1}{2}}$ as well as to offline simplification. Besides, for practical

experiences, we also conducted experiments with a prototypical implementation of GRTS, reported in Section 2.8.

In the following, we first describe the simulation setup and then give the results on reduction efficiency, communication, and computational costs. Based on these results, we then draw conclusions for the selection of a concrete tracking approach for a given application scenario.

2.7.1 Setup

We implemented a simulation software for CDR, CDR_m, $\text{GRTS}_\text{k}^\text{Opt}$, $\text{GRTS}_\text{m}^\text{Opt}$, $\text{GRTS}_\text{mc}^\text{Opt}$, $\text{GRTS}_\text{k}^\text{Sec}$, $\text{GRTS}_\text{m}^\text{Sec}$, $\text{GRTS}_\text{mc}^\text{Sec}$, and the existing trajectory tracking approach $\text{LDR}_{\frac{1}{2}}$ [TCS+06] as well as the optimal line simplification algorithm (Ref^Opt) by Imai and Iri [II88] and the Douglas-Peucker algorithm (Ref^DP) [DP73] in the C programming language. We selected Ref^Opt as a reference for comparing our results to the best possible reduction rate, while Ref^DP is a commonly used offline heuristic.

For simulating these algorithms with realistic data, we downloaded several thousand GPS traces (i.e. trajectories sensed by customary GPS receivers) from the OpenStreetMap project [OSM]. In several processing steps, we filtered those traces that provide an individual position fix for each second – and thus have not undergone any previous data reduction – and that could be classified clearly according to their means of transportation into foot, bicycle, and motor vehicle. For classifying a trajectory, we not only relied upon its speed characteristics but also on representative tags specified on the OpenStreetMap website.

Then, we simulated the execution of the trajectory tracking approaches by sequentially feeding the algorithms with the recorded positions given in the GPS traces. For each variant and realization, we measured the number of vertices of the resulting simplified trajectories, the numbers of update messages, and the amounts of transmitted data, depending on ϵ varied from 25 to 100 m. Further, we measured the space consumption and computing time per position fix.

We used a sensing period of $T_\text{S} = 1\,\text{s}$ and a sensor inaccuracy of $\sigma = 7.8\,\text{m}$ with $\sigma_\text{noise} = 0.1\sigma$ in accordance with the GPS traces and the inaccuracies

reported in [GPS-Perf, Zog09]. We further assumed an update time of $T_U = 0.2\,s$. If not stated otherwise, we considered the acceleration-based movement constraint by $a_{max} = 10\,m/s^2$, which gives a maximum sensing deviation of $\delta = 9.1\,m$ and an offset of 22.9 m in the update condition of LDR.

In addition, we applied the offline algorithms Ref^{Opt} and Ref^{DP} with bound $\epsilon - \delta$ to each GPS trace and measured the number of vertices of the resulting simplified trajectories.

All experiments were performed on Intel Xeon Linux Servers with 3.0 GHz using 2 GB RAM.

The different speeds of the means of transportation do not yield any significant differences when comparing the different approaches with each other, but only when considering the absolute values for reduction efficiency and communication. Therefore, we give the average results of the 3×100 largest GPS traces of the three means of transportation and refer to the individual means of transportation and speeds where necessary. Each of the 300 trajectories comprises 1400 to 16500 GPS positions, i.e. spans about 20 min to 5 h.

2.7.2 Reduction Efficiency

The reduction efficiency of trajectory simplification is measured by the *reduction rate* defined as the number of sensed positions divided by the number of vertices of the simplified trajectory $\vec{u}(t)$, i.e. $|(s_1, \ldots, s_n)|/|(u_1, \ldots, u_m)|$.

Figure 2.15 shows the reduction rates of the major tracking approaches and the two reference offline algorithms Ref^{Opt} and Ref^{DP}. As discussed below, the reduction rate of CDR_m with $m = 500$ is equal to the reduction rate of CDR. Similarly, the reduction rate of $\text{GRTS}_m^{\text{Sec}}$ with $m = 500$ is equal to the reduction rate of $\text{GRTS}_{mc}^{\text{Sec}}$ and $\text{GRTS}_k^{\text{Sec}}$, where the latter can be considered as $\text{GRTS}_m^{\text{Sec}}$ with $m = \infty$. For GRTS_{mc} we used $c = 1$ if not stated otherwise. Below, we show that $c > 1$ does not give any improvement.

The reduction rate of CDR (or CDR_m with $m = 500$) is at least twice the reduction rate of $\text{LDR}_{\frac{1}{2}}$, consistent with the analysis in Section 2.3. For $\epsilon < 45\,m$, $\text{LDR}_{\frac{1}{2}}$ does not perform any reduction since its internal accuracy bound $\epsilon' := \frac{1}{2}\epsilon$ becomes smaller than the offset given by the sensor inaccuracy and the movement constraint in the update condition of LDR. Thus, LDR

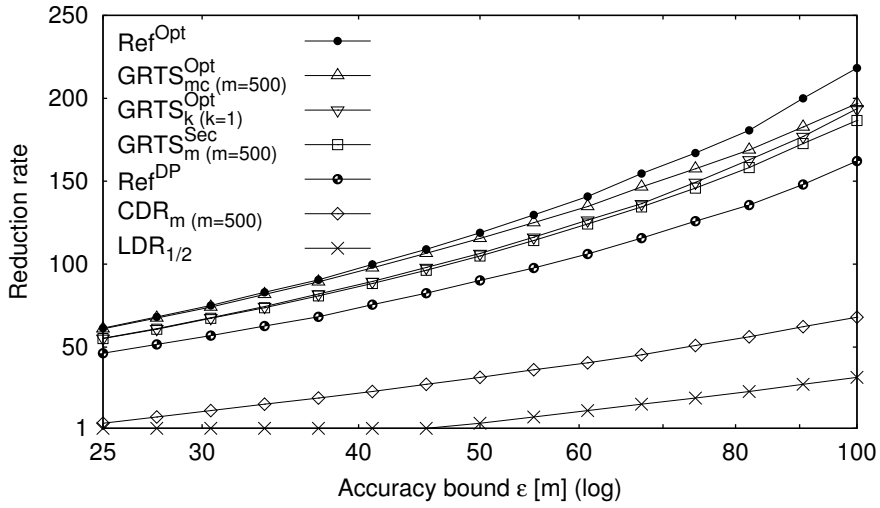


Figure 2.15: Reduction rates of major real-time trajectory tracking and offline simplification approaches.

causes an update after each sensing operation.[5]

All GRTS realizations given in Figure 2.15 outperform the CDR variants by at least factor 2.7 and $\mathrm{LDR}_{\frac{1}{2}}$ by about factor 5.5. This confirms the importance of separating tracking of the current position from simplification of the past trajectory. For example, for $\epsilon = 50$, $\mathrm{GRTS}_{\mathrm{m}}^{\mathrm{Sec}}$ generates 34.3 vertices for $\vec{u}(t)$ on average, whereas CDR (or $\mathrm{CDR}_{\mathrm{m}}$ with $m = 500$) generates about 113.6 vertices.

The reduction rates of the GRTS realizations are always less than 15% below the best possible reduction by the optimal algorithm $\mathrm{Ref}^{\mathrm{Opt}}$. Moreover, the GRTS realizations always outperform $\mathrm{Ref}^{\mathrm{DP}}$. In detail, the reduction rate of $\mathrm{GRTS}_{\mathrm{m}}^{\mathrm{Sec}}$ is 15 to 19% greater than the reduction rate of $\mathrm{Ref}^{\mathrm{DP}}$, whereas $\mathrm{GRTS}_{\mathrm{mc}}^{\mathrm{Opt}}$ even achieves up to 32%. This is a surprising result given the fact that $\mathrm{Ref}^{\mathrm{DP}}$ is performed offline on the entire GPS traces.

The reduction rate of $\mathrm{GRTS}_{\mathrm{m}}^{\mathrm{Sec}}$ is always 1 to 4% below the reduction rate of $\mathrm{GRTS}_{\mathrm{k}}^{\mathrm{Opt}}$ with $k = 1$. The reason is that the variable part of $\vec{u}(t)$ comprises

[5] As mentioned in Section 2.3, many works do not clearly state whether they account for the sensing period, the update time, and the sensor inaccuracy in the update condition of LDR, or not. In the latter case, the factor between the reduction rates of CDR (or $\mathrm{CDR}_{\mathrm{m}}$ with $m = 500$) and $\mathrm{LDR}_{\frac{1}{2}}$ may decrease to about 1.5.

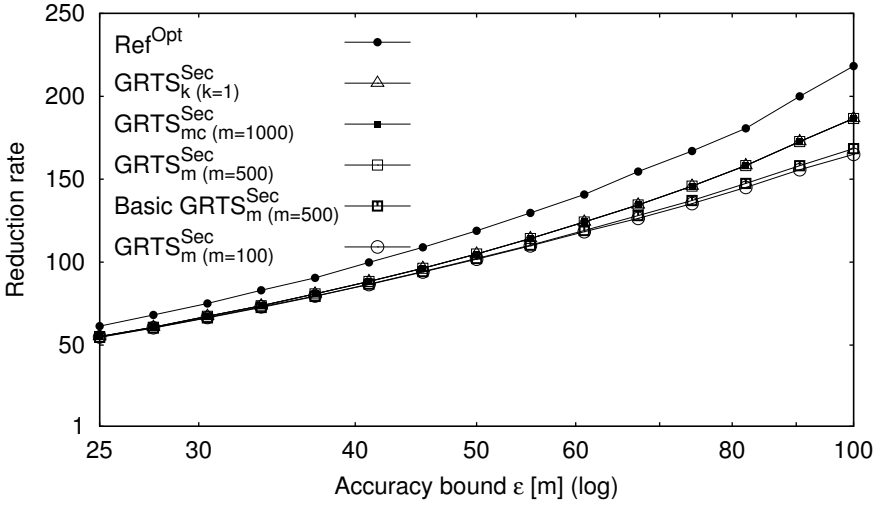


Figure 2.16: Reduction rates of $\text{GRTS}_{\text{k}}^{\text{Sec}}$, $\text{GRTS}_{\text{m}}^{\text{Sec}}$, and $\text{GRTS}_{\text{mc}}^{\text{Sec}}$.

only one vertex in all realizations with the section heuristic – independent of m or k.

Figure 2.16 details the reduction performance of these realizations, where *Basic* $\text{GRTS}_{\text{m}}^{\text{Sec}}$ refers to the realization without the optimization of the section heuristic proposed in Section 2.5.6. It shows that $\text{GRTS}_{\text{m}}^{\text{Sec}}$ with $m = 500$ suffices to achieve the best reduction rate that is possible with the section heuristic. Neither the use of GRTS_{mc}, nor a parameterization of $m > 500$ gives better results. Note again that $\text{GRTS}_{\text{k}}^{\text{Sec}}$ can be considered as $\text{GRTS}_{\text{m}}^{\text{Sec}}$ with $m = \infty$.

The figure further reveals the importance of optimization of the section heuristic in the case where the sensing history $\mathbb{S}$ is bounded by some m. For example, for $m = 500$, it increases the reduction performance by up to 10%. This optimization is also the reason why the compression approach by GRTS_{mc} has no effect on the reduction rate of $\text{GRTS}_{\text{mc}}^{\text{Sec}}$ compared to $\text{GRTS}_{\text{m}}^{\text{Sec}}$.

Analogously, Figure 2.17 shows the reduction performance of the GRTS realizations with the optimal simplification algorithm by Imai and Iri [II88]. For readability, it shows the relative reduction rate compared to the simplification algorithm being applied offline to the entire GPS traces, i.e. Ref^{Opt}.

Clearly, the relative reduction rate is always ≤ 1. The rate would be equal

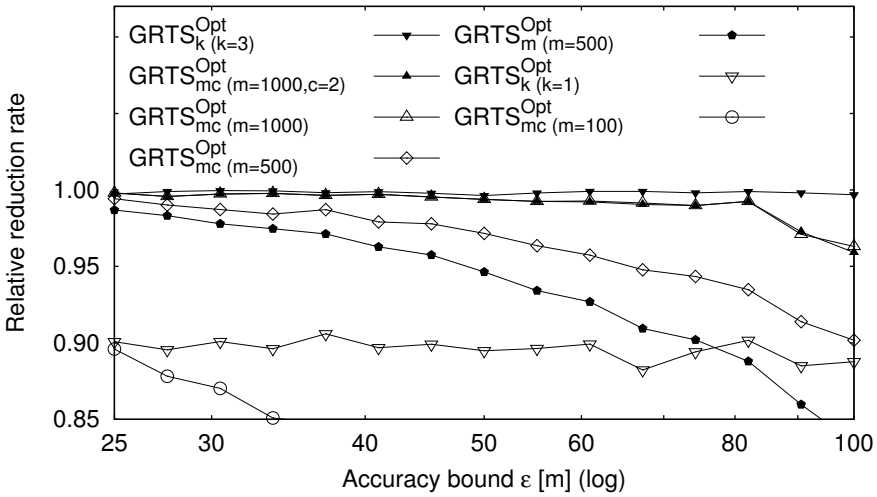


Figure 2.17: Reduction by $\mathrm{GRTS}^{\mathrm{Opt}}_{\mathrm{k}}$, $\mathrm{GRTS}^{\mathrm{Opt}}_{\mathrm{m}}$, $\mathrm{GRTS}^{\mathrm{Opt}}_{\mathrm{mc}}$ relative to $\mathrm{Ref}^{\mathrm{Opt}}$.

to one for $\mathrm{GRTS}^{\mathrm{Opt}}_{\mathrm{k}}$ with $k = \infty$ since there would be no stable part of $\vec{u}(t)$. Yet, the simulation results show that $k = 3$ almost suffices, as the average reduction rate is only 0.2% smaller than the best possible reduction rate. For $k = 1$, the difference is already about 10%.

However, as explained below, the computational costs of $\mathrm{GRTS}^{\mathrm{Opt}}_{\mathrm{k}}$ are too high for practical use due to the unbounded size of $\mathbb{S}$. Hence, $\mathrm{GRTS}^{\mathrm{Opt}}_{\mathrm{m}}$ or $\mathrm{GRTS}^{\mathrm{Opt}}_{\mathrm{mc}}$ have to be used, where the $\mathrm{GRTS}_{\mathrm{mc}}$ realization effectively outperforms the $\mathrm{GRTS}_{\mathrm{m}}$ realization.

Interestingly, $\mathrm{GRTS}^{\mathrm{Opt}}_{\mathrm{mc}}$ with $m = 1000$ and $c = 1$ may also achieve more than 99% of the best possible reduction rate. Enabling more than one compressed position by choosing $c > 1$ does not give any improvement, which is consistent with the discussion in Section 2.5.3. Even with a small value of $m = 500$, the relative reduction rate of $\mathrm{GRTS}^{\mathrm{Opt}}_{\mathrm{mc}}$ is less than 3% below the best possible rate – at least for $\epsilon < 50\,\mathrm{m}$.

For larger values of ϵ, the relative reduction rates of $\mathrm{GRTS}^{\mathrm{Opt}}_{\mathrm{m}}$ and $\mathrm{GRTS}^{\mathrm{Opt}}_{\mathrm{mc}}$ noticeably decrease. The reason is that the average number of sensed positions spanned by a line section of the simplified trajectory $\vec{u}(t)$ increases with ϵ, whereas $|\mathbb{S}|$ is bounded by m. For large ϵ this problem may be alleviated by increasing the sensing period T_{S}, despite the resultant increase of δ. This is

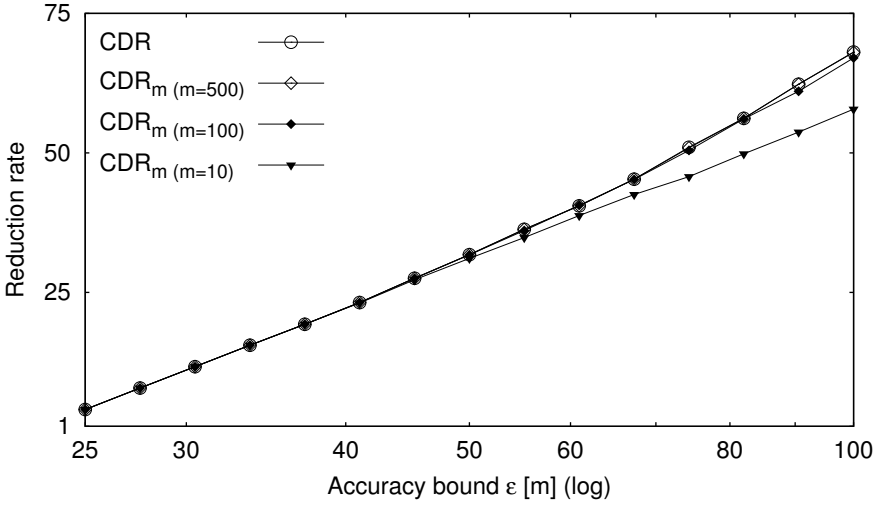


Figure 2.18: Reduction rates of CDR and $\mathrm{CDR_m}$ for different m.

also the reason why we do not give any results for $\epsilon > 100\,\mathrm{m}$.

For the sake of completeness, Figure 2.18 gives the reduction rates of CDR and $\mathrm{CDR_m}$ for different values of m. As mentioned above, the reduction performance of CDR is equal to the performance of $\mathrm{CDR_m}$ with $m = 500$. However, even with only $m = 10$, $\mathrm{CDR_m}$ achieves a remarkable performance compared to CDR.

All the relative reduction rates similarly apply to the individual means of transportation. For example, for $\epsilon = 50\,\mathrm{m}$, the reduction rate of $\mathrm{GRTS_{mc}^{Opt}}$ with $m = 500$ always is at least 95% of the best possible rate – by foot, bicycle, and motor vehicle.

The absolute reduction rates, however, depend on the mean of transportation due to the different ratio between the typical speed and ϵ. For instance, for $\epsilon = 50\,\mathrm{m}$, the reduction rate of $\mathrm{GRTS_{mc}^{Opt}}$ ($m = 500$) is 208.1 for pedestrians, 89.0 for bicycles, and 49.5 for motor vehicles.

Figure 2.19 renders these differences more precisely by showing the reduction rate depending on the speed. For this purpose, we grouped the GPS traces by their average speed and then computed the average reduction rate for each group and approach and parameterization for $\epsilon = 50\,\mathrm{m}$.

The reduction rates are comparatively high for slow objects, as to be ex-

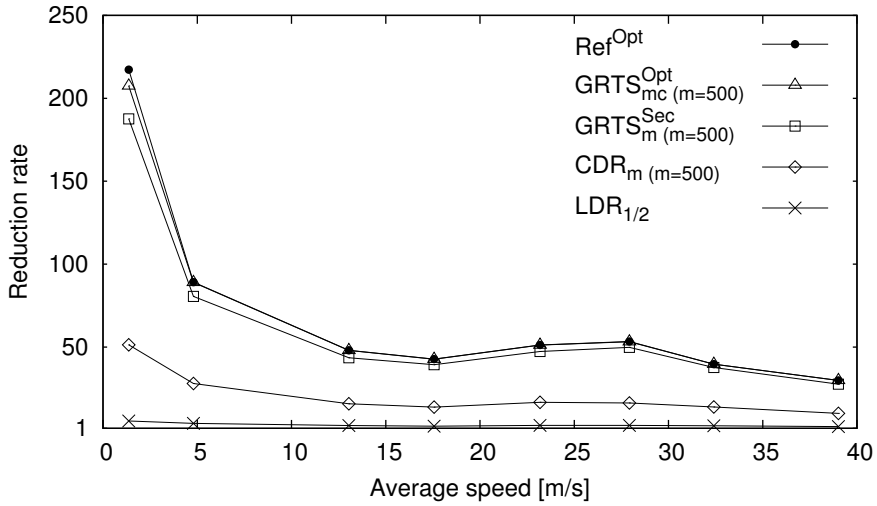


Figure 2.19: Reduction rates depending on average speed for $\epsilon = 50\,\mathrm{m}$.

pected. In addition, the rates largely decrease with increasing speed. Exceptions for an average speed of more than 15 m/s result from the fact that average speed correlates with the kind of road (streets, rural roads, highways), implying different movement characteristics.

Figure 2.20 shows the reduction rates of $\mathrm{CDR_m}$ and $\mathrm{GRTS_{mc}^{Opt}}$ for the acceleration-based movement constraint given by $a_{\max} = 10\,\mathrm{m/s^2}$ and the speed-based movement constraint by $v_{\max} = 20\,\mathrm{m/s}$. The figure only gives the results from GPS traces recorded by foot or bicycle, as $v_{\max} = 20\,\mathrm{m/s}$ does not apply to cars. With GRTS, the use of $a_{\max}$ increases the reduction rate by 7 to 76% compared to $v_{\max}$, depending on ϵ. The larger ϵ, the smaller is the increase since the difference between the corresponding maximum sensing deviations δ vanishes in comparison to ϵ.

Similar applies to the CDR variant. With $v_{\max} = 20\,\mathrm{m/s}$, CDR does not perform any simplification for $\epsilon \leq 30\,\mathrm{m}$ since the simplification is based on LDR and the offset in the update condition exceeds ϵ, causing an update after each sensing operation.

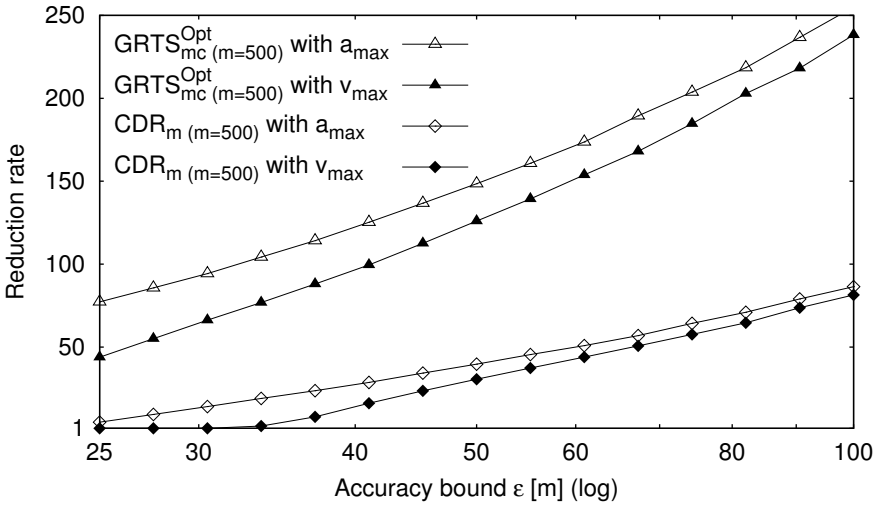


Figure 2.20: Reduction rates of $\mathrm{CDR_m}$ and $\mathrm{GRTS_{mc}^{Opt}}$ for $v_{\max} = 20\,\mathrm{m/s}$ and $a_{\max} = 10\,\mathrm{m/s^2}$. (Only for GPS traces recorded by foot or bicycle.)

2.7.3 Communication Costs

Figure 2.21 shows the number of update messages generated by $\mathrm{LDR}_{\frac{1}{2}}$, $\mathrm{CDR_m}$ ($m = 500$), and GRTS per hour depending on ϵ. It allows verifying that the number of update messages by GRTS is independent on the variant and parameterization of k or m as it only depends on LDR.

The figure also shows that the number of updates caused by the additional section condition of CDR and the limitation of $|\mathbb{S}|$ is negligible (less than 1%) compared to the number of updates caused by LDR, consistent with the analysis of LDR in Section 2.3.

For $\epsilon \leq 45\,\mathrm{m}$, $\mathrm{LDR}_{\frac{1}{2}}$ sends an update after each sensing operation since the internal accuracy bound $\epsilon' := \frac{1}{2}\epsilon$ is smaller than the offset in the update condition given by $a_{\max}$.

This offset is also the reason why GRTS sends more than 1000 updates per hour for $\epsilon = 25\,\mathrm{m}$. This number could by reduced by relaxing the real-time constraint of trajectory tracking as discussed at the end of Section 2.6.

This problem is intensified with the speed-based movement constraint given by $v_{\max} = 20\,\mathrm{m/s}$, as depicted in Figure 2.22, in line with the analytical com-

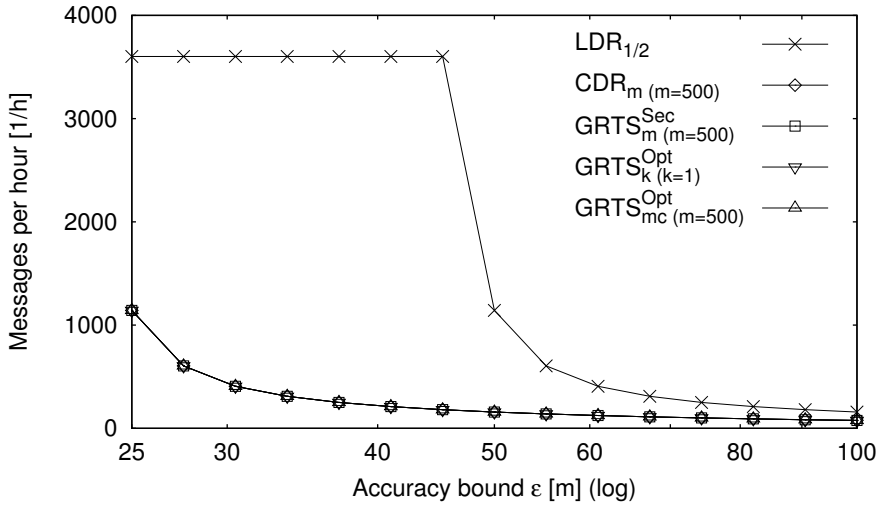


Figure 2.21: Update messages sent by major tracking approaches.

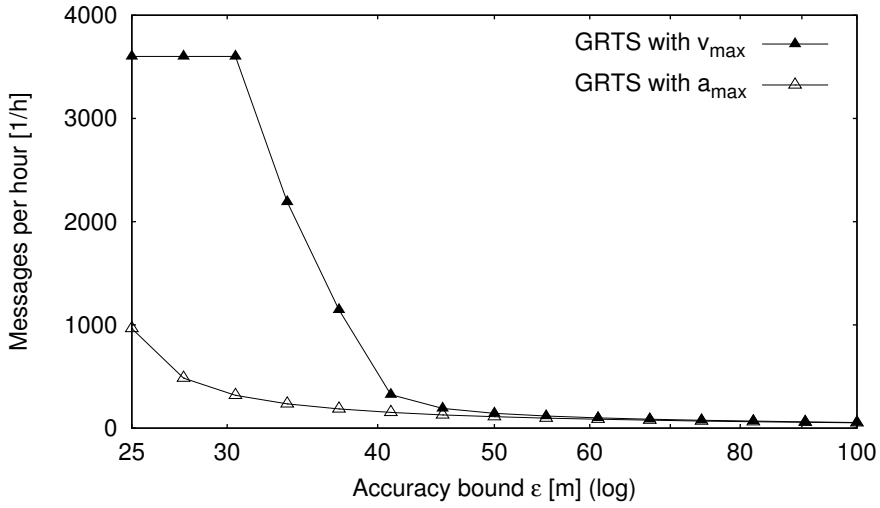


Figure 2.22: Update messages sent by GRTS for $v_{\max} = 20\,\mathrm{m/s}$ and $a_{\max} = 10\,\mathrm{m/s^2}$. (Only for GPS traces recorded by foot or bicycle.)

parison of speed-based and accuracy-based constraints in Section 2.6.

The update messages of $\mathrm{LDR}_{\frac{1}{2}}$ and CDR contain only a prediction, where the origin gives a vertex of $\vec{u}(t)$. GRTS, in contrast, additionally inserts the number of vertices to remove from the variable part of $\vec{u}(t)$ and the vertices

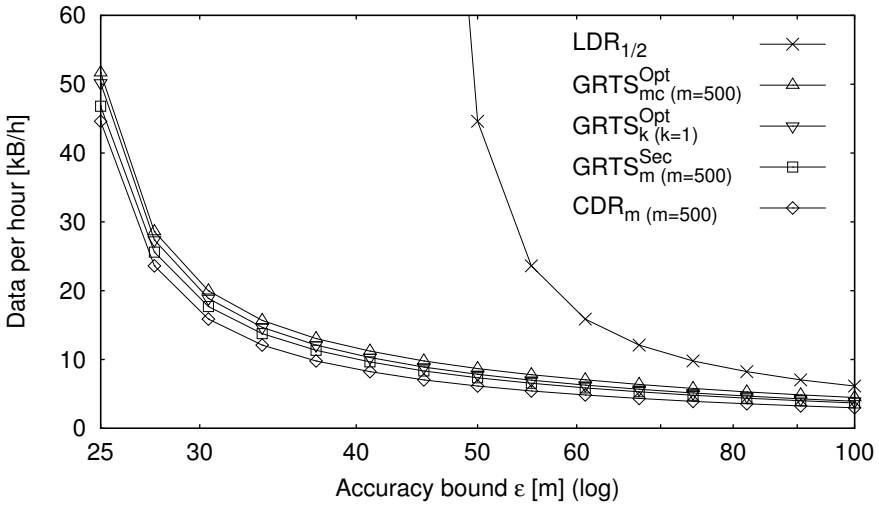


Figure 2.23: Amounts of data transmitted by major tracking approaches.

to add. Obviously, this causes GRTS to transmit a higher amount of data than CDR as illustrated in Figure 2.23. However, the additional amount of transmitted data is small compared to the higher reduction rates of the GRTS variants of more than a factor of two. For example, $\mathrm{GRTS}_{\mathrm{m}}^{\mathrm{Sec}}$ transmits only 5 to 23% more data than CDR. Assuming a header size of 28 byte (UDP/IP) per message, the difference is only 2 to 13%.

The differences between the GRTS realizations are caused by the different sizes of the variable part of $\vec{u}(t)$. In case of $\mathrm{GRTS}_{\mathrm{k}}^{\mathrm{Opt}}$ $(k = 1)$ and $\mathrm{GRTS}_{\mathrm{m}}^{\mathrm{Sec}}$, the variable part comprises only one vertex, whereas it may comprise multiple vertices with $\mathrm{GRTS}_{\mathrm{mc}}^{\mathrm{Opt}}$. Therefore, the update messages of $\mathrm{GRTS}_{\mathrm{mc}}^{\mathrm{Opt}}$ are slightly larger and replace more vertices on average than the update messages of the other two realizations.

The difference between $\mathrm{GRTS}_{\mathrm{k}}^{\mathrm{Opt}}$ $(k = 1)$ and $\mathrm{GRTS}_{\mathrm{m}}^{\mathrm{Sec}}$ of about 7% is caused by the fact that $\mathrm{GRTS}_{\mathrm{k}}^{\mathrm{Opt}}$ replaces the one vertex of the variable part more frequently than $\mathrm{GRTS}_{\mathrm{m}}^{\mathrm{Sec}}$, to achieve a better reduction rate. This shows that the two goals of the trajectory tracking problem – to minimize the communication cost and to minimize the number of vertices of the simplified trajectory – contradict for high reduction rates, as already hinted in Section 2.1.

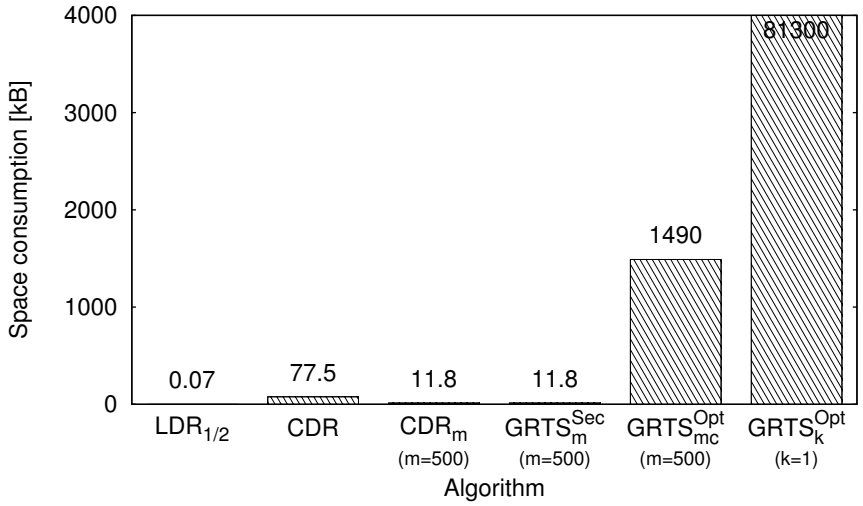


Figure 2.24: Space consumption of major tracking algorithms.

2.7.4 Computational Costs

We now analyze the maximum space consumptions and computing times of $\mathrm{LDR}_{\frac{1}{2}}$, CDR, $\mathrm{CDR_m}$, and important GRTS realizations. The space consumption is measured in kilobytes by summing up the space consumption of the different variables and arrays, particularly including the sensing history $\mathbb{S}$. The maximum computing time for processing a new sensed position is measured in processor ticks, i.e. clock cycles of the CPU, using the processor's time stamp counter. To filter out interrupts of the process under test, we simulated the trajectory tracking algorithms without other user processes and repeated each measurement ten times.

Figure 2.24 shows the maximum space consumption of the tracking algorithms for all GPS traces and ϵ values. The space consumption of $\mathrm{LDR}_{\frac{1}{2}}$ is negligible since it does not store a sensing history. In our simulations, the space consumption of CDR is well below 100 kB, although the size of sensing history of CDR is theoretically unbounded. Note that space consumption of CDR without the optimization of the sensing history proposed in Section 2.4.2 is 151.1 kB. Thus, the optimization saves 49%.

More important, the space consumption of $\mathrm{CDR_m}$ with $m = 500$ is only

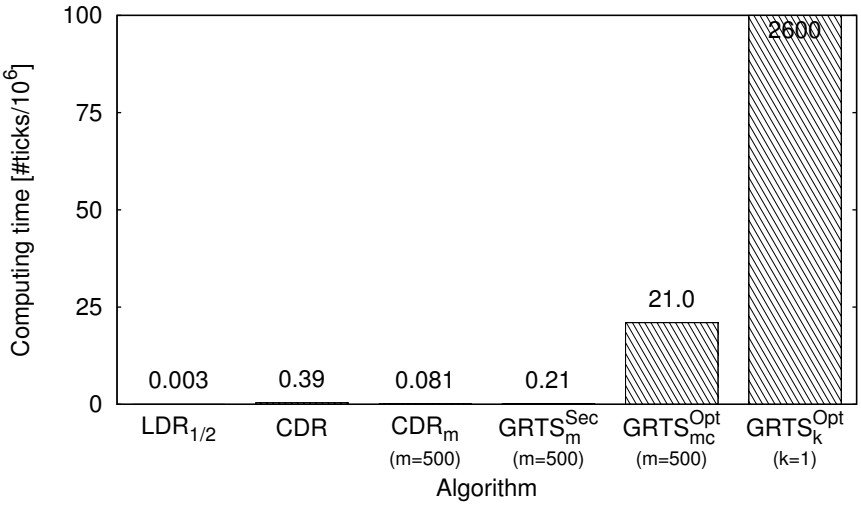


Figure 2.25: Maximum computing times of major tracking algorithms.

11.8 kB, despite the fact that there is no noticeable difference between the reduction performance of CDR and $\mathrm{CDR_m}$ with this parameterization.

$\mathrm{GRTS_m^{Sec}}$ with $m = 500$ also consumes only 11.8 kB since the section heuristic resembles the section condition of CDR and does not require any extensive data structures in addition to $\mathbb{S}$.

$\mathrm{GRTS_{mc}^{Opt}}$ with $m = 500$, in contrast, consumes 1.49 MB as it constructs a graph with up to $m \cdot (m-1)/2$ edges over $\mathbb{S}$. Yet, the space consumption is bounded to this value, which can be seen from the fact that in our simulations $\mathrm{GRTS_k^{Opt}}$ with $k = 1$ consumes up to 55 times more space – although the reduction performance of $\mathrm{GRTS_{mc}^{Opt}}$ with $m = 500$ is higher.

The huge space consumption by $\mathrm{GRTS_k^{Opt}}$ is reflected in the maximum computing time per position fix, given in Figure 2.25 in million ticks. The numbers in this figure can be approximately considered as milliseconds on a 1 GHz processor of a smartphone. This shows that the computational costs of $\mathrm{GRTS_k^{Opt}}$ are too high for practical use. The computing time may even exceed the sensing period T_S.

The maximum computing time of $\mathrm{GRTS_{mc}^{Opt}}$, in contrast, is only about 21 ms on a 1 GHz processor and thus only a fraction of the typical sensing period of $T_\mathrm{S} = 1\,\mathrm{s}$.

Nevertheless, the computational costs of $\text{GRTS}_{\text{mc}}^{\text{Opt}}$ are huge compared to $\text{GRTS}_{\text{m}}^{\text{Sec}}$, namely about factor 1000. Therefore, $\text{GRTS}_{\text{mc}}^{\text{Opt}}$ should be preferred to $\text{GRTS}_{\text{m}}^{\text{Sec}}$ only if the moving object has sufficient computational resources and reduction efficiency is of highest priority.

2.7.5 Conclusions for the Selection of a Tracking Approach

With CDR, CDR_{m}, and the various GRTS realizations, we proposed and evaluated almost ten approaches for real-time trajectory tracking. We also explained and showed that the communication cost and the number of vertices of the simplified trajectory cannot be completely minimized both together since these two goals contradict to some (small) degree. Figure 2.26 depicts all the trajectory tracking approaches and their relations.

As stated above, $\text{GRTS}_{\text{m}}^{\text{Opt}}$ is completely outperformed by $\text{GRTS}_{\text{mc}}^{\text{Opt}}$, whereas with $\text{GRTS}_{\text{mc}}^{\text{Sec}}$ the compression approach of GRTS_{mc} is superseded by the optimization of $\mathbb{S}$ proposed in Section 2.5.6. Therefore, those approaches are dimmed in Figure 2.26.

Moreover, in consideration of the extensive computational costs of $\text{GRTS}_{\text{k}}^{\text{Opt}}$, we advise to use tracking approaches with bounded space consumptions and computing times. For this reason, we deem CDR_{m}, $\text{GRTS}_{\text{m}}^{\text{Sec}}$, and $\text{GRTS}_{\text{mc}}^{\text{Opt}}$ to be particularly suited for practical use, depending on the actual requirements and resources in a given application scenario:

1. $\text{GRTS}_{\text{m}}^{\text{Sec}}$: This approach affords high reduction performance at comparatively low computational costs. Besides, it is simple to implement compared to realizations of GRTS with the optimal line simplification algorithm by Imai and Iri [II88]. Therefore, we suppose that $\text{GRTS}_{\text{m}}^{\text{Sec}}$ meets the requirements of most use cases.

2. $\text{GRTS}_{\text{mc}}^{\text{Opt}}$: If reduction has maximum priority and the moving objects have sufficient computational power, this approach should be used since it affords to reach almost best possible reduction rates, depending on m. The average reduction rates of $\text{GRTS}_{\text{m}}^{\text{Sec}}$, in contrast, are 10 to 15% below the best possible ones.

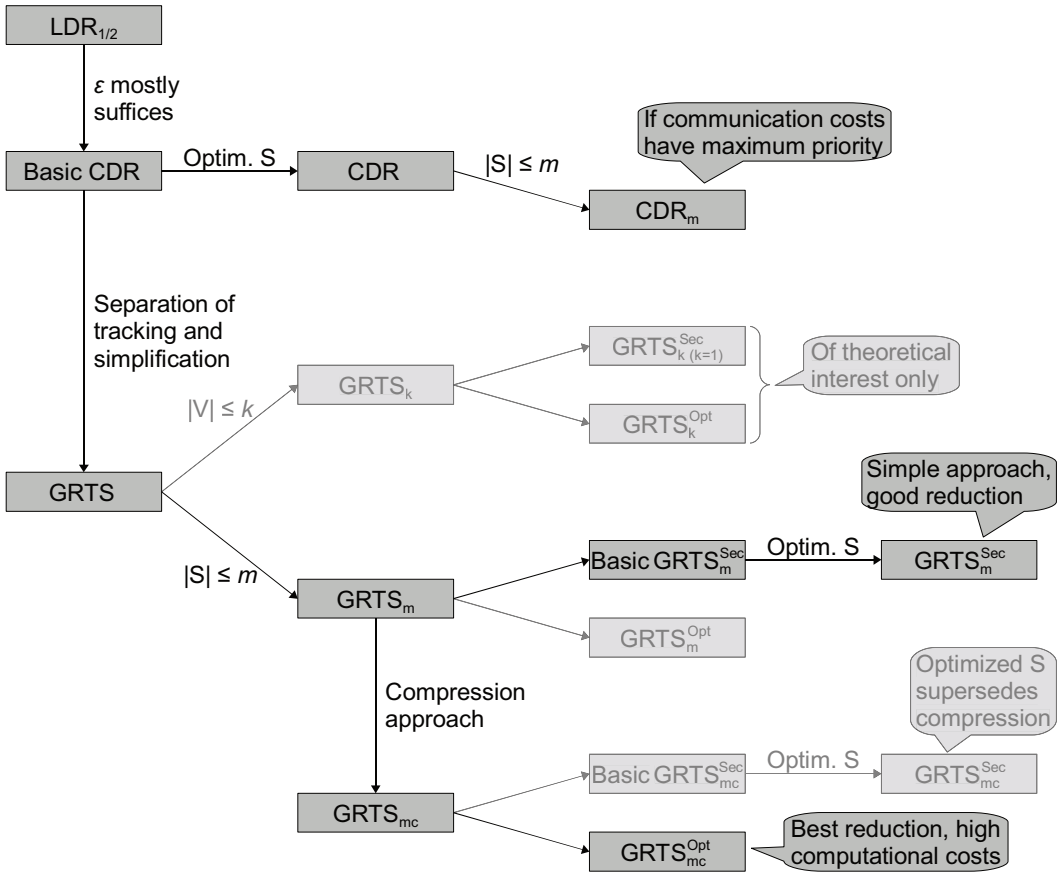


Figure 2.26: Overview of proposed real-time trajectory tracking approaches.

3. $\mathrm{CDR_m}$: In case that communication costs have maximum priority, $\mathrm{CDR_m}$ should be used since it minimizes the amounts of transmitted data. However, note again that $\mathrm{CDR_m}$ reaches only about one third of the reduction of $\mathrm{GRTS_m^{Sec}}$ and $\mathrm{GRTS_{mc}^{Opt}}$, even for large ϵ.

2.8 Prototypical Implementation

In this section, we present an implementation of GRTS together with a fully functional MOD system, consisting of two components named **mobile component** and **MOD server**. Furthermore, Google Earth is used as sample client application to query and visualize the trajectories. Figure 2.27 depicts the architecture of

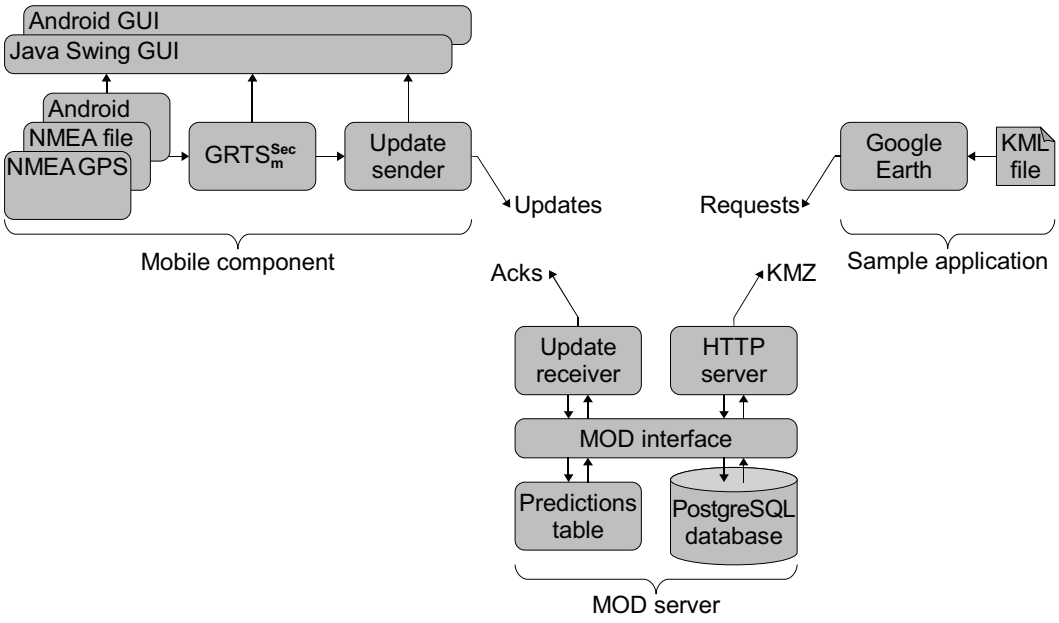


Figure 2.27: System architecture of prototypical implementation.

the system. Next, we describe the different components.

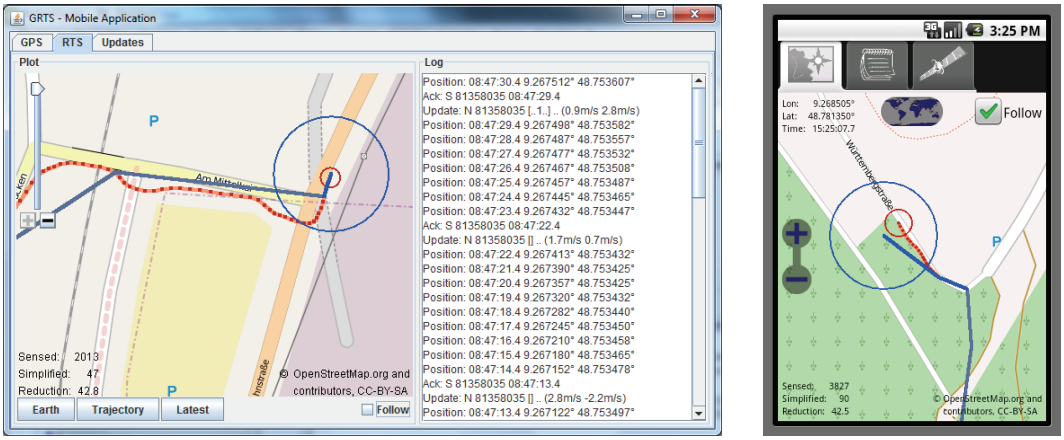
Mobile component is executed by each object being tracked using a subnotebook or smartphone with a GPS receiver. It is implemented in the Java programming language using a modular architecture and consists of four subcomponents.

NMEA GPS, **NMEA file**, and **Android GPS** are three alternative subcomponents for obtaining position data from GPS receivers that output standard NMEA 0183 sentences, previously recorded NMEA traces, or the Location Manager of the Android operating system, respectively.

GRTS$_m^{Sec}$ is an implementation of the GRTS$_m$ algorithm with the section heuristic as described in Section 2.5.6.

The **update sender** subcomponent implements ordered and reliable message passing between the moving object and the MOD server. For this purpose, the MOD server acknowledges each update message. If **update sender** does not receive an acknowledgment within a certain time span, it repeats the update message.

To illustrate the functioning of GRTS, we implemented two GUIs for differ-



(a) Java Swing GUI. (b) Android GUI.

Figure 2.28: Screenshots of graphical user interfaces for **mobile component**.

ent devices: Figure 2.28a is a screenshot of the Java Swing-based GUI. It shows sensed and the simplified trajectory drawn on the map of the OpenStreetMap project [OSM]. The large circle depicts the accuracy bound ϵ. The small circle illustrates the maximum sensing deviation δ. The right part shows a log of the GRTS algorithm. Two further panels provide information about the last sensed position and network statistics and show logs of the latest NMEA sentences and the sent and received messages.

Figure 2.28b analogously gives a screenshot of the GUI for Android-based smartphones.

MOD server uses a **PostgreSQL database** for persistent storage of the trajectories. The current predictions are stored in a main-memory table named **predictions table**. The **MOD interface** provides several spatial query types by means of different Java methods. Given a spatial query, it translates the query for the PostgreSQL database and the predictions table and merges their results. Similarly, the **MOD interface** splits updates into SQL updates and updates to the predictions table.

A separate subcomponent called **update receiver** listens for update messages from the moving objects, passes them to **MOD interface**, and acknowledges them.

Sample application. Google Earth is used as sample application to visualize all

trajectories stored by the MOD. For this purpose, it is launched with a small file in the Keyhole Markup Language (KML). This file contains the name of the host executing the MOD server and instructs Google Earth to query the MOD server for a KML representation of all trajectories once per second using HTTP. A lightweight **HTTP server** attached to the MOD server receives those requests, queries the MOD correspondingly for all trajectories, translates the result into KML, compresses it to KMZ, and sends a response to the Google Earth client.

By comparing the trajectories shown on the mobile devices with the trajectories shown in Google Earth, it is possible to verify visually that the simplification is performed in real-time.

Our implementation slightly extends the GRTS protocol by a timeout mechanism to cope with failures. If the MOD server does not receive an update message from a certain object for 90 s, it terminates the trajectory with the prediction origin as last vertex. The moving objects, in turn, transmit a new update message 60 s after the last update, even if not required by LDR.

Moreover, the update messages and acknowledgments are encrypted using the Advanced Encryption Standard (AES) with pre-shared keys.

Besides failures, clock synchronization is a critical issue. For correct query results, the clocks of the MOD server and the moving objects should be synchronized to UTC within a few tens of milliseconds or better.

Synchronizing the clocks of the moving objects is no problem in practical use since GPS receivers provide very accurate timestamps. Then, the clock of the MOD server can be synchronized by means of one or more (trustworthy) moving objects. For this purpose, each update message includes its sent time and the one-way latency. The latency is estimated from measurements of the round-trip times, taking the processing times at the MOD server into account. For this purpose, the processing times are measured and included in the acknowledgments.

During tests and demonstrations with replayed NMEA traces, the clocks of the moving objects are not synchronized as well, due to the absence of current GPS information. For these purposes, the above scheme is reversed as follows: For each moving object, the MOD server estimates the skew between its own

clock and the object's clock using the sent times and latency estimations given in the update messages. Then, it shifts the timestamps of all positions given in the update messages by this skew, respectively. Thus, it synchronizes these times to its own clock.

We conducted several experiments driving a car equipped with an OQO sub-notebook [OQO] and a Wintec WBT-300 GPS receiver [Wintec] providing four position fixes per second. This update rate particularly allows tracking fast objects with small ϵ. During our experiments, we used $\epsilon = 25\,\mathrm{m}$. Besides four network outages lasting several minutes, the prototypical tracking system successfully allowed for tracking the car and its trajectory for more than nine hours from several home computers.

During this experiment, we measured a reduction rate of 70. Per hour, only 60 kB of data were transmitted to the MOD server, including all communication overhead such as retransmissions due to lost UDP packets. These experimental results coincide with the results of our simulations (cf. Figure 2.15 in Section 2.7.2 and Figure 2.23 in Section 2.7.3).

2.9 Related Work

In this section, we first give a brief overview to line simplification algorithms in general before we discuss existing approaches for trajectory simplification in particular. Finally, we consider existing Internet services for tracking moving objects' trajectories. We omit dead reckoning protocols for tracking the current position of moving objects since these have been analyzed comprehensively in Section 2.3.

Line simplification refers to a multitude of algorithmic problems on approximating a given polyline by a simplified one with fewer vertices. The two basic problem classes are:

1. min-#: Minimizing the number of vertices of the simplified polyline under a given accuracy bound.

2. min-ϵ: Minimizing the deviation between the two polylines under a given number of vertices for the simplified one.

The min-# problems can be further classified by the *dimensionality* of the underlying space (e.g. $\mathbb{R}^2$ or $\mathbb{R}^3$), the *distance metric* to measure the distance between two points (e.g. Manhattan distance (L_1), Euclidean distance (L_2), or uniform metric), and the *error measure* to determine the distance between two polylines from the pairwise distances of their points [AHPMW05]. For the latter, most works implicitly consider the Hausdorff distance, defined as the largest distance from an arbitrary point of the one polyline to the closest point of the other polyline. However, there also exists works considering the Fréchet distance (e.g. [AHPMW05, AdBHZ07]).

According to these criteria, efficient real-time trajectory tracking can be considered as min-# problem in the case of Hausdorff distance under the (time-)uniform distance metric in $\mathbb{R}^{1+d}$ with $d = 2$ or 3, as explained at the beginning of the chapter.

As further explained above, the Douglas-Peucker algorithm [DP73], the optimal algorithm by Imai and Iri [II88], and the section heuristic [MdB04, AHPMW05, HGNM08] are three prominent approaches for min-# simplification. Several works including [HS94] and [GKM+07] propose variants of the Douglas-Peucker algorithm with improved running times. Similar applies to the optimal algorithm but limited to $\mathbb{R}^2$ and specific distance metrics and error measures (e.g. [CC92, Var96, AV00]). This is the reason why we considered the original algorithm by Imai and Iri in this chapter.

Cao et al. [CWT06] discuss the use of the Douglas-Peucker heuristic for *offline* trajectory simplification. They consider four different distance metrics, including the time-uniform distance metric, and compare the Douglas-Peucker heuristic against the optimal algorithm regarding reduction performance and computing time. Their results on the reduction performance are in line with our results. For the distance metric E_2, which disregards the temporal component of the trajectories but only considers the Euclidean distance, they use a variant of the optimal algorithm by Chan and Chin [CC92] tailored to $\mathbb{R}^2$. Nevertheless, they measure more than factor 1000 between the computing times of the optimal algorithm and the Douglas-Peucker heuristic. Note that the disregard of the temporal component by the distance metric E_2 is problematic for many applications since the point of the simplified line section $\overline{u_j\, u_{j+1}}$ that

is closest to a given sensed position s_i with $u_j.t \leq s_i.t \leq u_{j+1}.t$ generally differs from the interpolated position $\overline{u_j\, u_{j+1}}(s_i.t)$.

Gudmundsson et al. [GKM+07] likewise propose the use of the Douglas-Peucker heuristic for offline trajectory simplification and argue against the optimal algorithm because of its running time.

With $\text{GRTS}_{\text{mc}}^{\text{Opt}}$ we successfully applied the optimal algorithm for *online* simplification – at acceptable computing times and with reduction rates close to the best possible offline rates and significantly greater than the reduction rates of the Douglas-Peucker heuristic.

Meratnia and de By [MdB04] propose the section heuristic for online and offline trajectory simplification but with a different error measure based on the *average* deviation between corresponding points of the original and the simplified trajectory. In detail, they refer to the section heuristic as Opening-Window algorithm (OPW) and distinguish two variants. The one with the better reduction efficiency, which corresponds to the section heuristic as explained here, is called BOPW-TR. Due the different error measure, the maximum deviation between the original trajectory and the simplified one is not bounded and depends on the simplification algorithm.

This also applies to threshold-guided sampling, a heuristic for online trajectory simplification proposed in [PPS06b]. It adds the most recent sensed position s_{R} as vertex to the simplified trajectory only if the speed or direction of the latest velocity compared to the velocity between the previous sensed positions and the average velocity between the last two vertices of the simplified trajectory exceeds a certain threshold. Therefore, the deviation between the original trajectory and the simplified one is not bounded.

In [PPS06a], the same authors further propose the AmTree, a data structure for managing an incoming stream of sensed positions with constant storage consumption. The AmTree "forgets" more and more positions over time so that fewer positions are known for the far past than for the recent past. Again, the deviation between the original trajectory and the resulting simplified one is not bounded by some predefined accuracy.

In [HGNM08], a software component for online preprocessing position data of mobile objects is presented. The component aims at reducing the position

data to be stored by a database according to a given accuracy bound. The authors propose five different reduction algorithms, where in fact only one – the section heuristic – performs line simplification, i.e. yields a *connected* simplified trajectory.

None of the above works considers real-time trajectory tracking for remote moving objects.

In [TJ07], Tiešytė and Jensen present an approach for real-time trajectory tracking based on LDR. They propose an algorithm for computing a connected trajectory on the basis of the linear predictions, which approximates the actual trajectory according to the same accuracy bound used with LDR. However, their findings only apply to pre-known routes like bus lines, i.e. movement in $\mathbb{R}^1$.

In [TCS+06], Trajcevski et al. prove that the simplified trajectory given by the origins of the linear predictions of LDR with accuracy bound ϵ approximates the actual trajectory by 2ϵ [TCS+06]. Based on this finding they conclude that $\text{LDR}_{\frac{1}{2}}$, i.e. LDR with $\epsilon' := \frac{1}{2}\epsilon$, allows for trajectory tracking with accuracy bound ϵ. As discussed in detail in Section 2.3, this approach is very conservative and therefore is outperformed by GRTS by factor five in terms of reduction efficiency.

In the last years, several tracking services have been launched on the Internet. Two of the most popular such services are Google Latitude [Latitude] and Yahoo! Fire Eagle [FireEagle].

Google Latitude allows users to share their current position – i.e. the position of their mobile phone – with certain other people. Users can adjust the accuracy of the position information provided to each other individually. Google states that the update frequency by the mobile application is determined by several factors including the remaining battery power and the current speed. If the mobile application runs in background, GPS is not used to preserve battery life [Latitude].

Hence, the service does not aim to track the user's whole trajectory according to a specific accuracy bound of few meters. Nevertheless, Google Latitude allows the user to query his or her past movement as sequence of recorded positions.

Fire Eagle provides an API that allows users to store their current position and to disseminate it to websites and applications of the user's choice. It does not store any past positions.

InstaMapper [InstaMapper] and Map My Tracks [MapTracks] are two services for tracking the whole trajectory of a user in real-time. Both offer respective applications for mobile phones as well as websites to view a user's trajectory on a map.

The mobile application of InstaMapper periodically transmits an update message every 5, 30, or 60 s to the server, depending on whether the user moves at $\geq$ 20 mph or less, and whether the user's trajectory is currently viewed by others or not. Thus, it does not perform any line simplification in the classical meaning.

Similar applies to Map My Tracks. Its API even allows transmitting *every* sensed position in real-time or in batch updates to the server.

Besides, none of these services uses dead reckoning for tracking the current position. Therefore, the time domain of a trajectory managed by a service does not increase continuously.

Hence, with GRTS, we proposed the first approach for real-time trajectory tracking that clearly separates tracking from simplification and therefore can achieve near-optimal reduction rates at acceptable, bounded computing times.

2.10 Summary

In this chapter, we presented *Connection-Preserving Dead Reckoning* (CDR) and *Generic Remote Real-Time Trajectory Simplification* (GRTS) for tracking the trajectories of a large number of moving objects at a MOD efficiently – which is a prerequisite for supporting the primary context of such objects and for accessing any context information associated with their trajectories.

For this purpose, the objects sense their positions periodically but report only a subset of the positions to the MOD so that the resulting simplified trajectory approximates the actual movement according to some predefined accuracy bound. To inform the MOD about the current position, CDR and GRTS use dead reckoning.

However, while CDR is solely based on dead reckoning, GRTS separates the tracking of the current position from the simplification of the past trajectory. Therefore, GRTS outperforms CDR by more than factor two in terms of reduction performance whereas CDR minimizes the amount of data communicated over the wireless network.

For both, CDR and GRTS, we proposed optimized algorithms with bounded space consumption and computing time. In addition, we investigated different realizations of GRTS with two important line simplification algorithms and evaluated the resulting trade-off between computational costs and reduction efficiency.

The realization $\text{GRTS}_{\text{m}}^{\text{Sec}}$ with a simple line simplification heuristic affords substantial reduction performance at low computational costs. In detail, it reaches 85 to 90% of the best possible (offline) reduction rate at computing times of less than 0.25 ms on a 1 GHz processor. The realization $\text{GRTS}_{\text{mc}}^{\text{Opt}}$ with the optimal line simplification algorithm by Imai and Iri [II88], in contrast, may reach more than 97% of the best possible reduction rate at computing times of at most 21 ms.

3 Distributed Indexing of Space-Partitioned Trajectories

In this chapter, we address the fourth subproblem of how to provide efficient access to distributed dynamic context information, namely the distributed indexing of trajectories to enable efficient access to context information associated with the movement of mobile objects.

By the example of spatiotemporal queries in space-partitioned MODs, i.e. the primary context *location*, we show how to route queries along distributed trajectory segments of moving objects efficiently. For this purpose, we present the *Distributed Trajectory Index* (DTI), which allows for such efficient query routing by creating an overlay network for each trajectory. We further propose an enhanced index called DTI+S. It accelerates the processing of queries on aggregates of dynamic attributes, like the maximum speed during a time interval, by augmenting DTI with summaries of trajectory segments. A discrete-event simulator is used to show the effectiveness of DTI and DTI+S in a series of experiments. Finally, we discuss related work and give a brief summary of the chapter.

3.1 Preliminaries

When querying trajectory data or context information associated with trajectories, we can distinguish between two classes of spatiotemporal queries, *coordinate-based* and *trajectory-based* queries [PJT00, GS05]. Queries of the former class refer to all trajectories satisfying a certain spatial relationship to a specified region or point. For instance, a spatiotemporal range query, which returns all moving objects residing in a given region during a given time interval, belongs to that class. In contrast, queries of the latter class refer to the trajectory of a single moving object and return information concerning a specified position or segment of that trajectory. For example, such a query may

refer to the trajectory segment of a moving object – given by a time interval and the object's identifier – delivering the segment itself or just the segment's length. Both query classes are highly relevant to context-aware systems in general and MODs in particular and hence should be efficiently supported by appropriate index structures.

Managing the trajectories of large numbers of objects requires distributing the trajectory data to multiple servers. Two kinds of partitioning are conceivable, *spatial partitioning* and *object-based partitioning*. With spatial partitioning, a server stores all trajectory segments that overlap with the geographic service region associated with this server. Therefore, a moving object's trajectory data is typically distributed over multiple servers. In contrast, with object-based partitioning a moving object's trajectory is entirely stored on a single server, where the server responsible for a moving object is determined by a well-known mapping.

A critical issue with query processing in such a MOD is routing a given query to the servers that store the queried data. With respect to query routing, the two kinds of partitioning have different characteristics:

1. *Object-based partitioning:* Object-based partitioning is well suited for processing a trajectory-based query since the queried trajectory is stored on a single server, directly given by the partitioning scheme and the identifier of the queried object. On the other hand, a coordinate-based query has to be distributed to a large number of servers in general, since every server might store relevant data on the queried region and time interval.

2. *Spatial partitioning:* For a coordinate-based query, such as a range query, the set of servers that store relevant data is given directly by the queried region and the mapping from space to servers. If the queried region is small compared to the servers' service regions, then the query has to be routed to few servers only. An algorithm for efficient distributed processing of range queries has been proposed in [TDSC07]. On the other hand, for a trajectory-based query, the set of servers that store the queried data not only depends on the query parameters and the spatial partitioning but also on the actual route of the queried trajectory.

Partitioning	**Object-based**	**Spatial**
Trajectory-based queries	+	?
Coordinate-based queries	−	+
Update-aware distribution	−	+

Table 3.1: Characteristics of partitioning schemes.

At first glance, the characteristics of spatial partitioning seem to be complementary to the ones of object-based partitioning. Yet, there is a crucial difference in favor of spatial partitioning: In case of an object-based partitioning, potentially every server can store relevant data for a given coordinate-based query, even if the queried range is small. This, however, does not apply to spatial partitioning and trajectory-based queries: For a given trajectory and a short time interval, the relevant data is stored by only few neighboring servers due to the functional dependency between space and time induced by the movement of the respective object.

Another important advantageous characteristic of spatial partitioning is that current position information can be stored close to the moving objects by distributing the servers according to their service regions. We refer to this approach as *update-aware distribution*. It may reduce the overhead for position updates substantially [PS01, LR02]. Therefore, spatial partitioning and update-aware distribution are also widely used in scalable location services [PS01, LR02, ZZL07].

Table 3.1 briefly summarizes the characteristics of both kinds of partitioning. The question mark in Table 3.1 shall indicate that spatial partitioning does neither inherently prevent efficient processing of trajectory-based queries nor that it immediately allows for an efficient processing scheme.

In this chapter, we therefore present the *Distributed Trajectory Index* (DTI) [LDR08b], enabling efficient routing of trajectory-based queries in space-partitioned MODs. A DTI realizes an overlay network of servers storing segments of a certain trajectory. Our evaluations show that DTI reduces the time for routing trajectory-based queries to the relevant servers by up to 69% compared to routing without DTI.

We further present an enhanced index called *DTI+S* optimizing aggrega-

tion queries on trajectories like "What was the maximum speed of object o during time interval $[t_a, t_b]$?". DTI+S optimizes DTI for such queries by augmenting the index with summaries of trajectory segments. For a MOD system consisting of 1000 servers, our evaluation shows that DTI+S can reduce the time for processing aggregation queries by two orders of magnitude compared to processing without DTI+S. DTI and DTI+S achieve these savings with negligible storage consumptions compared to the trajectory data being stored. In our evaluations DTI+S accounts for less than 2.3% of the overall storage consumption.

3.2 Assumptions and Notation

We consider a space-partitioned MOD that stores the trajectories of a set of moving objects with identifiers o_1 to o_m, where m is in the magnitude of thousands or millions in general.

Consistent with CDR and GRTS (cf. Chapter 2) and most works on MODs (e.g. [LFG+03,MGA03,GS05]), the trajectory of an object o is a spatiotemporal polyline, represented by a sequence of timestamped positions $u_1, u_2, \ldots$. Each vertex u_i is a data record with two attributes t and $\vec{p}$, where $u_i.t$ denotes the sensing time and $u_i.\vec{p}$ denotes the sensed position on the plane, sphere, or space, accordingly.

We refer to any connected clipping of a trajectory as *trajectory segment*. Such a segment simply can be specified by a time interval $[t_a, t_b]$ on a known trajectory.

The space-partitioned MOD is responsible for tracking moving objects in a certain geographic area, called *service area*. This area is partitioned into a set of disjoint *service regions*. Each service region is managed by an individual server s_i. Therefore, the MOD is composed of a collection of servers s_1 to s_n, each of which is responsible for a single service region. For any moving object, a server stores only positions that are located in its service region. Consequently, a moving object's trajectory generally is maintained by multiple servers, each storing the segment intersecting its service region. Moreover, a server only stores trajectory segments of those moving objects that have visited its service region at least once. We do not make any assumptions regarding the storage

and indexing of the trajectory segments within a server. Any index structure for past trajectory data such as the TB-tree [PJT00] can be used.

We call the server in whose service region object o is currently located the *current server* of o. $S(o)$ denotes the servers storing segments of the object's trajectory. $S(o,t)$ denotes the server storing the segment comprising timestamp t. Two servers are *neighbors* if their service regions adjoin. Figure 3.1 illustrates a space-partitioned MOD with seven servers s_1 to s_7.

We assume that the servers compose a geographic overlay network with respect to their service regions. Given a message and a target point $\vec{p}$, they are able to route the message geographically from any server to the server whose service region contains $\vec{p}$. We do not make any further assumptions about how geographic routing is implemented. Any greedy geographic routing scheme can be used (e.g. [Kle00, RFH+01]).

The current positions of the moving objects can be determined by any kind of position sensors located on the moving objects (e.g. GPS receivers) or within the infrastructure surrounding the objects. We assume that an object's movements are reported to the corresponding server using an appropriate tracking protocol for trajectories such as GRTS proposed in Chapter 2. If an object moves from one service region to another, a *handover* is performed, which supplies the address of the object's previous current server to the new one.[1]

Any entity connected to the network of servers can act as a *client* of the MOD. A client can issue a query at an arbitrary server. The servers process the query in a distributed fashion and transmit the result to the client.

The MOD is capable of processing coordinate- *and* trajectory-based queries. In this chapter, however, we address the latter class of queries only, as justified in Section 3.1. An algorithm for processing range queries in a space-partitioned MOD is given in [TDSC07]. As stated above, trajectory-based queries refer to the trajectory segment of a single object. In particular, a query $q_{\text{type}}(o, t_\text{s}, t_\text{e})$ refers to the segment specified by time interval $[q.t_\text{s}, q.t_\text{e}]$ of moving object $q.o$. The type of the query defines what information concerning this *queried segment*

[1] With GRTS, the sensing history of the moving object should be cleared during a handover, to ensure that the stable part of the simplified trajectory comprises the current position in the new service region. Thus, the variable part is temporarily eliminated. This prevents later changes to the trajectory segment stored by the previous current server.

is returned to the client. In this chapter, we consider four representative types:

1. *Segment query:* the specified segment is returned as a list of positions. If $q.t_\mathrm{s} = q.t_\mathrm{e}$, only one timestamped position is returned. Hence, this query type also comprises position queries.

2. *Length query:* the length of the specified segment is returned.

3. *Max-speed query:* the maximum speed of object $q.o$ in the specified segment is returned.

4. *Region-relation query:* This type of query takes an additional parameter R defining a geographic region. The query returns the set of time intervals within the queried interval $[q.t_\mathrm{s}, q.t_\mathrm{e}]$ during which $q.o$ was located inside $q.R$.

In the following, $S(q)$ denotes the servers storing the queried segment of q. Note that $S(q) \subseteq S(q.o)$.

3.3 Basic Scheme

In this section, we introduce a basic scheme for processing trajectory-based queries in space-partitioned MODs. This scheme is improved by DTI and DTI+S in the subsequent two sections.

A client may send a query q to any server. Unfortunately, due to the spatial partitioning there exists no immediate mapping from the parameters $q.o$, $q.t_\mathrm{s}$ and $q.t_\mathrm{e}$ to the servers $S(q)$. Therefore, we need a mechanism for routing q to $S(q)$.

With our basic scheme, query routing comprises three phases: (1.) Routing q to any server $s_i \in S(q.o)$, (2.) routing q to $S(q.o, q.t_\mathrm{s})$ or $S(q.o, q.t_\mathrm{e})$, i.e. to the start or end of the queried segment, and (3.) routing q to the remaining servers of $S(q)$, i.e. traversing the queried segment to collect the queried information.

First phase. A server $s_i \in S(q.o)$ can be found by potentially searching all servers of the MOD, which obviously does not scale well. Instead we apply a *home server* scheme, where each moving object o has a home server that

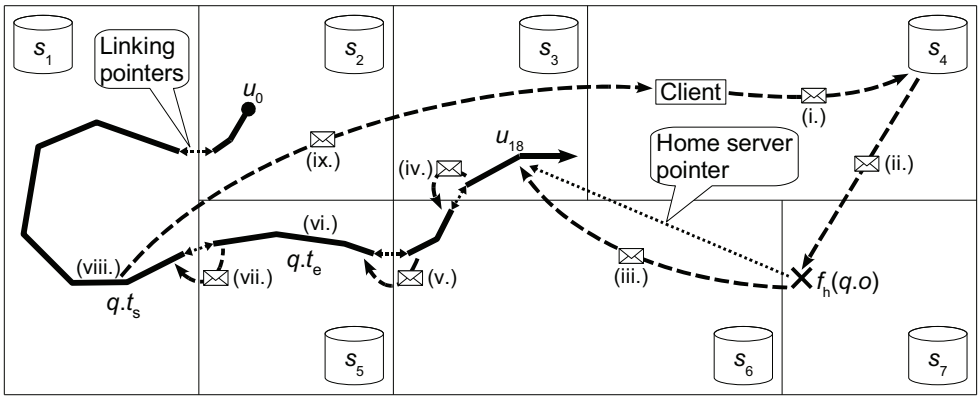


Figure 3.1: Query processing using the basic scheme.

knows (at least) one server of $S(o)$. Our approach is inspired by distributed hash tables, which are used to (indirectly) assign home server roles to the various servers of the MOD. For that purpose, all servers share a fixed hash function $\vec{f}_{\mathrm{h}}$ mapping the moving objects' identifiers to points in the service area: $\vec{f}_{\mathrm{h}} : \{o_1, \ldots, o_m\} \to \mathbb{R}^2$. The home server of moving object o is defined to be the server whose service region contains the geographic point $\vec{f}_{\mathrm{h}}(o)$. This server maintains a *home server pointer* u_{h} to some timestamped position u_i of o's trajectory. The pointer u_{h} is simply a *copy* of its target u_i. Hence, it is independent of the server storing u_i. This independence has the advantage that home server pointers are stable to reconfigurations, i.e. need not be updated if their target positions are relocated to other servers.

How are the targets of home server pointers selected? A simple mechanism is to select the moving object's initial position u_0. Of course, we can think of much more elaborate schemes, which adapt the pointer to frequently queried time intervals or which maintain more than one pointer. Since the DTI scheme is independent of any particular solution to that problem, we do not go into further detail here.

The server that first receives a query q from the client geographically routes q to $\vec{f}_{\mathrm{h}}(q.o)$, i.e. to the home server of $q.o$. The home server then geographically routes the query to $u_{\mathrm{h}}.\vec{p}$. This completes the first phase. If any server $s_i \in S(q.o)$ receives q on its way to $u_{\mathrm{h}}.\vec{p}$, the first phase is completed even earlier.

Second phase. The query q is routed to the server storing the beginning or end of the queried segment, i.e. to $S(q.o, q.t_\mathrm{s})$ or to $S(q.o, q.t_\mathrm{e})$. Without loss of generality, we only consider routing to $S(q.o, q.t_\mathrm{e})$ in the following.[2] Each server that receives q transmits it to the neighbor server storing the previous or rather subsequent segment that is closer to $q.t_\mathrm{e}$ until $S(q.o, q.t_\mathrm{e})$ is reached. We refer to this kind of linear routing along a trajectory as *trajectory-based routing*.

For this purpose, the servers maintain two *linking pointers* for each segment they store: The *backward linking pointer* points to the last position of the previous segment and the *forward linking pointer* points to first position of the subsequent segment. A linking pointer is a copy of the timestamped position it points to, like a home server pointer. This enables each server to decide locally on the next routing step.

The linking pointers are created as follows: For each moving object, the space-partitioned MOD maintains a data structure called *shadow object* (SO). The SO for object o is transferred from the current server of o to the next one as part of the handover procedure. Besides other information, the SO contains the latest trajectory information of o. For example, when using the GRTS protocol, the SO contains the variable and predicted part of the trajectory. When the tracking protocol signals a handover (cf. Section 3.2), the new current server informs the previous current server about the object's position within the new service region. The previous server creates the forward linking pointer. Then it transmits the SO to the new current server. This server creates the backward linking pointer and then updates the SO accordingly.

Third phase. Now, q is processed monotonously from $q.t_\mathrm{e}$ to $q.t_\mathrm{s}$ using trajectory-based routing. The server $S(q.o, q.t_\mathrm{e})$ processes q backwards in time until the end of the previous segment given by the backward linking pointer. Then, it transmits q and the partial result r to the corresponding neighbor server. This server further processes q on its segment until the end of its previous segment and merges its local result with r. This procedure is repeated until $S(q.o, q.t_\mathrm{s})$ receives q. This server processes q from the end of its segment

[2] The improved scheme DTI+S requires that q is routed to $S(q.o, q.t_\mathrm{e})$ since it processes q backwards in time from $q.t_\mathrm{e}$ to $q.t_\mathrm{s}$, cf. Section 3.5.

to $q.t_\mathrm{s}$ and merges the local result with r. Finally, it sends the complete result r to the client.

The type of r and its merging with local results depend on the query type. In case of a length query, r is the length of the segment processed so far. Local results simply are added to r. Similar applies to the other types of queries. Note that the result of a segment query is not an aggregated value but simply a copy of the queried segment. Therefore, the servers $S(q)$ can alternatively send their local results to the client directly instead of concatenating them in r.

Figure 3.1 illustrates the three phases: (i.) s_4 receives the query q from the client. (ii.) s_4 geographically routes q to $\vec{f}_\mathrm{h}(q.o)$, i.e. to s_7. (iii.) s_7 routes q to $u_{18}.\vec{p}$ according to its home server pointer $u_\mathrm{h} = u_{18}$. (iv.) The respective server s_3 sends q to its neighbor server s_6 as the queried time interval $[q.t_\mathrm{s}, q.t_\mathrm{e}]$ is before s_3's segment. (v.) s_6 sends the query to s_5 for the same reason. (vi.) s_5 starts processing q since its segment comprises $q.t_\mathrm{e}$. (vii.) Then, it sends q and the partial result r to its neighbor s_1. (viii.) This server completes the processing since its segment comprises $q.t_\mathrm{s}$. (ix.) Finally, s_1 sends r to the client.

3.4 Distributes Trajectory Index

Trajectory-based routing in the second phase can take many geographic routing hops and thus time. It depends on the trajectory's route, the service regions and the *temporal routing distance*, i.e. the time span between the segment of the first server of the second phase and $q.t_\mathrm{e}$. For example, for uniform movement and uniform service regions the number of hops linearly depends on the temporal routing distance.

The goal of the DTI scheme is to increase the efficiency of routing by creating a distributed index called DTI over the servers $S(o)$ of each trajectory. DTI stores additional pointers along the trajectory. These *DTI pointers* allow for direct routes from one server's segment to temporal distant segments stored by other servers. Thus, a DTI realizes an overlay network of servers in $S(o)$.

The DTI scheme employs the idea of maintaining multiple pointers spanning

different distances from skip lists [Pug90] and applies it to distributed indexing of trajectory data. More precisely, a DTI is created according to a perfect bidirectional skip list, illustrated in Figure 3.2a. The nodes of such a list are sequentially numbered starting at zero. The pointers have different *levels*. The pointers at level 1 compose a doubly-linked list over all nodes. The pointers at level 2 compose a doubly-linked list over all nodes with even sequence numbers. The pointers at level i compose a doubly-linked list over all nodes with sequence numbers divisible by 2^{i-1}. The *height* of a node is the maximum level of the pointers it maintains. Thus, the height l of a node with sequence number $k > 0$ is $l = \max\{i : 2^{i-1} \mid k\} = 1 + \max\{i : 2^i \mid k\}$. The height of node 0 is always equal to the maximum height of all other nodes.

Figure 3.2b illustrates how the skip list principle is applied to a DTI for a given trajectory. The DTI scheme selects individual positions to act as *anchors* of list nodes. The DTI implements such a logical node by a data record named *DTI node*. A DTI node is maintained by the server that stores the node's anchor. For example, s_4 stores node 4 with anchor u_{32}.

A DTI node contains its sequence number, its anchor, and the pointers to other DTI nodes. Such a *DTI pointer* is simply a copy of the anchor of the node it refers to. Thus, the DTI pointers are realized like linking pointers and home server pointers. Therefore, trajectory-based routing in the second phase of the basic scheme can be adapted easily to the more efficient *DTI-based routing*, explained below.

The DTI of an object o is extended incrementally during runtime as soon as the time span since the creation of the last DTI node exceeds a certain threshold T_{R} (e.g. one or few hours, cf. Section 3.6.2): After processing the next update message, the current server creates a new DTI node anchored at the latest position it stores.[3] Therefore, a server can store no DTI nodes at all as well as several DTI nodes regarding a certain segment. The creation of DTI nodes is explained in detail in Section 3.4.2.

[3] With GRTS, the anchor of the new DTI node is the last position of the first part of the simplified trajectory, since this part is not changed by future update messages. For this purpose, the moving object either has to use a realization where the number of vertices of the variable part is known (e.g. in particular $\mathrm{GRTS}_{\mathrm{m}}^{\mathrm{Sec}}$) or inform the server accordingly by an additional field in the update messages.

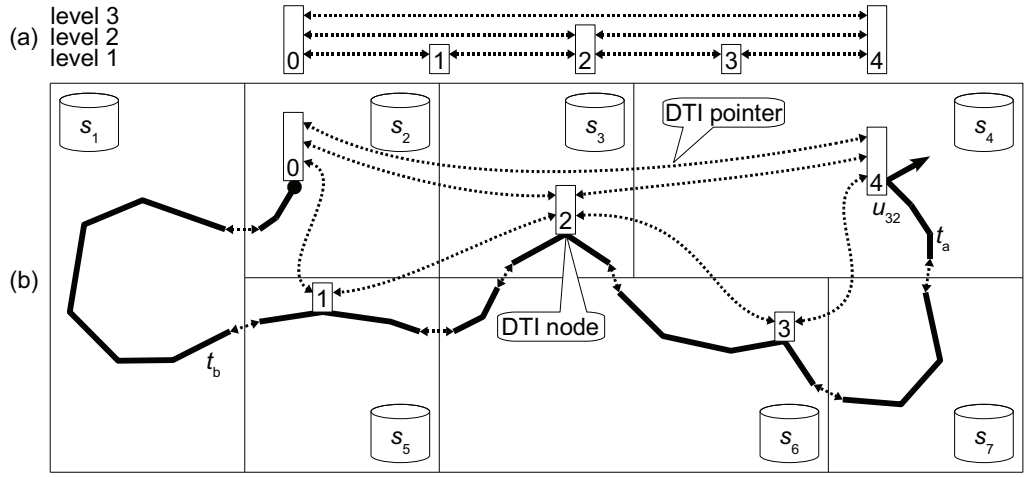


Figure 3.2: Skip list and DTI.

The distributed storage of the DTI nodes according to their anchors is useful since it provides direct network locality between index and indexed data. Furthermore, a DTI is robust regarding repartitioning of service regions due to the indirect addressing based on timestamped positions instead of network addresses, c.f. Section 3.4.4. Hence, a DTI realizes a *lightweight* overlay network on top of geographic routing, which again can be realized as an overlay network.

The overall storage consumption of a DTI linearly depends on its number of DTI nodes since the expected number of DTI pointers per node is *independent* of the number of DTI nodes. This can be seen from the following series: On average, half of the nodes maintain two pointers, one fourth maintain four pointers, one eighth maintain six pointers, and so on. Thus, the expected number of pointers at an arbitrary DTI node is equal to

$$\frac{1}{2} \cdot 2 + \frac{1}{4} \cdot 4 + \frac{1}{8} \cdot 6 + \ldots + \frac{1}{2^i} \cdot 2i + \ldots = \lim_{n \to \infty} 2 \sum_{i=1}^{n} \frac{i}{2^i} = 4 \ .$$

Accordingly, the expected height of an arbitrary DTI node is $l = 2$.

```
 1: receive query q and pointer u
 2: if stores segment that comprises q.t_e then
 3:     Enter third phase of basic scheme ...
 4: else
 5:     if maintains any pointer on q.o's trajectory then
 6:         u' ← pointer that is temporally closest to q.t_e
 7:         if |u'.t − q.t_e| < |u.t − q.t_e| then
 8:             u ← u'
 9:         end if
10:     end if
11:     geographically route q and u towards u.p⃗
12: end if
```

Figure 3.3: DTI-based routing algorithm.

3.4.1 DTI-based Routing

The fundamental principle of DTI-based routing is *greedy temporal routing*: Given a query q, each server geographically routes q to the target of the pointer u that minimizes $|u.t - q.t_e|$. Figure 3.3 gives a formal description of the algorithm executed at each server. The algorithm not only transmits q but also the pointer u currently being used since geographic routing along a DTI pointer may encounter servers that do not store any information about the queried trajectory.

If a server receives q and u, it first checks whether it stores the segment that comprises $q.t_e$ (line 2). If not, it selects the pointer u' that is temporally closest to $q.t_e$ from *all* pointers known to it – in particular DTI pointers, but also linking pointers and possibly even the moving object's home server pointer (line 6). If u' is even closer to $q.t_e$ than u, it replaces u accordingly (lines 7 – 8). Finally, it geographically routes q and u towards u's target. For example, if server s_3 in Figure 3.2b receives a query with $q.t_e = t_a$, then DTI-based routing uses the forward DTI pointer from DTI node 2 to node 4 and thus saves two geographic routing hops compared to trajectory-based routing.

The routing algorithm above generally achieves a good routing performance due to the following advantageous properties of a perfect bidirectional skip list:

1. The bidirectionality allows for efficient routing forward and backward in time.

2. The levels guarantee that a server always maintains more DTI pointers to temporally close segments than to distant ones. Hence, the smaller the temporal distance to $q.t_\mathrm{e}$, the finer is the support by pointers.

3. The number of DTI pointers for routing from one DTI node to another node logarithmically depends on the temporal distance of the two nodes.

The importance of the latter two properties for efficient routing in overlay networks is well known from the seminal work of Kleinberg [Kle00]. The performance of DTI-based routing depends on the actual route of the trajectory, the performance of the underlying geographic routing algorithm, and the temporal density of the DTI nodes. The latter is determined by the rate $1/T_\mathrm{R}$ at which new DTI nodes are created. For a given T_R, uniform movement, and a fully meshed geographic overlay network, the number of geographic routing hops with DTI-based routing logarithmically depends on the temporal routing distance. It polylogarithmically depends on the temporal routing distance in case of a geographic overlay network with long-range links according to Kleinberg [Kle00].

The choice of the parameter T_R, which defines the time between two consecutive DTI nodes, is a trade-off between the overhead for storing and accessing the DTI pointers within a server and the performance improvement for query routing. However, our evaluation results in Section 3.6.2 show that suitable values for T_R range within about two orders of magnitude.

The routing performance particularly improves with *implicit shortcuts* – if a server stores more than one segment of the same trajectory and thus enables to jump locally to a temporally distant segment. Implicit shortcuts further show that the above algorithm only approximates the optimal routing path. Each server chooses the temporally closest pointer independent of whether the server at the pointer's target provides implicit shortcuts or not.

A second reason for suboptimal routing paths is illustrated in Figure 3.2b. If s_2 receives a query with $q.t_\mathrm{e} = t_\mathrm{b}$ then using the DTI pointer from DTI node 0 to 1 results in an additional geographic routing hop compared to routing along

the forward linking pointer from s_2's segment to s_1's segment. The reason is that the temporal distance to $q.t_e$, used for choosing the next pointer, only approximates the actual routing distance.

3.4.2 Creation of DTI Nodes

Once the time span since the creation of the last DTI node exceeds the threshold T_R, the current server of the respective object o creates a new node when receiving the next position update from o. The DTI scheme stores the sequence number and anchor of the last DTI node in the SO for this purpose. If the current server creates a new DTI node, it determines the sequence number k of the new node and the backward DTI pointer to node $k-1 = k-2^0$ at level 1. Then it creates a *DTI-pointer message*, containing k and the anchor of the new DTI node. It geographically routes the message to the server that maintains node $k-1$. This server creates the opposite forward DTI pointer at level 1. If the new node's height l equals one, then the server immediately acknowledges the DTI-pointer message. If $l > 1$, it adds the backward DTI pointer to node $k-2$ to the message and routes the message to this pointer's target. The server that maintains DTI node $k-2$ creates the forward DTI pointer at level 2 to node k, adds its backward DTI pointer to $k-4 = k-2^2$ to the message, and then routes the message to node $k-4$. This procedure is repeated on ascending levels until node $k-2^{l-1}$ is reached. The server of node $k-2^{l-1}$ creates the forward DTI pointer on level l to the new node k and then routes the message back to the current server of the moving object. With it, the current server creates the new node's backward DTI pointers on the levels 2 to l. Thus, the creation of a new DTI node with height l involves routing the DTI-pointer message geographically to at most l other servers and creating $2l$ DTI pointers, where $l = 2$ on average, see above.

Figure 3.4 illustrates this procedure for creating DTI node 8. The Roman numbers denote the chronological order of the creations of the DTI pointers and the intermediate transmissions of the DTI-pointer message.

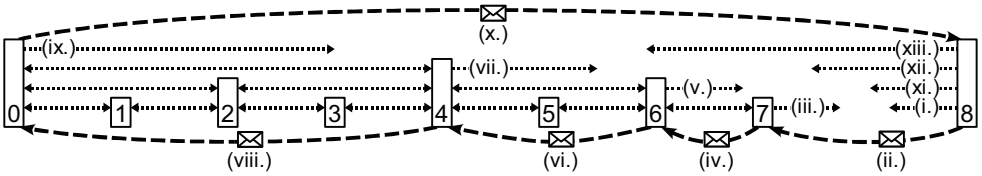


Figure 3.4: Creation of DTI node 8.

3.4.3 Home Server Pointer and DTI

DTI-based routing works if the home server pointer refers to any position on the moving object's trajectory. However, in order to reduce the average routing time it is advantageous if the home server pointer points to the anchor of a DTI node with a large number of DTI pointers. The DTI scheme guarantees that the home server pointer always points to the highest DTI node $k > 0$ by updating the home server pointer each time it creates a DTI node with $k = 2^i$.

3.4.4 Service Region Repartitioning

To achieve a scalable MOD service, it is essential that service regions can be split and merged locally without global reconfiguration. That is, splitting and merging must only affect a small number of servers. The basic scheme and DTI meet this requirement well: Anchors of DTI nodes and all kinds of pointers base on timestamped positions and not server addresses.

Splitting the service region of a server s_i with a new server s_j therefore only affects segments, pointers and DTI nodes of s_i. In detail, the following five tasks have to be performed:

1. Migrating each SO from s_i to s_j whose moving object is located in s_j's service region.

2. Splitting the segments stored by s_i according to the new service regions and creating linking pointers at the resulting splits.

3. Migrating the segments that are located in s_j's service region to s_j together with their linking pointers.

4. Moving each home server pointer of moving object o from s_i to s_j where $\vec{f}_{\mathrm{h}}(o)$, cf. Section 3.3, is located in the service region of s_j.

5. Migrating the DTI nodes, whose anchors are located in s_j's service region, and their summaries to s_j.

A similar procedure applies to merging two neighboring service regions.

3.5 DTI with Summaries

We introduced the DTI scheme in the previous section to reduce the routing overhead for the second phase of query processing. This scheme, however, does not affect the third phase, where trajectory-based routing may still cause a substantial routing overhead. Obviously, this overhead depends on the time span $q.t_{\mathrm{e}} - q.t_{\mathrm{s}}$. In the following, we extend the DTI scheme to increase the routing performance and thus query processing performance for the third phase.

The basic idea of the extended scheme, called DTI+S, is to attach *summaries* to the DTI pointers. A summary attached to a DTI pointer records aggregated information concerning the segment between the anchor of the respective DTI node and the pointer's target. For example, a summary may store the length of this segment and the maximum speed recorded for this segment. Summaries may substantially speed up processing in the third phase. In many cases, queries can be forwarded along DTI pointers using the aggregates stored in the attached summaries. Without those summaries, this information would have to be aggregated by additionally visiting all other servers of $S(q)$.

This approach can be applied to *any* type of query on an aggregate of a dynamic attribute that can be computed by partial aggregation [MFHH02], i.e. by means of smaller aggregates. It can be applied to length, max-speed, and region-relation queries in particular. For these queries, each summary stores the length of the segment it refers to, the maximum speed within the segment, and the minimum bounding rectangle (MBR) of the segment, respectively. Clearly, summaries cannot be used for optimizing segment queries as the result of such a query is not an aggregated value.

In principle, summaries can be attached to forward and backward DTI pointers. Whether summaries are maintained for a single or both directions is a

trade-off between storage consumption and routing performance. In the following we assume that summaries are only stored with the backward pointers. Therefore, query processing in the third phase always starts at $q.t_e$. However, the scheme can be easily extended to provide summaries for both directions.

The summaries increase the average storage consumption per DTI node by a constant amount of data. Assuming that storing a timestamped position requires $3 \times 8 = 24$ byte and that an integer takes 4 byte, a DTI node without summaries but with anchor, sequence number, and DTI pointers – four on average, cf. Section 3.4 – consumes $24 + 4 + 4 \times 24 \approx 125$ byte. For the three kinds of aggregation queries considered here, a summary contains six floating-point values, i.e. it requires $6 \times 8 = 48$ byte. Thus, a DTI node with summaries consumes $125 + 2 \times 48 \approx 225$ byte on average.

3.5.1 Construction of DTI+S

The DTI+S scheme creates the summaries together with the backward pointers. For this purpose, each SO additionally contains a summary of the segment between the latest DTI node and the latest known positions. The current server of the moving object updates this summary with each position update. For example, in case of the segment length it increments the length given in the summary by the distance between the new position given in the position update message and the previous position.

If the current server creates a new DTI node with sequence number k the SO's summary yields the summary for the segment between the DTI node $k-1$ and the new node k, i.e. it belongs to the new backward DTI pointer at level 0. The summaries belonging to the backward pointers on higher levels are created by concatenating the SO's summary with summaries for segments between previous DTI nodes.

For this purpose, the DTI-pointer message for a new node k with height l is extended as follows: The servers of the nodes $k - 2^0, k - 2^1, \ldots, k - 2^{l-2}$ not only add the backward DTI pointer to the next node in the list to the message but also add the corresponding summary. When the server of the new node k finally receives the message, it computes the summaries for the backward pointers at the levels 2 to l: For the summary attached to the pointer at level

```
1: receive query q, pointer u and partial result r
2: while u.t > q.t_s do
3:     if has usable summary to cut short from u.t then
4:         determine summary that enables longest shortcut
5:         merge r with aggregate given in the summary
6:         u ← backward DTI pointer of the summary
7:     else if has segment that spans u.t then
8:         u' ← linking pointer to previous segment
9:         if has DTI node between u'.t and u.t then
10:            u' ← anchor of latest such DTI node
11:        end if
12:        if q.t_s > u'.t then
13:            u' ← interpolate timestamped position at q.t_s
14:        end if
15:        process q on local segment between u'.t and u.t
16:        merge r with result between u'.t and u.t
17:        u ← u'
18:    else
19:        geographically route q, u, and r towards u.p⃗
20:        return
21:    end if
22: end while
23: send r to client
```

Figure 3.5: Algorithm for DTI+S-based query routing and processing.

2, it concatenates the SO's summary and the summary on the segment between node $k - 2^0$ and node $k - 2^1$. For the summary at level 3, it concatenates the just created summary and the summary on the segment between the nodes $k - 2^1$ and $k - 2^2$, and so on.

Two consecutive summaries are concatenated by aggregating each pair of aggregates belonging together. For example, the lengths of the segments simply are added.

3.5.2 Query Processing

Routing with DTI+S in the third phase is also based on *greedy temporal routing*. However, a given query q only may be routed along a backward DTI pointer if the pointer's summary is *usable* for processing q. That is, the summary provides the needed information for processing q on the summary's time interval.

Whether a summary can be used depends on the query type. Therefore, we first explain the generic algorithm for DTI+S-based routing and processing. Then, we render the algorithm more precisely by discussing usable summaries for the types of aggregation queries considered here.

Figure 3.5 shows the generic algorithm executed at each server. It monotonously processes q from $q.t_\mathrm{e}$ to $q.t_\mathrm{s}$ backwards in time. The current progress is stored by the timestamp $u.t$ of the pointer u. Initially u is equal to the position at $q.t_\mathrm{e}$.

A server s_i that receives q, u and r repeatedly tries to process q from $u.t$ on. In each repetition it first tries to process q by means of a usable summary (line 3 – 6) and otherwise by the segment that comprises $u.t$ (lines 7 – 17). If it neither stores a usable summary nor the segment that comprises $u.t$, then it routes q towards the server of the remaining part of the queried time interval (lines 18 – 19). If the server completed query processing ($u.t \leq q.t_\mathrm{s}$ in line 2), it sends the aggregated result to the client.

If the server maintains any usable summaries, it greedily processes q by means of the one reaching furthest in the past. If it has no usable summary but stores the segment spanning $u.t$, then it processes q by means of this segment. At every visited DTI node, s_i tries again to find a usable summary by repeating the while loop instead of simply traversing the segment linearly.

Whether a summary can be used depends on the query type. However, there exists a necessary condition, applying to all query types, namely that the time interval $[t_i, t_j]$ of a usable summary must contain $u.t$, i.e. $t_i < u.t \leq t_j$. For other summaries, the above algorithm cannot advance $u.t$ towards $q.t_\mathrm{s}$.

Next, we state the sufficient conditions for the usability of summaries for each type of aggregation query considered in this chapter:

- *Length queries:* A summary is usable if and only if its time interval $[t_i, t_j]$

holds for $q.t_s \leq t_i$ and $u.t = t_j$. Thus, a usable summary always belongs to the DTI node with anchor $u.t$.

- *Max-speed queries:* A summary is usable iff it fulfills one of the following two conditions: (1.) The summary's time interval $[t_i, t_j]$ holds for $q.t_s \leq t_i \leq u.t$ and $u.t \leq t_j \leq q.t_e$. (2.) The summary holds for the necessary condition and the maximum speed given in the summary is less or equal to the partial result r. For this reason – compared to length queries – the algorithm sometimes can even use a summary on a time interval $[t_i, t_j]$ with $t_i < q.t_s$ or $t_j > q.t_e$.

- *Region-relation queries:* A summary is usable iff it fulfills the necessary condition above and its MBR shows that the corresponding segment is located completely inside or completely outside the queried region $q.R$. That is, $q.R$ either completely contains the MBR or does not intersect the MBR. If the MBR only partially overlaps $q.R$, the summary is unusable. However, in this case the query possibly may be processed by a sequence of summaries on sub-segments of this summary's segment, since the MBRs of two consecutive segments generally cover much less space than the MBR of both segments.

3.6 Evaluation

In this section, we evaluate the effectiveness of DTI and DTI+S for query routing and processing in space-partitioned MODs. First, we explain the simulation setup. Then, we discuss the performance of DTI-based routing by considering the processing time in the first and second phase only. Finally, we give results on the overall processing time with DTI+S.

3.6.1 Setup

We implemented a discrete event simulator for space-partitioned MODs that allows for measuring the query processing times resulting from network latencies, disk I/O, and CPU.

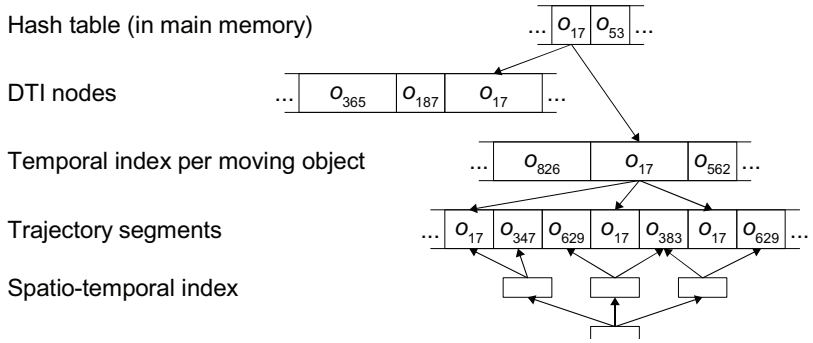


Figure 3.6: Storage layout.

We conducted a series of experiments, simulating a space-partitioned MOD with a service area of 4500 km × 2000 km (≈ Continental U.S.) over a time of $3 \cdot 10^7$ s ≈ 1 year. If not stated otherwise, the MOD is composed of 1000 servers with rectangular service regions with side lengths between 140 and 560 km. The network latencies for query routing are modeled using a real topology data set of the AT&T Internet backbone [LLN03]. Each server of the MOD is connected to the router closest to the center of its service region, assuming a network link with a delay of 3 ms. Then, the latencies are calculated using Dijkstra's algorithm.

For geographic routing of queries and other messages, each server maintains a shortest-path routing table containing *contacts* – network address and service region – for all neighbor servers. If not stated otherwise, the routing table further contains ten *long-range contacts* (LRCs) to distant servers, randomly chosen using inverse square distribution [Kle00]. If a server receives a message with target $\vec{p}$ then it greedily chooses the contact whose service region is closest to $\vec{p}$ and sends the message to the respective server.

Each server maintains a table for permanent storage of the trajectory segments within its service region and their linking pointers as well as a table for the respective DTI nodes and summaries. The latter table is clustered by the identifiers of the moving objects. Hence, all DTI nodes and summaries of a given object can be read in a single operation. We do not make any assumptions on the clustering of the trajectory segments table. Particularly, it can be clustered to optimize processing of coordinate-based queries using any

appropriate index such as the TB-tree [PJT00]. We only assume – as with TB-tree – that each page contains consecutive timestamped positions of a single moving object only. For efficient trajectory-based access to the trajectory segments table, a server maintains small, sparse temporal indexes for each moving object that ever entered its service region. These temporal indexes also are permanently stored, clustered by the moving object's identifiers. Figure 3.6 illustrates this storage layout. A hash table in main memory provides initial access to the pages that contain the DTI nodes and the temporal index of a given moving object.

In our simulations, we assumed a page size of 4 kB, a seek time of 10 ms, and a transfer rate of 30 MB/s. With these values and the above-mentioned storage requirements for the DTI nodes and summaries (24 byte for a timestamped position and $8 + 4 + 8 = 20$ byte for an entry in the temporal index), the disk I/O times can be calculated.

Each server further maintains a hash table for SOs and a hash table for home server pointers, not shown in Figure 3.6. Due to the frequent accesses to this data and the small storage requirements (about 200 byte per SO and 24 byte per home server pointer) both can be cached in main memory.

The CPU times for processing within a server were measured during our experiments on an Intel Xeon Linux Server with 3.0 GHz using 4 GB RAM. The following results, however, show that they are negligible compared to the network latencies and disk I/O times.

Regarding the processing of trajectory-based queries the mentioned storage layout is independent of the overall number of moving objects. Therefore, only a small set of moving objects has to be simulated. In our experiments we simulated 100 moving objects moving according to the mobility model proposed in [Haa97]: Each moving object starts at a random position with random direction and random speed $v \leq v_{\max} = 10\,\mathrm{m/s}$. Every 10 s it randomly changes its direction between -0.1 and $+0.1$ rad and its speed between -3 and $+3\,\mathrm{m/s}$. In contrast to [Haa97] the movement is not wrapped at the borders of the service area but reflected. The moving objects report their positions with $\mathrm{GRTS}_{\mathrm{m}}^{\mathrm{Sec}}$ (cf. Chapter 2) using an accuracy bound of 25 m. For each moving object, $\mathrm{GRTS}_{\mathrm{m}}^{\mathrm{Sec}}$ transmits $8.8 \cdot 10^6$ update messages during simulation and creates a trajectory with $8.7 \cdot 10^5$ vertices, i.e. position data records.

For $2 \cdot 10^7\,\mathrm{s} \approx 8\,\mathrm{months}$, the moving objects only report their positions. Then, for $10^7\,\mathrm{s} \approx 4\,\mathrm{months}$, each moving object additionally poses queries according to a Poisson process with rate $\lambda = 0.01\,\mathrm{s}^{-1}$. Thus, the simulation results are the averages of about 10^7 queries. We simulated a uniform mix of segment, length, max-speed, and region-relation queries. A moving object o randomly chooses the parameters of a query q as follows:

- $q.o$: Half of the queries refer to the client's trajectory, i.e. $q.o = o$. The other half refers to any other moving object.
- $q.t_\mathrm{e}$: The queried time interval ends 1000 to $10^7\,\mathrm{s}$ before current time t, where $\log[t - q.t_\mathrm{e}]$ is uniformly distributed. Thus, queries on the near past are more likely.
- $q.t_\mathrm{s}$: The queried time interval starts 1000 to $10^7\,\mathrm{s}$ before $q.t_\mathrm{e}$, where $\log[q.t_\mathrm{e} - q.t_\mathrm{s}]$ follows uniform distribution. Thus, queries on long time intervals are rare.

In case of a region-relation query, $q.R$ is a quadratic region with $l = 10$ to $1000\,\mathrm{km}$ side length, where $\log l$ is uniformly distributed. $q.R$ is randomly placed in the service area.

3.6.2 Routing Performance

To evaluate DTI-based routing independent of summaries, we only consider the first and second phase of query processing. For readability, we refer to the processing time in these two phases as *routing time* in the following.

Figure 3.7 shows the average routing time using the DTI scheme depending on the number of DTI nodes per trajectory created within simulation time, i.e. depending on the temporal density of the DTI nodes determined by the rate $1/T_\mathrm{R}$ at which new DTI nodes are created. For that purpose, T_R was varied between 1/8 and 8192 h in factors of two, i.e. from 450 to about $3 \cdot 10^7\,\mathrm{s}$. Figure 3.7 shows three curves: one for geographic routing without LRCs, with 10 LRCs, and with LRCs to all other servers, i.e. a fully meshed geographic overlay network. In the latter case, geographic routing to any point $\vec{p}$ only requires one geographic routing hop.

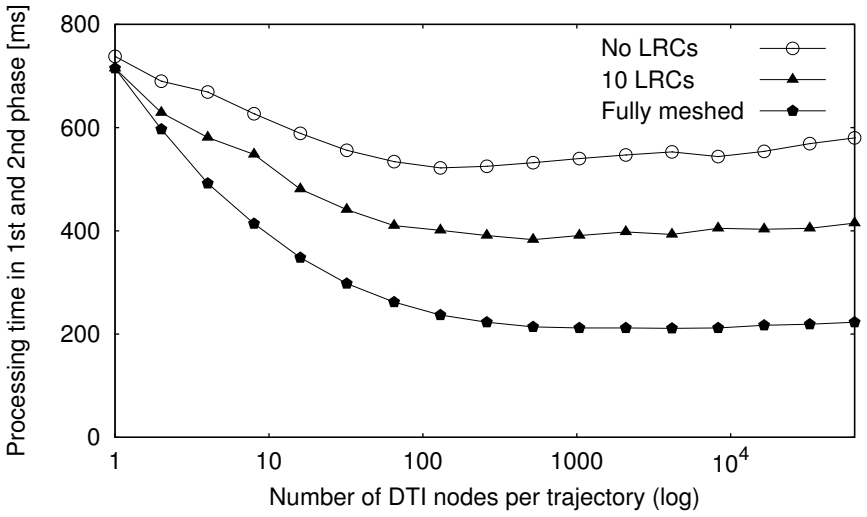


Figure 3.7: Routing time depending on DTI nodes.

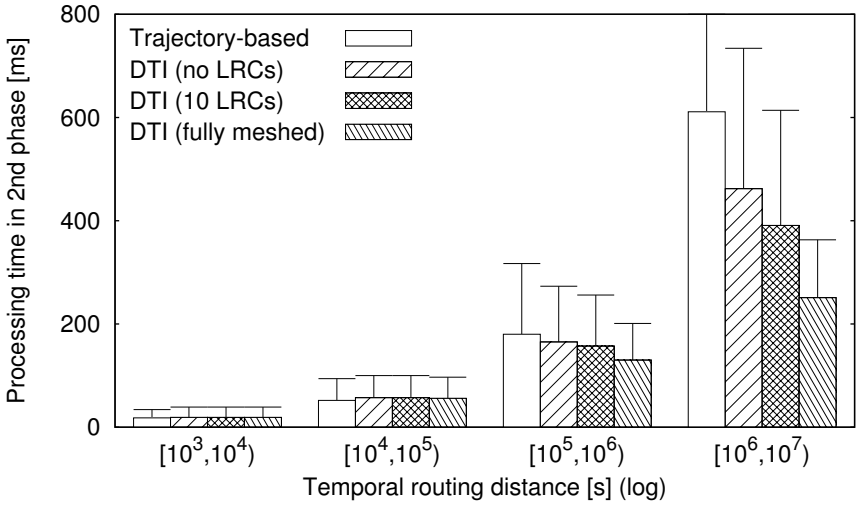


Figure 3.8: Routing time depending on routing distance.

With one DTI node per trajectory, DTI-based routing degenerates to trajectory-based routing. In this case, LRCs are useless except for the first phase, i.e. routing to the home server and along the home server pointer. Therefore the routing times with one DTI node are very similar for the different numbers of LRCs, between 715 and 738 ms. These times seem small compared to the

fact that each trajectory is partitioned to about 1900 segments on average, distributed to the major part of the servers. The reason is that trajectory-based routing also uses implicit shortcuts on servers storing two or more segments of the queried trajectory, cf. Section 3.4.

Nevertheless, with increasing numbers of DTI nodes the routing times significantly decrease. As expected, the more LRCs each server maintains, the larger are the savings – up to 29% (no LRCs), 46% (10 LRCs), or 70% (fully meshed). The maximum savings are reached with about 100 to 10^4 DTI nodes per trajectory, more precisely with $T_{\mathrm{R}} = 64$ to 1 h. For more than 10^4 DTI nodes, the routing times slightly increase due to the disk I/O for reading the additional DTI nodes. This shows that per trajectory one DTI node every few hours is sufficient for efficient DTI-based routing in our scenario. It further shows that the optimal routing performance is achieved also if T_{R} varies within about two orders of magnitude.

In all subsequent simulations, we used $T_{\mathrm{R}} = 4$ h, resulting in 2080 DTI nodes per trajectory within simulation time.

Figure 3.8 depicts the routing time in the second phase over the temporal routing distance, i.e. the time-span to cover in the second phase. As expected, DTI does not improve routing for short temporal routing distances. For temporal distances between 1000 and 10^5 s, routing with the basic scheme as well as routing with the DTI scheme both require only about 19 ms in the second phase. Yet, for long distances – 10^6 to 10^7 s – and a fully meshed geographic overlay network the DTI scheme reduces the routing time by 69%. With 10 LRCs per server, it reduces the second phase by 36% and even without LRCs the DTI scheme saves 24%.

3.6.3 DTI+S-based Routing and Processing

Next, we evaluate the DTI+S scheme by measuring the *overall processing time*, i.e. the sum of processing times in all three phases. Note that this time does not include the transmission of the result to the query issuer but only the processing within the space-partitioned MOD system.

Figure 3.9 shows the overall processing times with and without DTI+S for the different query types itemized to network latencies, disk I/O times, and

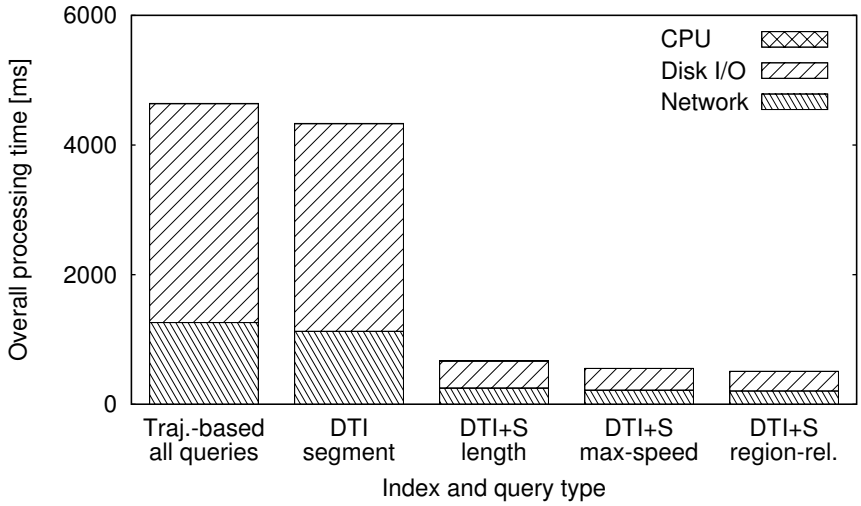


Figure 3.9: Itemized processing time per query type.

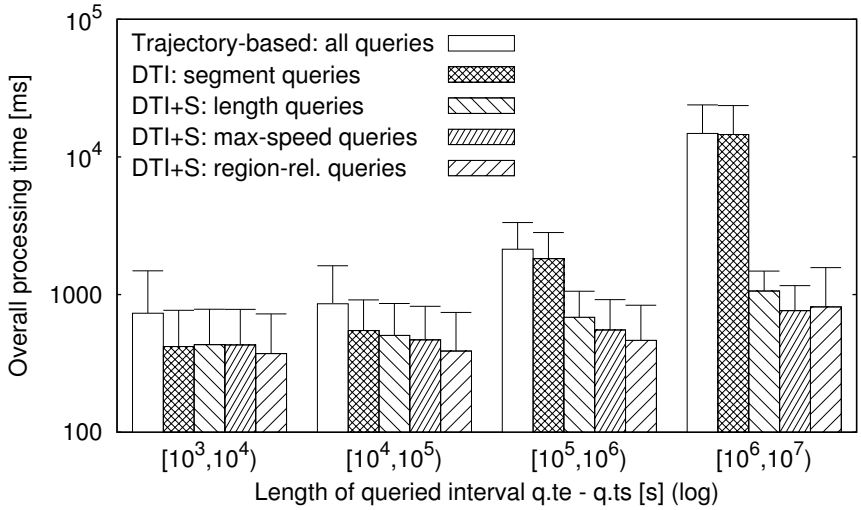


Figure 3.10: Processing time depending on queried time.

CPU times. The figure shows only one column for query processing without DTI+S since thereby the network latencies and disk I/O times are identical for all query types and since the CPU times always are negligible compared to the other two values. In detail, without DTI+S, the average CPU time for query processing varies between 3 ms for a segment query and 25 ms for a

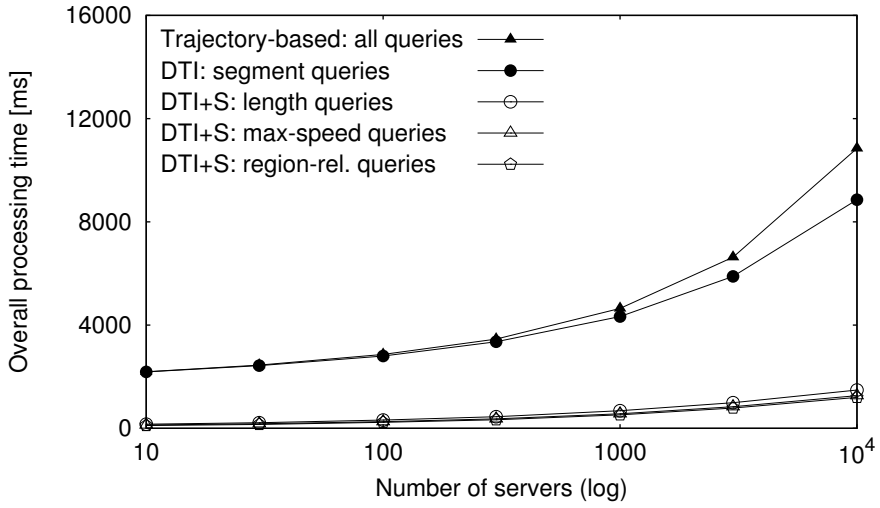


Figure 3.11: Processing time depending on number of servers.

max-speed query. With DTI+S, the CPU time is 3 ms for a segment query and about 1 ms for the other query types.

Without DTI+S and for segment queries, the disk I/O times of about 3400 ms clearly exceed the network latencies of about 1300 ms. With the use of summaries, i.e. for length, max-speed or region-relation queries, they range between 300 and 420 ms and 200 and 250 ms, respectively. Hence, the summaries reduce the network latencies by more than 75% and the disk I/O times by even more than 85%.

Figure 3.10 shows the overall processing time for different lengths of the queried time intervals $q.t_\mathrm{e} - q.t_\mathrm{s}$ and for different query types. Without DTI+S, it linearly increases with the length of the queried time interval since implicit shortcuts cannot be utilized in the third phase. For a query on weeks or months – 10^6 to 10^7 s – processing without DTI+S requires almost 15000 ms on average. With DTI, also segment queries show a linear increase of the processing time since segment queries are not improved by summaries.

For the other three types of queries, the DTI+S scheme saves large amounts of time:

- *Length queries:* Processing a length query with $10^6\,\mathrm{s} \leq q.t_\mathrm{e} - q.t_\mathrm{s} < 10^7\,\mathrm{s}$ requires 1059 ms on average.

- *Max-speed queries:* With $10^6\,\mathrm{s} \leq q.t_\mathrm{e} - q.t_\mathrm{s} < 10^7\,\mathrm{s}$, they only require 764 ms on average as with these queries more summaries are usable than with length queries. Hence, the DTI+S scheme can reduce the overall processing time by more than 95%.

- *Region-relation queries:* For small queried time intervals, DTI+S-based processing requires slightly less time than for the other types of queries since with a region-relation query the second phase can end prematurely if a server $s_i \in S(q.o) \setminus S(q)$ with a usable summary is encountered. With longer time intervals, processing of region-relation queries requires more time than processing of length or max-speed queries since the MBRs of summaries on very long time intervals mostly are too large to be usable.

Figure 3.11 depicts the overall processing time depending on the number of servers. It shows that the mentioned savings even further increase with the number of servers.

The DTI+S scheme achieves these savings with negligible storage consumption compared to the amounts of trajectory data stored by the space-partitioned MOD. With the mentioned $8.7 \cdot 10^5$ vertexes, each trajectory consumes about $8.7 \cdot 10^5 \times 24\,\mathrm{byte} \approx 19.9\,\mathrm{MB}$. DTI+S, on the contrary, consumes only $2080 \times 225\,\mathrm{byte} \approx 0.45\,\mathrm{MB}$ per trajectory. Thus, DTI+S accounts for less than 2.3% of the overall storage consumption.

3.7 Related Work

Our research on space-partitioned MODs is related to MODs in general and scalable location services. The former are mainly considered in database research whereas the latter originate from mobile computing.

In the last decade numerous index structures for efficient access to past trajectory data in MODs have been proposed [PJT00, MGA03, GS05, HKTG06, PSJ06, NR07]. However, none of these index structures addresses distributed systems. Therefore they consider neither query routing, nor summaries of segments stored by multiple servers to reduce routing and processing times. DTI+S is complementary to these approaches. It optimizes the distributed

processing including query routing, while these approaches optimize local query processing.

Location services such as location registers for mobile communications networks [PS01], the Globe Location Service [vSHHT98], the large-scale location service presented in [LR02], and GeoGrid [ZZL07] enable scalable management of moving objects' current positions. Similar to space-partitioned MODs, they use spatial partitioning for distributing the position data among sets of servers for scalability reasons. However, these systems do not store past trajectory data.

PLACE* is a distributed data stream management system, which processes continuous range and k-nearest-neighbor queries on a set of moving objects [XECA07]. For scalability, its service area is partitioned to regional servers similar to a space-partitioned. However, it does not process queries on past position data.

In [TDSC07], Trajcevski et al. present the Bresenham-based Overlay for Routing and Aggregation (BORA) for efficient processing of coordinate-based queries – in particular of spatiotemporal range queries – in space-partitioned MODs. Given a range query, a tree-like routing structure is created from the queried service regions to the destination server where the query result is needed. The partial results are routed and aggregated along this structure until the complete result reaches the destination server.

BORA and DTI+S thus complement one another. Together they allow for efficient processing of coordinate- *and* trajectory-based queries.

3.8 Summary

In this chapter, we presented the *Distributed Trajectory Index* (DTI) for efficient routing of trajectory-based queries in space-partitioned MODs. A DTI realizes an overlay network between servers storing segments of a certain trajectory. Therefore, it enables to access trajectory information about a given moving object efficiently – but can likewise be used to access any other (time-dependent) information associated with this trajectory.

Our evaluation shows that the DTI scheme significantly reduces the time for routing compared to plain trajectory-based routing. For instance, for a MOD

with 1000 servers, DTI reduces the routing time by up to 69%.

Furthermore, we proposed DTI+S, which enhances DTI by maintaining summaries on trajectory segments. The summaries enable efficient processing of queries on aggregates of dynamic attributes such as the length of a segment, the maximum speed during a time interval, or the intersection of a segment and a given region. Our simulations show that DTI+S can reduce the overall processing time by more than 95%.

4 Describing and Indexing of Context Models

In this chapter, we address the first two subproblems of how to provide efficient access to distributed dynamic context information, namely the formal describing of context models and the indexing of such descriptions, to allow for efficient discovery of relevant context models out of potentially millions of descriptions.

We propose an extended logic-based formalism for describing context models, which can be applied to any heterogeneous information system (HIS) based on a shared ontology. Next, we propose the *Source Description Class Tree* (SDC-Tree) for indexing such descriptions, featuring multidimensional indexing capabilities. We also propose a generic algorithm for splitting nodes of the tree, which can be used with arbitrary (context) ontologies. We show the effectiveness of the SDC-Tree and the split algorithm in simulations with descriptions of potential context models. Finally, we discuss related work and give a brief summary of the chapter.

4.1 Preliminaries

Regarding the discovery of context models, a distributed global-scale context management system can be considered as a heterogeneous information systems (HIS) consisting of a large number of information sources. The sources provide information on different aspects and clippings of the overall domain of discourse, i.e. the physical world – possibly using different local schemas.

Existing approaches for discovering the sources of a HIS that are relevant for a certain application task or query utilize the schema mappings between the sources or to some shared schema to exclude those sources from processing that do not provide *any* information about the queried relations or classes (e.g. [BCV99, AKK+03, KAP07]). This approach, however, does not scale to a global context management system and other large-scale HIS, where each

context model or source provides a comparatively small clipping of information for a certain class or relation, and applications likewise are interested in information about few entities of a certain class or relation only.

For example, there may exist thousands of providers of context models with building plans, i.e. information on rooms, corridors, stairways and other building parts, from different buildings all over the world. This information may be used for a number of context-aware applications such as indoor navigation, augmented visualization, and museum guidance. However, such an application only requires one or few plans of certain buildings at a time.

Therefore, as explained in Section 1.2, an expressive formalism for describing the context models or other sources in a concise manner and for matching them against compatible queries is needed – as well as a suitable tree structure for indexing such model or source descriptions. Since the designing of an index structure highly depends on the description formalism and its matching semantics, both subproblems are tackled together in this chapter.

Different description formalisms are imaginable, ranging from simple keyword lists, which are particularly used for text sources, to logic-based approaches for structured sources using constraints, e.g. to exclude a source about expensive hotels (price $\geq$ \$200) from a query for hostels (price $\leq$ \$30).

At a first glance, those two extremes seem to imply different matching semantics: Keywords imply *positive* semantics in view of the fact that a description matches a given query only if both share one or more explicitly stated keywords. Thus, a query issuer has to know the "right" keywords to discover a certain context model or source. Logic-based formalisms for source discovery, on the contrary, imply *negative* semantics in view of the fact that a description matches a query unless the converse can be proven by means of correspondent constraints.

In the following, we propose an extended logic-based description formalism that enables positive *and* negative matching semantics [LDR10b]. The formalism bases on *defined classes* and refines the idea of describing the sources by constraints in several ways: It allows for conjunctive and generally nested constraints over not only all attributes but also all relations of the classes of the shared ontology. The formalism enables to distinguish between different perspectives on the contents of a context model or source and thus alternative

descriptions. For example, it allows expressing that a 3D building model of the British Museum can be discovered either just by the location of the museum *or* just by the famous and unique name, whereas a less famous museum (with ambiguous name) may have to be discovered by its name *and* the coarse location. Finally, the formalism allows adjusting between positive and negative matching semantics for each query separately, which we suppose to be important for effective source discovery in context management and large-scale HIS.

That followed, we propose the *Source Description Class Tree* (SDC-Tree) for indexing source descriptions by means of their defined classes [LDR10b]. To enable efficient discovery, the SDC-Tree features multidimensional indexing capabilities for the different attributes and the IS-A hierarchy of the shared ontology, but also incorporates the existence or absence of constraints and the ranges of the constraints. For this purpose, it supports three different types of node split operations, exploiting the expressiveness of the description formalism.

Due to its flexibility regarding node splitting, the SDC-Tree can be easily adapted to a wide range of ontologies and source descriptions. Therefore, we further present a generic split algorithm suitable for arbitrary ontologies.

4.2 Assumptions and Notation

For generality, we do not confine ourselves to context management but consider any HIS where the sources share some ontology, providing a conceptualization of the overall domain of discourse. Such an ontology may be handcrafted for the purpose of source discovery only, or may be automatically generated from some shared (object-relational) schema. Our examples, however, are inspired by large-scale context management – even though we refer to the context providers and their models as sources in the following.

For the purpose of source descriptions and discovery, we consider three ontology components only:

1. *Classes* such as **ExhibitionHall**, **Building**, and **BuildingPart** organized in an IS-A hierarchy, i.e. where each class C has a unique *parent class* $\mathcal{P}rnt(C)$,

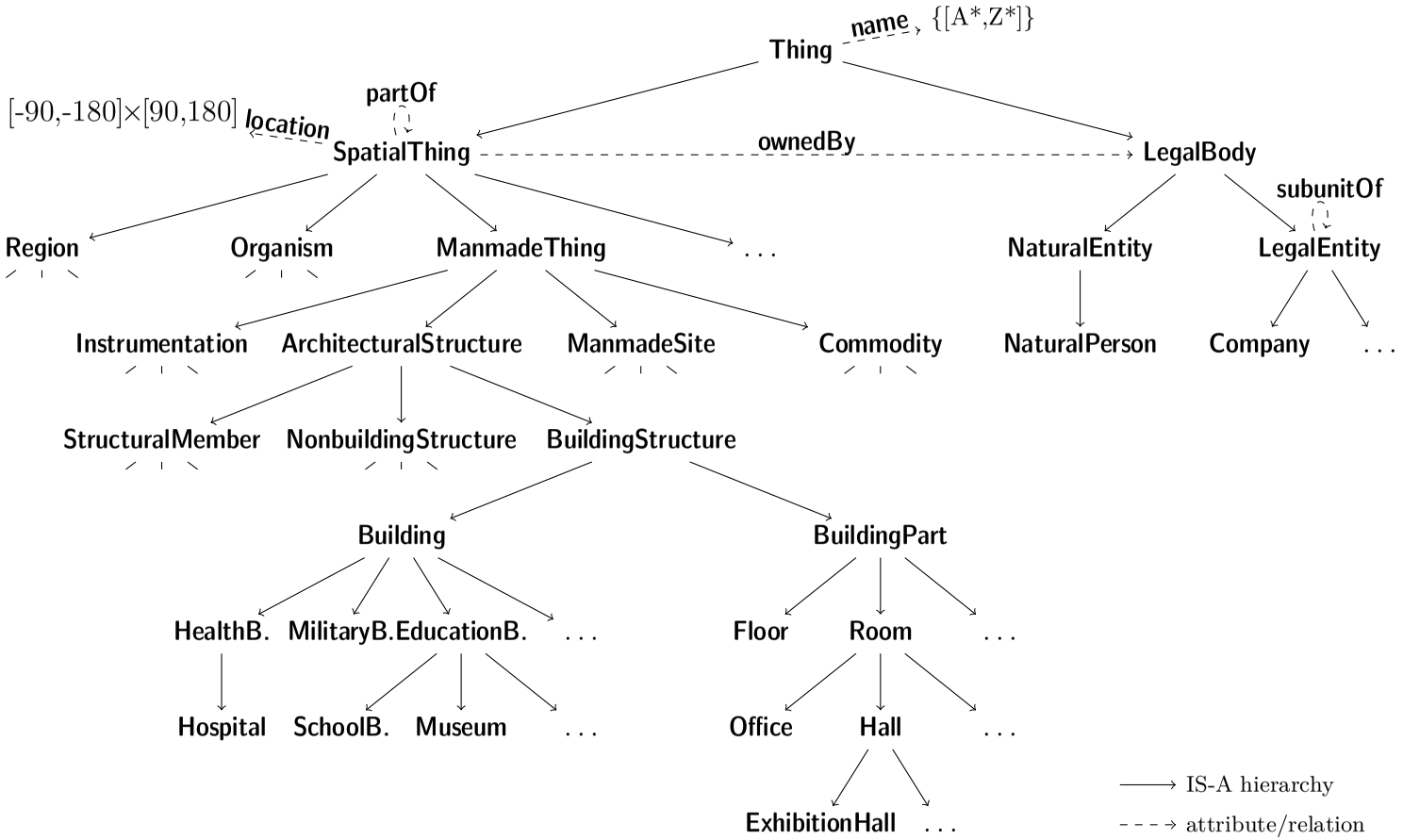


Figure 4.1: Context ontology created on the basis of [ADL, SUMO, Proton].

except for the top class $C_\top$. The expression $C \prec C'$ states that C is a subclass of C'.

2. *Attributes* such as **name** and **location** with primitive value ranges such as the set of all strings, the integers from -50 to 70, or the points (latitude and longitude) on the sphere. We denote the range of an attribute a by $\mathcal{Rng}(a)$ and the *domain* of a, i.e. the class C and subclasses of C it belongs to, by $\mathcal{Dom}(a) = C$.

   Ranges can be represented efficiently by intervals, sets of intervals, polygons, etc. depending on the data type (e.g. $\mathcal{Rng}(a) = \{[\text{-}50{,}70]\}$). In general, an object can have multiple values for a certain attribute, which may be specified by such expressions as well.

3. *Relations* such as **partOf** and **ownedBy** between entities of certain classes. Similar to attributes, we refer to the *domain* of a relation r as $\mathcal{Dom}(r) = C$ and to the *range* of r as $\mathcal{Rng}(r) = C'$.

Note that the ontology particularly enables *concrete domains*, i.e. attribute value ranges with (partial) orders and relational operators like $\leq$ and $\geq$, in addition to $=$ and $\neq$.

We do not make any assumptions on how applications interact with the HIS since all the different interaction patterns (e.g. query/response, data streams, events) require a mechanism to discover or decide on the relevant sources in advance.

Our examples throughout the chapter are inspired by a distributed context management system using the spatial context ontology given in Figure 4.1. The range [-90,-180]×[90,180] of the attribute **location** denotes the latitudes and longitudes of the world. This ontology has been created on the basis of the Alexandria Digital Library Feature Type Thesaurus [ADL], the Suggested Upper Merged Ontology (SUMO) [SUMO], and the PROTON Ontology (PROTo ONtology) [Proton].

It has been also used for the evaluation of the SDC-Tree described in Section 4.5.

4.3 Description Formalism

In this section, we analyze and explain how to describe sources by defined classes. Based on the definition of source descriptions, we finally discuss how to formulate compatible queries and how to match source descriptions against such queries.

4.3.1 Describing Sources by Defined Classes

Sources generally provide information about one or few coherent clippings of the domain of discourse. Thus, a source represents one or more sets of entities sharing some characteristic properties that can be described concisely by the class and one or more constraints. For example, the entities represented by a museums database for London simply share the properties that they all belong to the class museum and are located in London, and thus can be described by

$$\langle \textsf{Museum} : \textsf{location} \in \{\text{London, UK}\} \rangle \ ,$$

where we assume that the range {London, UK} is given as polygon on the globe.[1]

Such a description is a complete formal specification of a subclass of **Museum** and therefore is referred to as *defined class* (or *complex concept*) [BCM+03].

The characteristic properties may be also relations to one or few entities, which again can be described by defined classes. For example, the entities represented by a 3D building plan of the British Museum share the property that they are all part of the British Museum. Therefore, they can be described by

$$D_1 = \langle \textsf{BuildingPart} : \textsf{partOf} \in \langle \textsf{Museum} : \textsf{name} \in \{\text{“British Museum”}\} \rangle \rangle \ ,$$

using the famous and unique name of the museum.

Therefore, we propose to describe a source by one or more defined classes of the following form.

[1] There exist hybrid location models that allow determining the geographic area (polygon of latitude-longitude pairs) for a given location name and vice-versa [BD05].

Definition 2 (Defined class): A defined class D is specified by a class C followed by a conjunction of constraints on zero or more attributes and relations whose domains comprise C:

$$D = \left\langle C : a_1 \in X_1 \wedge a_2 \in X_2 \wedge \ldots \wedge r_1 \in \bar{D}_1 \wedge r_2 \in \bar{D}_2 \wedge \ldots \right\rangle$$

We refer to the class C as *base* of the defined class and introduce the operator $\mathcal{Base}(D)$ to retrieve C. The base specifies that the source provides information about entities of the class C or of a subclass of C.

A constraint $a_i \in X_i$, with $X_i \subseteq \mathcal{Rng}(a_i)$, states that the entities represented by the source have at least one value $x_i \in X_i$ for the attribute a_i. A range X_i can be given by a set- or interval-based expression depending on the data type. To retrieve X_i we introduce the operator $\mathcal{Con}_{a_i}(D)$. To test whether D actually has a constraint on attribute a, we introduce the operator $\textsc{isCon}_a(D)$ returning True or False, respectively.

Analogously, an expression $r_j \in \bar{D}_j$ specifies a constraint on the relation r_j to the range $\bar{D}_j$, where $\bar{D}_j$ again is a defined class of the above form with $\mathcal{Base}(\bar{D}_j) \preceq \mathcal{Rng}(r_j)$. We refer to $\bar{D}_j$ as *nested* defined class of D. The constraint states that the entities represented by the source have a relation r_j to an entity belonging to $\bar{D}_j$. The operator $\mathcal{Con}_{r_j}(D)$ returns the nested defined class $\bar{D}_j$ and the operator $\textsc{isCon}_r(D)$ queries whether D has a constraint on the relation r or not.

As mentioned above, a source may represent multiple coherent clippings of the domain of discourse. These can be described independently of each other. Without loss of generality, we therefore consider only one clipping per source in the following. Nevertheless, even a single clipping may be described by multiple, alternative defined classes. For example, the building plan of the British museum can be also described by

$$D_2 = \langle \textsf{BuildingPart} : \textsf{location} \in \{\text{44 Gt Russell St, London, UK}\} \rangle \ .$$

Of course, the constraints of D_1 and D_2 could be merged into a single defined class. However, this would hide the fact that both, the location and the unique name of the British Museum, each suffice as unambiguous descriptions for the

museum and the corresponding building plan. As discussed in the next section, to realize source discovery with positive matching semantics, such different *perspectives* on a source should be reflected by separate defined classes as exemplified by $\{D_1, D_2\}$.

Nevertheless, a description may consist of multiple, conjunctive constraints to describe the contents of a source distinctively. For example, this applies to museums with ambiguous names such as "Local Heritage Museum". A source about a certain museum with that name has to be described either by the exact location or by the name and the coarse location together.

4.3.2 Matching Queries against Source Descriptions

Analogously to source descriptions, we propose to specify discovery queries by defined classes. However, in contrast to a source description, a query should consist of one defined class Q (*query class*) only, as argued below.

Intuitively, a given source description with *source classes* $\{D_1, \ldots, D_n\}$ should *match* a query class Q if Q refers to one or more entities of the domain of discourse described by $\{D_1, \ldots, D_n\}$. However, since the individual entities are not known for source discovery, different matching semantics are imaginable. As indicated in Section 4.1, we can distinguish between two extremes. Considering defined classes, these extremes can be specified as follows:

- *Positive* matching: The source description $\{D_1, \ldots, D_n\}$ matches the query Q only if there exists such a D_i such that D_i and Q have constraints on the same attributes and relations and that the ranges of correspondent constraints overlap.

  Thus, the query issuer has to "know" the combinations of attributes and relations that are reasonably used in source descriptions.

- *Negative* matching: The source description $\{D_1, \ldots, D_n\}$ matches the query Q, unless the converse can be proven by means of the ranges of two correspondent constraints. More precisely, the source description *dismatches* Q only if there exists a D_i such that D_i and Q have disjoint constraint ranges for a certain attribute or relation.

On the one hand, we suppose that negative matching generally returns too many irrelevant sources in large-scale HIS with thousands of sources. On the other hand, the positive matching semantics are very strict since they require the query issuer to know or presume the combinations of attribute and relations being (typically) used in source classes. Therefore, we propose a flexible approach between both extremes.

Our matching approach is based on two predicates: The *query matching predicate* $\rightsquigarrow_{\mathrm{Q}}$ reflects the positive matching semantics, while the *query dismatching predicate* $/\!\!/_{\mathrm{Q}}$ reflects the negative extreme. The predicates $\rightsquigarrow_{\mathrm{Q}}$ and $/\!\!/_{\mathrm{Q}}$ together yield three possible states between a pair D_i and Q. Based on these states it then is decided whether a source description $\{D_1, \ldots, D_n\}$ matches Q or not.

Definition 3 (Query matching predicate $\rightsquigarrow_{\mathrm{Q}}$): Given a source class D_i and a query class Q, it holds $D \rightsquigarrow_{\mathrm{Q}} Q$ iff the following three conditions hold:

1. $(\mathcal{B}ase(D_i) \succeq \mathcal{B}ase(Q)) \vee (\mathcal{B}ase(D_i) \preceq \mathcal{B}ase(Q))$
2. $\forall$ attribute a with $(\mathcal{D}om(a) \succeq \mathcal{B}ase(Q)) \wedge (\mathcal{D}om(a) \succeq \mathcal{B}ase(D_i))$:
   $\textsc{isCon}_a(D_i) \Rightarrow (\textsc{isCon}_a(Q) \wedge (\mathcal{C}on_a(D_i) \cap \mathcal{C}on_a(Q) \neq \emptyset))$
3. $\forall$ relation r with $(\mathcal{D}om(r) \succeq \mathcal{B}ase(Q)) \wedge (\mathcal{D}om(r) \succeq \mathcal{B}ase(D_i))$:
   $\textsc{isCon}_r(D_i) \Rightarrow (\textsc{isCon}_r(Q) \wedge (\mathcal{C}on_r(D_i) \rightsquigarrow_{\mathrm{Q}} \mathcal{C}on_r(Q)))$

Thus, D_i and Q have to base on classes that are equal or sub-/superclass of each other, and Q has to specify correspondent constraints with overlapping ranges for all constraints specified in D_i.

Note that Q may specify constraints on more attributes and relations than D_i. Therefore $\rightsquigarrow_{\mathrm{Q}}$ is not commutative. This alleviates the strict positive semantics and is the reason why a source should be described by multiple alternative defined classes – representing different perspectives as explained in Section 4.3.1 – with as few constraints as possible, whereas a query should consist of only one defined class, containing all known constraints.

Definition 4 (Query dismatching predicate $/\!\!/_{\mathrm{Q}}$): Given a source class D_i and a query class Q, it holds $D /\!\!/_{\mathrm{Q}} Q$ iff *one* of the following two conditions

holds:

1. $\exists$ attribute a with $(\mathcal{D}om(a) \succeq \mathcal{B}ase(Q)) \wedge (\mathcal{D}om(a) \succeq \mathcal{B}ase(D))$:
   $\textsc{isCon}_a(D) \wedge \textsc{isCon}_a(Q) \wedge (\mathcal{C}on_a(D) \cap \mathcal{C}on_a(Q) = \emptyset)$
2. $\exists$ relation r with $(\mathcal{D}om(r) \succeq \mathcal{B}ase(Q)) \wedge (\mathcal{D}om(r) \succeq \mathcal{B}ase(D))$:
   $\textsc{isCon}_r(D) \wedge \textsc{isCon}_r(Q) \wedge (\mathcal{C}on_r(D) \mathbin{/\!\!/_{\mathrm{Q}}} \mathcal{C}on_r(Q))$

This predicate directly implements the negative matching semantics. Furthermore, it is contrary to $\rightsquigarrow_{\mathrm{Q}}$. Thus, $D_i \rightsquigarrow_{\mathrm{Q}} Q$ implies that D_i and Q do not dismatch.

Definition 5 (Matching): Based on these predicates, we define that a source description $\{D_1, \ldots, D_n\}$ matches a query Q only if there exists a D_i with $D_i \rightsquigarrow_{\mathrm{Q}} Q$ but no D_j with $D_j \mathbin{/\!\!/_{\mathrm{Q}}} Q$.

Thus, $\rightsquigarrow_{\mathrm{Q}}$ is a necessary condition for matching, while $\mathbin{/\!\!/_{\mathrm{Q}}}$ is a sufficient condition for a dismatch.

For example, consider the source description $\{D_1, D_2\}$ given in Section 4.3.1, and the query

$$Q_1 = \langle \textsf{ExhibitionHall} : \textsf{location} \in \{\text{32-50 Gt Russell St, London, UK}\} \rangle \ .$$

It queries for sources providing information about exhibition halls – which are a subclass of **BuildingPart** – located between 32 and 50 Great Russell Street. The source description matches Q_1 as $D_2 \rightsquigarrow_{\mathrm{Q}} Q_1$. However, it does not match

$$Q_2 = \langle \textsf{ExhibitionHall} : \textsf{partOf} \in \langle \textsf{Museum} : \textsf{name} \in \{[\text{E*,F*}]\} \rangle \rangle$$

as neither D_1 nor D_2 match Q_2 according to $\rightsquigarrow_{\mathrm{Q}}$. Also, the source description does not match

$$Q_3 = \langle \textsf{ExhibitionHall} : \textsf{partOf} \in \langle \textsf{Museum} : \textsf{name} \in \{[\text{E*,F*}]\} \rangle \wedge \textsf{location} \in \{\text{London, UK}\} \rangle$$

since $D_1 \mathbin{/\!\!/_{\mathrm{Q}}} Q_3$, even though $D_2 \rightsquigarrow_{\mathrm{Q}} Q_3$.

Negative matching semantics can be realized (completely or partially) by

extending Q with *pseudo constraints* of the form $a \in *$ or $r \in *$. For an attribute a, $*$ simply evaluates to $\mathcal{Rng}(a)$. In case of a relation r, $*$ has to be applied recursively to all attributes and relations whose domains comprise $\mathcal{Rng}(r)$. However, for evaluating $\rightsquigarrow_{\mathrm{Q}}$ or $/\!\!/_{\mathrm{Q}}$ it suffices to know that $*$ overlaps with any nested defined class $\mathcal{Con}_r(D_i)$.

The more pseudo constraints are specified in a query Q, the more likely there exists a $D_i \in \{D_1, \ldots, D_n\}$ where $D_i \rightsquigarrow_{\mathrm{Q}} Q$. Thus, the influence of the predicate $\rightsquigarrow_{\mathrm{Q}}$ decreases, whereas the influence of $/\!\!/_{\mathrm{Q}}$ increases.

Based on $\rightsquigarrow_{\mathrm{Q}}$ we can directly define a subsumption predicate $\succeq_{\mathrm{Q}}$, which allows determining whether a certain defined class not only overlaps but *completely comprises* another defined class. This predicate is also essential for the SDC-Tree.

Definition 6 (Query subsumption predicate $\succeq_{\mathrm{Q}}$): A defined class D subsumes another defined class Q, denoted by $D \succeq_{\mathrm{Q}} Q$, iff the following three conditions hold:

1. $\mathcal{Base}(D) \succeq \mathcal{Base}(Q)$
2. $\forall$ attribute a with $\mathcal{Dom}(a) \succeq \mathcal{Base}(D)$:
   $\textsc{isCon}_a(D) \Rightarrow (\textsc{isCon}_a(Q) \wedge (\mathcal{Con}_a(D) \supseteq \mathcal{Con}_a(Q)))$
3. $\forall$ relation r with $\mathcal{Dom}(r) \succeq \mathcal{Base}(D)$:
   $\textsc{isCon}_r(D) \Rightarrow (\textsc{isCon}_r(Q) \wedge (\mathcal{Con}_r(D) \succeq_{\mathrm{Q}} \mathcal{Con}_r(Q)))$

4.4 Source Description Class Tree

In this section, we present the *Source Description Class Tree* (SDC-Tree), a powerful tree structure for indexing source descriptions by means of their defined classes and for efficiently retrieving those descriptions that match a given query Q.

Given a source description, the SDC-Tree uses each source class as index key and stores a corresponding entry, mapping the source class to the source description, at one or more leaf nodes. The SDC-Tree considers the necessary matching condition only given by the query matching predicate $\rightsquigarrow_{\mathrm{Q}}$. Source descriptions with a source class $D_i \rightsquigarrow_{\mathrm{Q}} Q$ that do not match because of an

additional $D_j \mathbin{/\!\!/_{\mathrm{Q}}} Q$ are filtered in a post-processing step.

The SDC-Tree supports different operations for splitting a leaf node (*split types*), to completely exploit the expressiveness of the description formalism. Next, we describe the tree's structure and the split types before we give the corresponding formal definitions. Finally, an algorithm for selecting and parameterizing the split types is presented.

4.4.1 Structure of SDC-Tree

Each node of the SDC-Tree is associated with a defined class named *node class* whose form slightly extends the above definition of defined classes, as explained below.

The tree reflects the subsumption hierarchy of the node classes. Thus, let N be the node class of an inner node and N' the node class of one of its child nodes, it holds $N \succeq_{\mathrm{Q}} N'$. The node class of the root node is $\langle C_\top, \textsc{True} : \rangle$, where $C_\top$ denotes the top class of the ontology. The role of True is explained below. The root node class matches every defined class.

A source class D to be indexed is passed down iteratively from the root to each child node whose node class N' matches D by the *index matching predicate* $\rightsquigarrow_{\mathrm{I}}$, defined below. This predicate is a special case of the query matching predicate $\rightsquigarrow_{\mathrm{Q}}$ such that $N' \rightsquigarrow_{\mathrm{I}} D$ implies $N' \rightsquigarrow_{\mathrm{Q}} D$. Finally, an entry that maps from D to the source description it belongs to is stored at every leaf node with node class $N \rightsquigarrow_{\mathrm{I}} D$. For example, in Figure 4.2 the source class

$$D = \langle \textsf{BuildingPart} : \textsf{location} \in \{44 \text{ Gt Russell St, London, UK}\} \rangle$$

is passed from N_1, via N_3 and N_6, to N_8 as D's base **BuildingPart** is a subclass of **SpatialThing** and the constraint on **location** (polygon of latitude-longitude pairs) is completely contained in the Northern Hemisphere specified by the constraint range [0,-180]×[90,180] of N_8.

A query Q, on the contrary, is passed down using $\rightsquigarrow_{\mathrm{Q}}$. Thus, Q is distributed to every leaf node with node class $N \rightsquigarrow_{\mathrm{Q}} Q$. Next, each entry of those leaf nodes is evaluated against Q using $\rightsquigarrow_{\mathrm{Q}}$ and the source descriptions of those entries with source class $D \rightsquigarrow_{\mathrm{Q}} Q$ are collected and duplicates are removed.

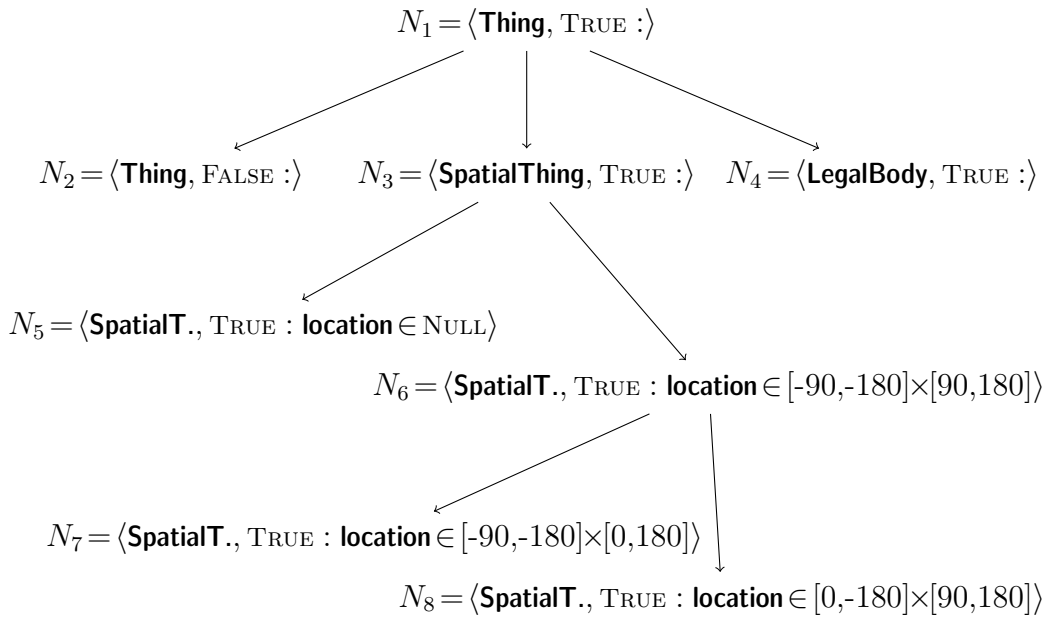


Figure 4.2: Examples of splits in the SDC-Tree.

Then, those source descriptions that do not match Q because of another source class $D \not\rightsquigarrow_{\mathrm{Q}} Q$ are removed. Finally, the remaining set of source descriptions is returned to the query issuer.

Since $\rightsquigarrow_{\mathrm{I}}$ is a special case of $\rightsquigarrow_{\mathrm{Q}}$, it generally holds $\{N : N \rightsquigarrow_{\mathrm{I}} D\} \subset \{N : N \rightsquigarrow_{\mathrm{Q}} D\}$. This property results from non-commutativity of $\rightsquigarrow_{\mathrm{Q}}$. For example, in Figure 4.2, the query class

$$Q = \langle \textbf{ExhibitionHall} : \textbf{location} \in \{\text{32-50 Gt Russell St, London, UK}\} \rangle ,$$

matching D from the above example, is passed from N_1 to the three leaf nodes N_2, N_5, and N_8 using $\rightsquigarrow_{\mathrm{Q}}$.

To reduce the number of source classes $\mathbb{D}$ indexed by a leaf node with node class N, N can be split into two or more new node classes $N'_1, N'_2, \ldots$.

We distinguish three split types, one for each aspect of how defined classes can differ from one another.

Base split. This type of split operation performs a split by means of the IS-A hierarchy of the shared ontology. Given a leaf node with node class N and source classes $\mathbb{D}$, it specifies $1 + l$ new child nodes with nodes classes N'_1 to N'_{1+l}, where $l = |\{C' : \mathcal{P}rnt(C') = \mathcal{B}ase(N)\}|$, which must be > 0.

N'_1 equals N except that it does not match source classes D with $\mathcal{B}ase(D) \prec \mathcal{B}ase(N)$ by $\rightsquigarrow_{\text{I}}$. For this purpose, a node class extends the form of defined classes by a Boolean parameter given after the base, as shown in Figure 4.2. This parameter indicates whether the node class matches source classes whose base is a proper subclass of its own base (TRUE) or not (FALSE). Hence, with N'_1 this parameter is set to FALSE and N'_1 cannot be further split by means of its base.

N'_2 to N'_{1+l} are equal to N, i.e. the mentioned parameter is TRUE, except that each of them has different new base C' with $\mathcal{P}rnt(C') = \mathcal{B}ase(N)$.

After a base split operation, each $D \in \mathbb{D}$ can be relocated unambiguously to one of the new nodes according to $\rightsquigarrow_{\text{I}}$. A query class Q with $N \rightsquigarrow_{\text{Q}} Q$, however, is matched by two or all new node classes.

A base split can be useful for two reasons: First, it may partition $\mathbb{D}$ to different nodes and thus allows pruning irrelevant source classes for a query class Q with $\mathcal{B}ase(Q) \prec \mathcal{B}ase(N)$. Second, it enables further node splits by means of attributes and relations that are specific to child classes of $\mathcal{B}ase(N)$.

Figure 4.2 exemplifies a base split at N_1: According to the sample ontology given in Section 4.2, the root class **Thing** has two subclasses **SpatialThing** and **LegalBody**. The base split therefore creates three node classes N_2, N_3, and N_4. Node class N_2 has the base **Thing** but does not match any subclasses according to $\rightsquigarrow_{\text{I}}$, whereas N_3 and N_4 have the bases **SpatialThing** and **LegalBody** and also match the respective subclasses.

Existence split. This type of split operation performs a split by specifying the existence or non-existence of an attribute constraint. Given a leaf node with source classes $\mathbb{D}$ and node class N without constraint on attribute a, it is useful to differentiate between source classes that have a constraint on a and those that have no constraint. Furthermore, existence splits are needed to enable range splits (see below) on the former set of source classes.

The split operation creates two new child nodes with node classes N'_1 and

N_2' as follows: N_1' equals N but additionally specifies a constraint $\mathcal{C}on_a(N_1') = \mathcal{R}ng(a)$. N_2' equals N but explicitly specifies that a source class D must *not* have a constraint on a in order to $N_2' \rightsquigarrow_{\text{I}} D$. For this purpose, node classes extend the definition of defined classes by the possibility to prevent a constraint on a indicated by $a \in \textsc{Null}$. Such a prevention is shown in Figure 4.2 at N_5, resulting from an existence split at N_3. Consequently, a source class D with $N \rightsquigarrow_{\text{I}} D$ matches either N_1' or N_2' only.

Since $\rightsquigarrow_{\text{Q}}$ does not consider the prevention of constraints, a query Q with $N \rightsquigarrow_{\text{Q}} Q$ is matched either by N_1' and N_2' or only by N_2'.

An existence split may be also performed by means of a relation r, where N_1' extends N by a $\mathcal{C}on_r(N_1') = \left\langle \mathcal{R}ng(r), \textsc{True} : \right\rangle$, accordingly.

Range split. Given a leaf node whose node class N has a constraint on attribute a, this operation performs a split by partitioning $\mathcal{C}on_a(N)$. This split type is especially useful if the source classes $\mathbb{D}$ indexed by the node significantly differ regarding $\mathcal{C}on_a(N)$.

A range split specifies two or more node class N_1', N_2', ... with ranges $\mathcal{C}on_a(N_i')$ such that $\mathcal{C}on_a(N_1') \cup \mathcal{C}on_a(N_2') \cup \ldots = \mathcal{C}on_a(N)$ and $\mathcal{C}on_a(N_i') \cap \mathcal{C}on_a(N_j') = \emptyset$, $\forall i \neq j$.

A source class D with $N \rightsquigarrow_{\text{I}} D$ whose constraint on a overlaps a certain $\mathcal{C}on_a(N_i')$ but no other $\mathcal{C}on_a(N_j')$ $(j \neq i)$ is indexed at the new child node with node class N_i' only. Otherwise, D is passed to multiple of the new nodes. Similarly, a query class may be passed to one or multiple of the new nodes.

Figure 4.2 gives an example of a range split with the node class N_6 by means of the constraint range [-90,-180]×[90,180] (the latitudes and longitudes of the whole world) into two ranges specifying the Southern and Northern Hemisphere, respectively.

The SDC-Tree does not pose any restrictions on how the ranges $\mathcal{C}on_a(N_1')$, $\mathcal{C}on_a(N_2')$, ... are determined. This also depends on the data type of a. The Generic Split Algorithm proposed in Section 4.4.4 uses the split algorithm of the R*-Tree [BKSS90] for this purpose.

Each of the above types of split operations can be also performed by means of a nested node class. For example, the node class

$$\langle \textsf{BuildingPart}, \textsc{True} : \textsf{partOf} \in \langle \textsf{Museum}, \textsc{True} : \textsf{name} \in \{[\text{A*}, \text{Z*}]\}\rangle\rangle$$

with a nested class for **partOf** may be split into

$$\langle \textsf{BuildingPart}, \textsc{True} : \textsf{partOf} \in \langle \textsf{Museum}, \textsc{True} : \textsf{name} \in \{[\text{A*}, \text{M*}]\}\rangle\rangle$$

and

$$\langle \textsf{BuildingPart}, \textsc{True} : \textsf{partOf} \in \langle \textsf{Museum}, \textsc{True} : \textsf{name} \in \{[\text{N*}, \text{Z*}]\}\rangle\rangle$$

using a range split by the constraint on **name**.

For every pair of source class D and query class Q, each split type guarantees that given an inner node with node class N there exists a child node with node class N' such that $(D \rightsquigarrow_{\text{Q}} Q) \wedge (N \rightsquigarrow_{\text{I}} D) \wedge (N \rightsquigarrow_{\text{Q}} Q)$ implies $(N' \rightsquigarrow_{\text{I}} D) \wedge (N' \rightsquigarrow_{\text{Q}} Q)$. This property is essential for the completeness of indexing of the SDC-Tree, i.e. that the SDC-Tree correctly determines all source classes D with $D \rightsquigarrow_{\text{Q}} Q$ for a given query Q. A proof of this property is given below in Section 4.4.3.

The SDC-Tree gives a lot of freedom regarding the splitting of nodes, despite the fact that base and existence splits generally are first required to enable range splits. Therefore, different strategies for the selection and parameterization of split operations are feasible.

For example, a thinkable approach is to first perform a series of base splits until no more base splits are possible, then to perform existence splits on all attributes and relations, and then to perform range splits where reasonable. However, in case that the source classes mainly differ in terms of nested classes on a certain relation, this approach would generate many unnecessary nodes.

Another possibility is to take the semantics and characteristics of the different classes, attributes, and relations into account. For example, in a distributed context management system the characteristic attribute **location** is likely to be used in many source classes and therefore an obvious candidate for splitting. This approach, however, requires a dedicated split algorithm for each ontology.

In consideration of this discussion, a split algorithm incorporating the actual

source descriptions is preferable to approaches accounting only for the ontology. Therefore, we propose a generic algorithm that is independent of the semantics of the shared ontology, but that makes its split decisions based on the indexed source classes. Before we render this algorithm more precisely, we next give formal definitions for a node class and the index matching predicate $\rightsquigarrow_{\mathrm{I}}$ and prove the completeness of indexing.

4.4.2 Formalism for Node Classes and Index Predicates

In the following, we give formal definitions for node classes and $\rightsquigarrow_{\mathrm{I}}$ and introduce an *index subsumption predicate* $\succeq_{\mathrm{I}}$.

Definition 7 (Node class): A node class N is a defined class according to the definition in Section 4.3.1 with an additional Boolean parameter b as follows

$$N = \left\langle C, b : a_1 \in X_1 \wedge a_2 \in X_2 \wedge \ldots \wedge r_1 \in \bar{N}_1 \wedge r_2 \in \bar{N}_2 \wedge \ldots \right\rangle \ .$$

The parameter b can be queried by the unary operator $\textsc{inclSub}(N) = b$ and denotes whether N may match other node classes or defined classes with base $C' \prec C$ by $\rightsquigarrow_{\mathrm{I}}$.

A range X_i may be Null to indicate that N prevents constraints on the corresponding attribute a_i. This property can be queried by a Boolean unary operator $\textsc{isPrev}_{a_i}(N)$. Hence, N may either

1. specify no constraint $(\neg\textsc{isCon}_{a_i}(N) \wedge \neg\textsc{isPrev}_{a_i}(N))$, or
2. specify a constraint $(\textsc{isCon}_{a_i}(N) \wedge \neg\textsc{isPrev}_{a_i}(N))$, or
3. prevent a constraint $(\neg\textsc{isCon}_{a_i}(N) \wedge \textsc{isPrev}_{a_i}(N))$.

The same applies to constraints on relations.

Definition 8 (Index matching predicate $\rightsquigarrow_{\mathrm{I}}$): A node class N matches a source class D, denoted by $N \rightsquigarrow_{\mathrm{I}} D$, iff the following three conditions hold:

1. $(\mathcal{Base}(N) = \mathcal{Base}(D)) \vee (\textsc{inclSub}(N) \wedge (\mathcal{Base}(N) \succ \mathcal{Base}(D)))$
2. $\forall$ attribute a with $\mathcal{Dom}(a) \succeq \mathcal{Base}(N)$:
   $\textsc{isCon}_a(N) \Rightarrow (\textsc{isCon}_a(D) \wedge (\mathcal{Con}_a(N) \cap \mathcal{Con}_a(D) \neq \emptyset))$,
   $\textsc{isPrev}_a(N) \Rightarrow \neg\textsc{isCon}_a(D)$
3. $\forall$ relation r with $\mathcal{Dom}(r) \succeq \mathcal{Base}(N)$:
   $\textsc{isCon}_r(N) \Rightarrow (\textsc{isCon}_r(D) \wedge (\mathcal{Con}_r(N) \rightsquigarrow_{\mathrm{I}} \mathcal{Con}_r(D)))$,
   $\textsc{isPrev}_r(N) \Rightarrow \neg\textsc{isCon}_r(D)$

Since each of the three conditions implies the respective condition of $\rightsquigarrow_{\mathrm{Q}}$, it follows that $N \rightsquigarrow_{\mathrm{I}} D$ implies $N \rightsquigarrow_{\mathrm{Q}} D$.

Analogous to the definition of $\rightsquigarrow_{\mathrm{I}}$, implying $\rightsquigarrow_{\mathrm{Q}}$, a transitive *index subsumption predicate* $\succeq_{\mathrm{I}}$ implying $\succeq_{\mathrm{Q}}$ can be defined – between pairs of node classes as well as pairs of node classes and source classes.

Definition 9 (Index subsumption predicate $\succeq_{\mathrm{I}}$): A node class N subsumes another node class N', denoted by $N \succeq_{\mathrm{I}} N'$, iff the following three conditions hold:

1. $(\textsc{inclSub}(N) \wedge (\mathcal{Base}(N) \succeq \mathcal{Base}(N'))) \vee$
   $(\neg\textsc{inclSub}(N) \wedge \neg\textsc{inclSub}(N') \wedge (\mathcal{Base}(N) = \mathcal{Base}(N')))$
2. $\forall$ attribute a with $\mathcal{Dom}(a) \succeq \mathcal{Base}(N)$:
   $\textsc{isCon}_a(N) \Rightarrow (\textsc{isCon}_a(N') \wedge (\mathcal{Con}_a(N) \supseteq \mathcal{Con}_a(N')))$,
   $\textsc{isPrev}_a(N) \Rightarrow \textsc{isPrev}_a(N')$
3. $\forall$ relation r with $\mathcal{Dom}(r) \succeq \mathcal{Base}(N)$:
   $\textsc{isCon}_r(N) \Rightarrow (\textsc{isCon}_r(N') \wedge (\mathcal{Con}_r(N) \succeq_{\mathrm{I}} \mathcal{Con}_r(N')))$,
   $\textsc{isPrev}_r(N) \Rightarrow \textsc{isPrev}_r(N')$

For the subsumption of a source class D, it holds: $N \succeq_{\mathrm{I}} D$ iff the following

three conditions hold:

1. $(\mathcal{Base}(N) = \mathcal{Base}(D)) \vee (\textsc{inclSub}(N) \wedge (\mathcal{Base}(N) \succ \mathcal{Base}(D)))$
2. $\forall$ attribute a with $\mathcal{Dom}(a) \succeq \mathcal{Base}(N)$:
   $\textsc{isCon}_a(N) \Rightarrow (\textsc{isCon}_a(D) \wedge (\mathcal{Con}_a(N) \supseteq \mathcal{Con}_a(D)))$,
   $\textsc{isPrev}_a(N) \Rightarrow \neg\textsc{isCon}_a(D)$
3. $\forall$ relation r with $\mathcal{Dom}(r) \succeq \mathcal{Base}(N)$:
   $\textsc{isCon}_r(N) \Rightarrow (\textsc{isCon}_r(D) \wedge (\mathcal{Con}_r(N) \succeq_{\text{I}} \mathcal{Con}_r(D)))$,
   $\textsc{isPrev}_r(N) \Rightarrow \neg\textsc{isCon}_r(D)$

It holds that the node classes of the SDC-Tree form a subsumption hierarchy with respect to $\succeq_{\text{I}}$ and thus $\succeq_{\text{Q}}$.

In summary, the query predicates $\rightsquigarrow_{\text{Q}}$, $/\!\!/_{\text{Q}}$, and $\succeq_{\text{Q}}$ ignore the two extensions of node classes by inclSub and isPrev, whereas the index predicates $\rightsquigarrow_{\text{I}}$ and $\succeq_{\text{I}}$ incorporate both extensions. The resulting implications between the five predicates can be depicted by a graph as follows:

$$\begin{array}{ccccc} \succeq_{\text{Q}} & \Rightarrow & \rightsquigarrow_{\text{Q}} & \Rightarrow & \text{not } /\!\!/_{\text{Q}} \\ \Uparrow & & \Uparrow & & \\ \succeq_{\text{I}} & \Rightarrow & \rightsquigarrow_{\text{I}} & & \end{array}$$

4.4.3 Proof for the Completeness of Indexing

To show that the SDC-Tree correctly determines all source classes $D \rightsquigarrow_{\text{Q}} Q$ for a query Q, we prove that for each pair D and Q there exists a leaf node with node class $N_k \rightsquigarrow_{\text{I}} D$ and $N_k \rightsquigarrow_{\text{Q}} Q$ and that this leaf node is found when matching Q by $\rightsquigarrow_{\text{Q}}$ against the node classes from the root downwards to the leaf nodes.

Theorem 2 (Completeness of indexing): For every pair of source class D and query class Q with $D \rightsquigarrow_{\text{Q}} Q$, there exists a sequence of nodes with node classes $N_1, \ldots, N_k$ from the root with node class $N_1 = \langle C_\top, \textsc{True} : \rangle$ to a leaf node with node class N_k such that

$$\forall N_i \in \{N_1, \ldots, N_k\} : (N_i \rightsquigarrow_{\text{I}} D) \wedge (N_i \rightsquigarrow_{\text{Q}} Q) \ .$$

Proof: Induction starting at the root node class N_1.

Base case: It is $N_1 = \langle C_\top, \textsc{True} : \rangle$. It holds, $\forall D : N_1 \rightsquigarrow_{\text{I}} D$ and $\forall Q : N_1 \rightsquigarrow_{\text{I}} Q$, independent of whether D matches Q by $\rightsquigarrow_{\text{Q}}$ or not.

Inductive step: For each split type, we show that given an inner node with node class N_i $(1 \leq i < k)$, there exists a child node with node class N_{i+1} such that $(D \rightsquigarrow_{\text{Q}} Q) \wedge (N_i \rightsquigarrow_{\text{I}} D) \wedge (N_i \rightsquigarrow_{\text{Q}} Q)$ implies $(N_{i+1} \rightsquigarrow_{\text{I}} D) \wedge (N_{i+1} \rightsquigarrow_{\text{Q}} Q)$. For readability, we do not consider splits by means of nested classes of N_i in the following, as the argumentations can be applied to them analogously.

1. *Base split:* It holds $\textsc{inclSub}(N_i) = \textsc{True}$ and $\mathcal{Base}(N_i) \succeq \mathcal{Base}(D)$ as $N_i \rightsquigarrow_{\text{I}} D$.

   If $\mathcal{Base}(N_i) = \mathcal{Base}(D)$, then N_i and the node class N_{i+1} of the first child only differ in the value of $\textsc{inclSub}$. Thus, it holds $N_{i+1} \rightsquigarrow_{\text{I}} D$ and $N_{i+1} \rightsquigarrow_{\text{Q}} Q$.

   Otherwise, if $\mathcal{Base}(N_i) \succ \mathcal{Base}(D)$, then there exists a child with node class N_{i+1} with $\mathcal{Prnt}(\mathcal{Base}(N_{i+1})) = \mathcal{Base}(N_i)$ and $\mathcal{Base}(N_{i+1}) \succeq \mathcal{Base}(D)$. It holds $N_{i+1} \rightsquigarrow_{\text{I}} D$ and also $N_{i+1} \rightsquigarrow_{\text{Q}} Q$ as $D \rightsquigarrow_{\text{Q}} Q$.

2. *Existence split:* Let be a the attribute that has been used for the existence split. Thus, it is $\textsc{isCon}_a(N_i) = \textsc{False}$ and $\textsc{isPrev}_a(N_i) = \textsc{False}$.

   If D has a constraint on a, then the node class N_{i+1} of the child node defining a constraint to $\mathcal{Rng}(a)$ will match D by $\rightsquigarrow_{\text{I}}$. As $D \rightsquigarrow_{\text{Q}} Q$ either requires Q to also have a constraint on a or $\mathcal{Base}(Q) \succ \mathcal{Dom}(a)$, it also holds $N_{i+1} \rightsquigarrow_{\text{Q}} Q$.

   If D does not have a constraint on a, then the node class N_{i+1} of the other child preventing constraints on a will match D by $\rightsquigarrow_{\text{I}}$. Clearly, it also holds $N_{i+1} \rightsquigarrow_{\text{Q}} Q$, as the query match predicate $\rightsquigarrow_{\text{Q}}$ ignores $\textsc{isPrev}_a(N_{i+1})$.

   The same argumentation applies to an existence split by means of a relation r.

3. *Range split:* Let a be the attribute that has been used for the range split. Next, we distinguish whether $\mathcal{Base}(Q) \succ \mathcal{Dom}(a)$ or $\mathcal{Base}(Q) \preceq \mathcal{Dom}(a)$.

In the former, $N_i \rightsquigarrow_{\mathrm{Q}} Q$ implies $N_{i+1} \rightsquigarrow_{\mathrm{Q}} Q$ as $\rightsquigarrow_{\mathrm{Q}}$ evaluates neither the value range $\mathcal{Con}_a(N_i)$ nor $\mathcal{Con}_a(N_{i+1})$ of the node class N_{i+1} of any child node.

In the latter, we again have to distinguish whether $\mathcal{Con}_a(D)$ and $\mathcal{Con}_a(Q)$ overlap within the range $\mathcal{Con}_a(N_i)$ or not.

If they overlap, i.e. $(\mathcal{Con}_a(D) \cap \mathcal{Con}_a(Q)) \cap \mathcal{Con}_a(N_i) \neq \emptyset$, then there exists at least one child with node class N_{i+1} where they overlap within $\mathcal{Con}_a(N_{i+1})$ since $\mathcal{Con}_a(N_i)$ is partitioned completely. Thus, $N_{i+1} \rightsquigarrow_{\mathrm{I}} D$ and $N_{i+1} \rightsquigarrow_{\mathrm{Q}} Q$.

The converse, i.e. that $\mathcal{Con}_a(D)$ and $\mathcal{Con}_a(Q)$ are disjoint within $\mathcal{Con}_a(N_i)$, cannot happen due to the inductive construction of $N_1, \ldots, N_i$: The first range split on a at node class N_f $(f < i)$ was performed on $\mathcal{Con}_a(N_f) = \mathcal{Rng}(a)$ definitely comprising $\mathcal{Con}_a(D) \cap \mathcal{Con}_a(Q)$. For this range split and any further range split on a, we always chose the child with node class N_j $(f < j < i)$ such that $\mathcal{Con}_a(N_j)$ also comprises $\mathcal{Con}_a(D) \cap \mathcal{Con}_a(Q)$, as just explained.

□

4.4.4 Generic Split Algorithm

For splitting nodes of the SDC-Tree, different strategies for selection and parameterization of split operations are imaginable. As argued in Section 4.4.1, a split algorithm incorporating the source descriptions is preferable to approaches that only consider the ontology.

In the following, we therefore propose the *Generic Split Algorithm* (GSAlg), which makes its split decisions based on the indexed source classes and which is independent of the semantics of the shared ontology. GSAlg splits a leaf node once the number of entries stored in the leaf reaches a certain *split size threshold* n_{split}.

For deciding on the child node classes, GSAlg proceeds as follows: First, it computes all possible splits. Then, it determines how the source classes are distributed to the different node classes of each split and rates the splits accordingly. Finally, GSAlg chooses the split with the best rating.

Computing all possible splits. For computing all possible splits of a leaf node with node class N storing the source classes $\mathbb{D}$, GSAlg uses a recursive function given in Figure 4.3. First, it computes the base split of N if possible (lines 2 to 4) before it computes all possible existence splits (lines 5 to 10). GSAlg then computes one possible range split, with two subranges X_1 and X_2, for each attribute constraint (lines 11 to 16). For this purpose, GSAlg uses different variants of the split algorithm of the R*-Tree [BKSS90], adapted to the different primitive data types. This algorithm takes the ranges of the constraints of the source classes $\mathbb{D}$ into account such that X_1 and X_2 are optimized to divide $\mathbb{D}$ into equal-sized halves as possible. Finally, a recursive call is performed for each nested node class $\bar{N} = \mathcal{Con}_r(N)$ (lines 17 to 25).

The number of possible splits is limited by the number of attributes and relations of the different classes of the shared ontology and therefore widely independent of the size of the SDC-Tree. Only for the very first splits, i.e. small tree sizes, there exist less possible splits as base splits are generally required to enable other types of splits.

Rating of the possible splits. The three types of splits highly differ regarding the node classes they create: A base split generally creates more than two node classes, while existence splits create exactly two node classes. With GSAlg, the latter also applies to range splits. Moreover, with existence and base splits each source class matches one of the new node classes only, whereas a source class may match multiple of the new node classes with a range split. Therefore, we propose three tailored rating functions – one for each split type – based on a uniform rating metric ranging from zero (bad) to one (good).

1. *Base split:* A base split is only useful if there exists a significant number of source classes $D \in \mathbb{D}$ with $\mathcal{Base}(D) \prec \mathcal{Base}(N)$. Otherwise, the base split neither allows for pruning large sets of source classes during query processing nor is the split required for subsequent existence and range splits. The rating function for the base split therefore is

$$\frac{|\{D \in \mathbb{D} : \mathcal{Base}(D) \prec \mathcal{Base}(N)\}|}{n_{\text{split}}} .$$

1: $\mathbb{S} \leftarrow \emptyset$ ▷ Set of possible splits, i.e. elements are sets of child node classes.
2: **if** INCLSUB(N) **and** $|\{C' : \mathcal{P}rnt(C') = \mathcal{B}ase(N)\}| > 0$ **then** ▷ Base split.
3: $\mathbb{S} \leftarrow \{\{N$ with INCLSUB(N) $\leftarrow$ FALSE, N with $\mathcal{B}ase(N) \leftarrow$ first $C', \ldots\}\}$
4: **end if**
5: **for all** attribute a where $\mathcal{D}om(a) \succeq \mathcal{B}ase(N)$ **do** ▷ Existence splits.
6: **if** $\neg$ISCON$_a(N)$ **and** $\neg$ISPREV$_a(N)$ **then**
7: $\mathbb{S} \leftarrow \mathbb{S} \cup \{\{N$ with $\mathcal{C}on_a(N) \leftarrow \mathcal{R}ng(a)$, N with ISPREV$_a(N) \leftarrow$ TRUE$\}\}$
8: **end if**
9: **end for**
10: $[\ldots]$ ▷ Same for each relation r where $\mathcal{D}om(r) \succeq \mathcal{B}ase(N)$
11: **for all** attribute a with $\mathcal{D}om(a) \succeq \mathcal{B}ase(N)$ **do** ▷ Range splits.
12: **if** ISCON$_a(N)$ **and** $|\mathcal{C}on_a(N)| > 1$ **then**
13: $(X_1, X_2) \leftarrow$ compute partitioning of $\mathcal{C}on_a(N)$
14: $\mathbb{S} \leftarrow \mathbb{S} \cup \{\{N$ with $\mathcal{C}on_a(N) \leftarrow X_1, N$ with $\mathcal{C}on_a(N) \leftarrow X_2\}\}$
15: **end if**
16: **end for**
17: **for all** relation r with $\mathcal{D}om(r) \succeq \mathcal{B}ase(N)$ **do** ▷ Splits by nested classes.
18: **if** ISCON$_r(N)$ **then**
19: $\bar{\mathbb{D}} \leftarrow \{\mathcal{C}on_r(D) : \forall D \in \mathbb{D}\}$ ▷ Get nested source classes.
20: $\bar{\mathbb{S}} \leftarrow$ recursive call with $\mathcal{C}on_r(N)$ and $\bar{\mathbb{D}}$
21: **for all** splits $\bar{\mathbb{N}} \in \bar{\mathbb{S}}$ **do** ▷ Nest split from recursive call.
22: $\mathbb{S} \leftarrow \mathbb{S} \cup \{N$ with $\mathcal{C}on_r(N) \leftarrow \bar{N} : \forall \bar{N} \in \bar{\mathbb{N}}\}$
23: **end for**
24: **end if**
25: **end for**
26: **return** $\mathbb{S}$

Figure 4.3: Algorithm for computing all possible splits of a leaf node with node class N and sources classes $\mathbb{D}$.

Thus, the less source classes $D \in \mathbb{D}$ have $\mathcal{B}ase(D) = \mathcal{B}ase(N)$, the better rated is the split.

2. *Existence split:* The utility of an existence split on attribute a increases with the number of source classes $D \in \mathbb{D}$ with ISCON$_a(D)$ = TRUE. If there exists a large number of such source classes, the existence split

particularly is required to enable further range splits on a. Yet, even if only about half of the number of source classes have constraints on a, the split is useful for queries that do not have constraints on a. Therefore, the rating function is

$$\min\left(1,\ \frac{2\,|\{D \in \mathbb{D} : \textsc{isCon}_a(D)|}{n_{\text{split}}}\right)\ .$$

The same applies to existence splits on relations.

3. *Range split:* A split by means of a constraint range $\mathcal{C}on_a(N)$ computed by GSAlg is particularly useful if it partitions $\mathbb{D}$ into roughly equal-sized halves. Also, the number of source classes that are matched by both new node classes N'_1 and N'_2 should be small. We refer to the former property as *distribution* and to the latter as *selectivity*. The selectivity is computed by the number of source classes matched by either N'_1 or N'_2 only:

$$\frac{|\{D \in \mathbb{D} : (N'_1 \rightsquigarrow_{\text{I}} D) \neq (N'_2 \rightsquigarrow_{\text{I}} D)\}|}{n_{\text{split}}}$$

The distribution is computed as the minimum of the numbers of source classes that are matched by N'_1 and N'_2, respectively:

$$\min\left(1,\ \frac{2\,|\{D \in \mathbb{D} : N'_1 \rightsquigarrow_{\text{I}} D\}|}{n_{\text{split}}}, \frac{2\,|\{D \in \mathbb{D} : N'_2 \rightsquigarrow_{\text{I}} D\}|}{n_{\text{split}}}\right)$$

The rating of the split is the product of both.

The ratings can be easily computed along with the possible splits in the above algorithm. With recursive calls, the ratings can be directly taken for the unnested splits generated in line 22 of Figure 4.3.

By the uniform rating, all three split types are considered to be of equal importance. In order to prefer individual split types to other ones, this property may be softened by attaching weights to the rating functions. In this way, for instance, it is possible to assert that the IS-A hierarchy is reflected at the top of the SDC-Tree, by preferring base splits. Our evaluation results in Section 4.5.2 yet indicate that such weights do not improve the search costs.

4.5 Evaluation

As proof of concept for the description formalism as well as to show the efficiency of the SDC-Tree and GSAlg, we implemented the SDC-Tree as a main-memory index and simulated it with descriptions of potential context models derived from the OpenStreetMap (OSM) database [OSM]. Next, we first describe the evaluation setup, followed by the results on search costs and tree sizes. Finally, we give results on insertion costs and node splitting.

4.5.1 Setup

The components of the shared ontology and the source descriptions could generally be expressed by a subset of OWL [OWL] and SPARQL [SPARQL]. For simplicity and direct support of 2D geometries, however, we developed an easily readable *Simple Ontology Language* (SOL) inspired by the notation for defined classes used in the previous sections, and implemented an appropriate ontology framework. For example, the defined class

$$\langle \textsf{BuildingPart} : \textsf{partOf} \in \langle \textsf{Museum} : \textsf{name} \in \{\text{“British Museum”}\}\rangle\rangle$$

is specified as

```
<BuildingPart : partOf IN <Museum : name IN {String:"British Museum"}>>
```

in SOL. 2D geometries simply are specified by Well-Known Text (WKT) [OpenGIS].

The ontology framework supports strings, integers and 2D geometries as primitive data types and provides common relational operators on their values (e.g. $<$ and $>$) and on sets of their values (e.g. $\subseteq$ and $\supseteq$). For managing 2D geometries, the framework uses the Java Topology Suite [JTS]. Regarding strings, the framework supports arbitrary intervals and sets of intervals on the lexicographical order of the strings. Hence, it not only allows specifying intervals of strings sharing a certain prefix but *any* finite or infinite interval such as [Museum, Museum] and [MA, MC), specifying the single string *Museum* and the set of all strings starting with *MA* or *MB*, respectively.

To simulate as realistic descriptions of context models as possible, we used the OSM database [OSM]:

1. Since the OSM database uses a flat tagging system only, we created a spatial ontology on the basis of three established top-level ontologies [ADL, SUMO, Proton] and mapped the most frequent tags to the corresponding classes, relations, and attributes.

2. We created nine templates for descriptions of potential context models, e.g. of models providing information about the building elements of a public building or about the transportation structures of a town. Each such description template consists of two or more defined class templates.

3. Using these templates and the OSM database extract for Europe, we generated $5 \cdot 10^5$ descriptions with more than $1.1 \cdot 10^6$ source classes.

We simulated the SDC-Tree with different numbers of descriptions (and thus source classes) as well as different values for the split size threshold n_{split}. For querying, we randomly selected 1000 of the source descriptions, merged the source classes of each description, and used the resulting 1000 defined classes as queries. In our simulations, these queries matched up to 100 descriptions. Each measurement was repeated 25 times with different subsets of the $5 \cdot 10^5$ descriptions.

To be able to assess the performance of GSAlg with *successive* insertion and splitting as described in Section 4.4.4, we also created the SDC-Tree by *bulk* insertion as follows: The simulated number of descriptions is inserted at once into a SDC-Tree consisting of a single leaf node. Then, the leaf node with the most entries is split repeatedly using GSAlg until the number of nodes in the tree is equal the number of nodes generated by successive insertion. For the selection and parameterization of the split operations again the algorithm for computing all possible splits and the rating functions of GSAlg are used, except that n_{split} is replaced by the actual number of entries in the leaf node.

4.5.2 Search Costs and Tree Sizes

First, we consider the search costs measured by the number of evaluations of the query matching predicate $\rightsquigarrow_{\text{Q}}$ per query and the tree size measured by the

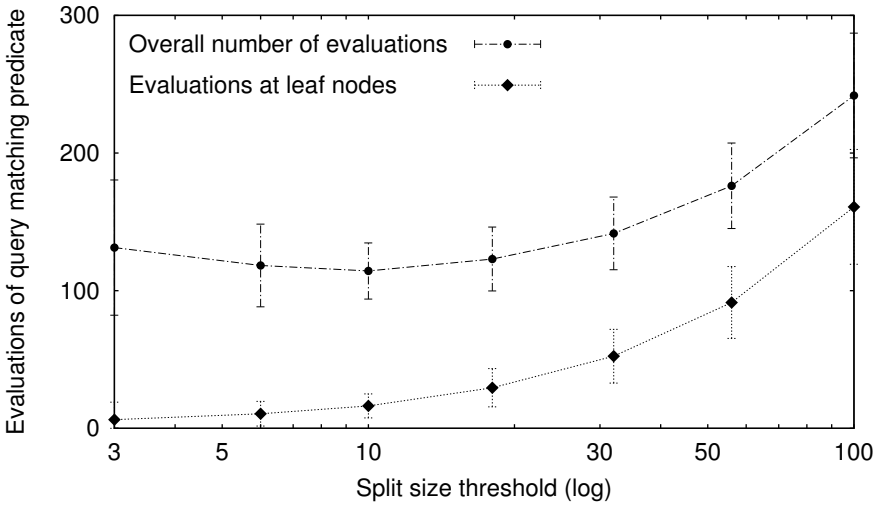


Figure 4.4: Search costs depending on split size threshold.

number of nodes.

Figure 4.4 gives the average search costs and standard deviations for a tree with 10^5 source classes depending on the split size threshold n_{split}. The upper line gives the overall number of $\rightsquigarrow_Q$ evaluations, whereas the lower line denotes the evaluations with source classes only. The latter increases monotonously with n_{split} due to the larger number of source classes per leaf. The overall number of evaluations, however, also increases with decreasing $n_{\text{split}} < 10$ due to the resulting large numbers of nodes. Therefore, we used $n_{\text{split}} = 10$ in the following.

Figure 4.5 shows that the search costs depend logarithmically on the number of index source classes and thus demonstrates the effectiveness of the SDC-Tree. For 1000 source classes, the SDC-Tree reduces the number of $\rightsquigarrow_Q$ evaluations to less than 10% compared to plain search over all descriptions. For 10^5 source classes, it even saves 99.9%. Besides, Figure 4.5 shows that the search costs with bulk insertion are only about 1% less than with successive insertion. Thus, successive insertion with GSAlg largely achieves the same indexing performance as bulk insertion, although the latter considers the whole (but fixed) set of source descriptions for the split decisions.

These results coincide with the linear dependency between the number of

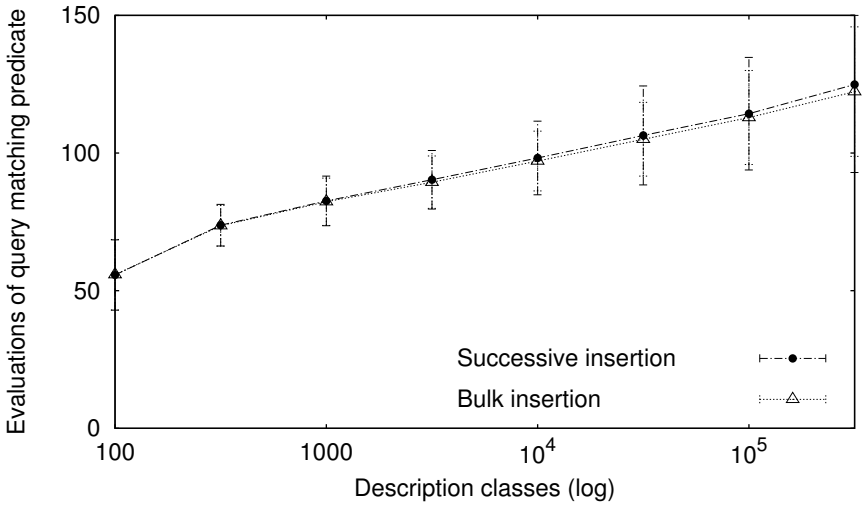


Figure 4.5: Search costs depending on source classes.

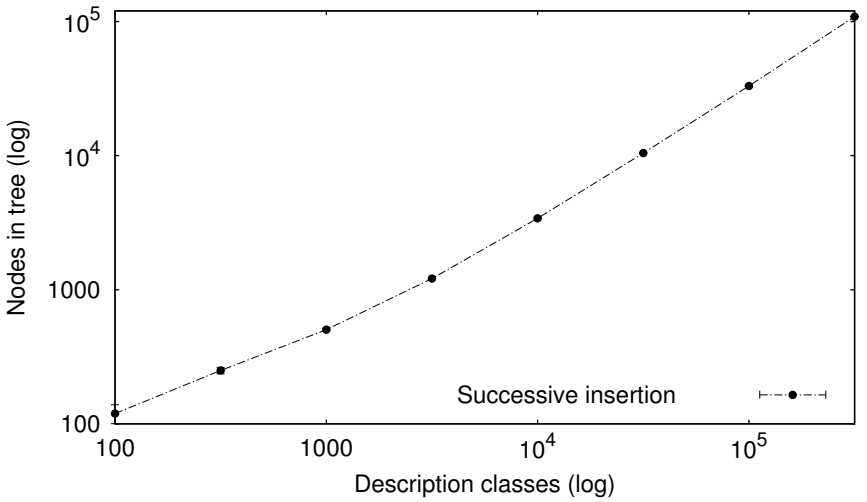


Figure 4.6: Tree size in nodes depending on source classes.

indexed source classes and the size of the SDC-Tree illustrated in Figure 4.6. Note the marginal standard deviations in these measurements.

Furthermore, to verify the choice of the uniform rating metric in GSAlg, we conducted experiments with *weighted* split ratings by multiplying the ratings of each split type with a fixed weight factor. We particularly focused on the ratio

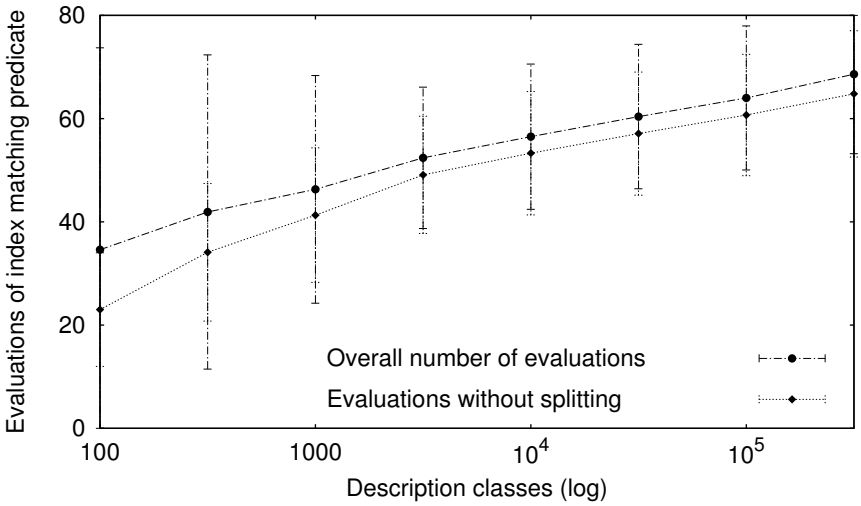


Figure 4.7: Insert costs depending on source classes.

between the weights for base and existence splits – which affect the structure of the node classes – and the weight factor for range splits. Our results show that such weight factors are not necessary: When underweighting range splits, the search costs slightly increase (e.g. by 5% for a weighting ratio of 0.5), while overweighting does not yield any noticeable changes.

4.5.3 Insertion and Splitting

Next, we consider the insertion costs measured by the number of evaluations of the index matching predicate $\rightsquigarrow_{\mathrm{I}}$ per source class and analyze the splitting with GSAlg. For this purpose, we recorded the costs and split statistics of the last 1000 successive insertions during each measurement.

Figure 4.7 shows the average insertion costs depending on the number of indexed source classes. The lower line only denotes the $\rightsquigarrow_{\mathrm{I}}$ evaluation for determining the leaf nodes to index a given source class. The upper line additionally includes the average number of $\rightsquigarrow_{\mathrm{I}}$ evaluations for splitting. It again shows the effectiveness of the SDC-Tree: Similar to searching with the query matching predicate $\rightsquigarrow_{\mathrm{Q}}$, the costs for determining the relevant leaf nodes by $\rightsquigarrow_{\mathrm{I}}$ depend logarithmically on the number of source classes. The costs for splitting are widely independent of the number of source classes and only

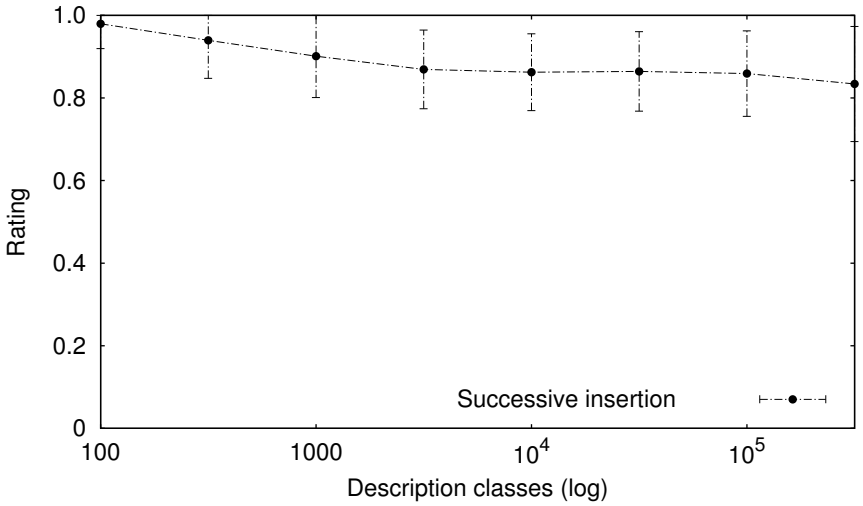


Figure 4.8: Rating of best possible split depending on source classes.

amount to about four evaluations of $\rightsquigarrow_{\mathrm{I}}$ per insertion.

Figure 4.8 gives the average rating of the best possible split depending on the number of indexed source classes. It starts with a value of about 1.0 since the first possible splits are base and existence splits only and all source classes have proper subclasses of the top class $C_\top$ as bases. Then, the possibility of range splits increases more and more, and the average rating slightly decreases as the constraint ranges of the source classes often overlap such that GSAlg cannot determine an optimal partitioning with maximum selectivity and uniform distribution. With very large numbers of source classes, this effect may increase, causing a slight decline in the average rating of the best possible split.

Figure 4.9 gives the nesting depth of the best possible split depending on the number of indexed source classes. A nesting depth of one specifies a split by means of a nested node class $\mathcal{C}on_r(N)$, a depth of two specifies a split by means of $\mathcal{C}on_{r'}(\mathcal{C}on_r(N))$, and so on. The simulated source classes have nesting depths of zero or one only, as we expect nesting depths of two or more to be rarely used in source description. This causes a standard deviation of 0.5 in our measurements. Nevertheless, the figure shows that GSAlg performs splits by means of nested node classes even for a tree indexing only 100 source classes. Thus, GSAlg rapidly adapts to source descriptions whose source classes mainly

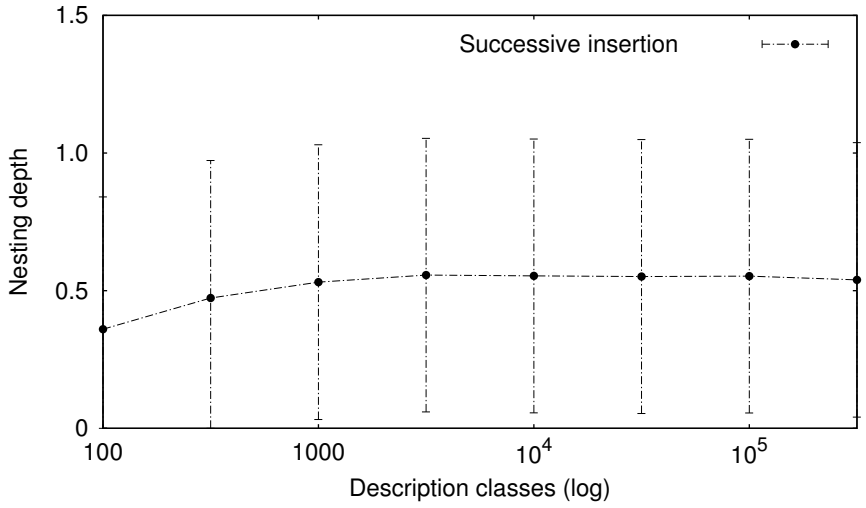


Figure 4.9: Nesting depth of best possible split depending on source classes.

differ in terms of nested ones.

4.6 Related Work

The SDC-Tree and the proposed source description formalism are related to integration systems in general and discovery services for information sources in particular.

Integration systems include federated DBMS and mediator-based approaches such as Information Manifold [LRO96], SIMS [AKS96], Infomaster [GKD97], DISCO [TRV98], MOMIS [BCV99], Object Globe [BKK+01], Amos II [RJK03], InfoSleuth [NNCB03], and Quete [KAP07] as well as peer data management systems (PDMS) such as WebFINDIT [OBB00], PeerDB [NOTZ03], Hyperion [AKK+03], and Piazza [HIM+04].[2]

These systems are based on formal mappings either to some shared global schema or between local schemas or ontologies. The classes or relations specified in the mappings give a coarse estimation of the entities represented by

[2]PDMS must not be mistaken with structured peer-to-peer systems and distributed hash tables (DHT) such as Chord [SMLN+03], which partition a given data set to a number of nodes.

a source. However, a source may only represent a small subset of the entities that belong to these classes. Several works (e.g. [LRO96,Li03,HIM+04,NFL04, KAP07]) therefore enhance these kinds of source descriptions with constraints on the attribute ranges, as illustrated in the example $V_2(c) \subseteq \text{CarForSale}(c)$, $\text{Price}(c, p)$, $p \geq 20000$ from [LRO96] to specify a database with cars priced above $20,000.

As argued above, source discovery and mapping between schemas or ontologies are two concerns – particularly when considering large-scale HIS since there may be thousands of sources that use one and the same local schema for providing information about different entities of a certain class.

There exist several dedicated approaches for source discovery in large-scale HIS. Text-source discovery services such as GlOSS [GGMT99] use statistical summaries of the textual contents to rank the sources with respect to a query of keywords.

Semantic Context Space (SCS) [GPZ05a] is a P2P overlay network for context information search. It assumes a shared upper ontology and a large number of sources providing context information by means of RDF triples to different clippings of the domain of discourse. SCS clusters all sources providing triples on the same class by creating overlay network links between them and organizes the clusters into a ringlike overlay network similar to the distributed hash table (DHT) *Chord* [SMLN+03]. Thus, SCS allows discovering all sources that provide information about queried class, but does not account for constraints on attributes or relations.

The authors of [KHM08] propose a similar approach based on an extension of the Chord protocol. The sources describe each relation of their local schemas and the belonging attributes by means of a shared ontology. These descriptions are stored in a Chord ring, using a predefined mapping from classes to integer keys. This approach allows retrieving all sources that provide information about a queried class, while taking into account the attributes of interest. However, it does not support constraints on the ranges of attributes or relations.

GloServ [AS07] also is a distributed discovery service based on a shared ontology. The nodes of the service are organized according to the IS-A hierarchy of the ontology. Sources are registered at those nodes that correspond to

classes for which they provide information. GloServ supports constraints on predefined attributes by refining the IS-A hierarchy with respect to the ranges of these attributes.

The description formalism presented in this chapter improves and refines the idea of using constraints on attributes in several ways, to facilitate precise descriptions of the sources' contents: It allows for alternative descriptions, constraints on relations, and nested constraints, as well as positive and negative matching semantics. With the SDC-Tree, we further proposed a flexible indexing approach for such source descriptions that incorporates not only the IS-A hierarchy – as the above-mentioned works – but also the existence or absence of constraints and the ranges of the constraints.

4.7 Summary

In this chapter, we proposed a powerful formalism that allows describing context models or the contents of other information sources concisely by multiple *defined classes*. The formalism refines the idea of describing information sources by constraints in several ways: It enables alternative descriptions, and it features complex, nested constraints on arbitrary attributes and relations. Finally, it allows adjusting between positive and negative matching semantics.

To enable efficient source discovery in large-scale HIS and global-scale context management in particular, we further proposed the *Source Description Class Tree* (SDC-Tree) for indexing of such descriptions by means of their defined classes. Besides, we presented a generic algorithm for splitting of leaf nodes during insertions and showed the efficiency of the SDC-Tree in a series of simulations with descriptions of potential context models derived from the OpenStreetMap database [OSM].

The proposed approach provides the foundation for realizing an efficient, precise discovery service for potentially millions of context models and providers – either as centralized, possibly replicated service or as distributed service by partitioning the SDC-Tree to a set of servers.

5 Conclusion

In this chapter, we summarize the results of this thesis and give a brief outlook on possible future research directions.

5.1 Summary

In future, context-awareness will be a key characteristic of most (mobile) applications and services. Driven by the advances in sensing technologies, millions of context models for different aspects and clippings of the physical world can be expected.

Sharing these models by a wide variety of applications poses a number of challenges. The first fundamental problem is how to provide efficient access to such immense amounts of distributed dynamic context information.

We surveyed and analyzed existing approaches for context management and showed that most of them are intended for comparatively small scenarios such as smart environments. From this introductory discussion, we derived four important subproblems of how to provide efficient access to distributed dynamic context information at a global scale: (1.) formal describing of context models, (2.) indexing of context model descriptions, (3.) efficient real-time trajectory tracking, and (4.) distributed indexing of space-partitioned trajectories. In doing so, we particularly focused on the mobility of devices and other objects – which may be entities in terms of the definition of context [Dey00, DA00], but even context providers.

For the first two subproblems, we proposed a powerful formalism allowing to describe context models concisely by multiple defined classes as well as the *Source Description Class Tree* (SDC-Tree) for indexing such descriptions. The formalism can be applied to any heterogeneous information system based on a shared ontology and refines the idea of describing information sources by

constraints in several ways: For instance, it enables alternative descriptions by multiple defined classes and complex, nested constraints on arbitrary relations.

We explained how to formulate compatible queries for context models (and thus context providers) and explained how to adjust between positive and negative matching semantics.

The SDC-Tree features multidimensional indexing capabilities for the different attributes and relations of the IS-A hierarchy of the shared ontology but also incorporates the existence of absence of constraints. Furthermore, we proposed a generic algorithm for splitting nodes of the tree, which can be used with arbitrary context ontologies.

For the third subproblem, we presented *Connection-Preserving Dead Reckoning* (CDR) and *Generic Remote Real-Time Trajectory Simplification* (GRTS), which enable efficient tracking of moving objects' trajectories with regard to storage consumption and communication cost. Both protocols use dead reckoning to inform the MOD in real-time about a simplified trajectory of each object that approximates the actual movement according to some predefined accuracy bound.

While CDR is solely based on dead reckoning, GRTS separates the tracking of the current position from the simplification of the past trajectory. Therefore, GRTS outperforms CDR by more than factor two in terms of reduction performance whereas CDR minimizes the amount of data communicated over the wireless network.

We proposed optimized algorithms with bounded space consumption and computing time for both protocols and investigated different realizations of GRTS with two important line simplification algorithms. Even with only a simple heuristic, GRTS affords substantial reduction performance at low computational costs. With an optimal line simplification algorithm, GRTS can reach more than 97% of the best possible offline reduction rate at a maximum computing time of at most 21 ms per position fix on a 1 GHz processor.

We proposed the *Distributed Trajectory Index* (DTI) to tackle the fourth subproblem. Given a number of trajectories that are partitioned spatially to multiple server for scalability reasons, a DTI realizes an overlay network according to a distributed skip list between those servers storing segments of

a certain trajectory. This approach enables to access trajectory information about a certain point in time or time interval efficiently. Our evaluation shows that the DTI scheme significantly reduces the time for routing queries to such segments compared to plain routing along the trajectory.

The extended DTI+S scheme further optimizes the efficient processing of queries on aggregates of dynamic attributes (e.g. the length of a segment or the maximum speed within a time interval) by maintaining summaries on trajectory segments.

Most important, the DTI scheme can be applied to any context information associated with trajectories.

In conclusion, by these approaches for scalable management of trajectories and context model descriptions, we presented a conceptual and algorithmic foundation for providing efficient access to distributed dynamic context information in large-scale scenarios.

5.2 Outlook

There exist several ways to extend the work of this thesis regarding future research. In the following, we first discuss two issues with the discovery of context models and then the privacy of trajectory information, which is a general problem in context-aware computing.

Discovering views on context models. The proposed description formalism and the SDC-Tree are a comprehensive solution for discovering context models and thus an important foundation for sharing context models by wide variety of applications. An obvious advancement is to also perform the processing of context information in a distributed fashion and to share and reuse partial results, which can be considered as temporary views in database terminology.

For example, within the Nexus project a middleware for distributed stream-based processing of spatial queries has been proposed [CEB+09].

To allow for sharing of such views, the proposed description formalism has to be extended to cover aggregates and other operators of respective query languages. Likewise, the definition of node classes of the SDC-Tree has to be

extended and new split types have to be introduced.

The views also cause a novel dynamism compared to the long-lasting descriptions of context models. Therefore, adaptive indexing mechanisms are needed for the SDC-Tree.

Evolution of context ontologies. For the description formalism and the SDC-Tree, we considered the shared context ontology to be unchanging – as most context management systems do. Folksonomies and other semantic tagging approaches with community projects such as OpenStreetMap [OSM], however, are one indication that context ontologies may evolve over time.

A straightforward solution to cope with such changes is to use a separate SDC-Tree for every version of the shared ontology. Yet, an integrated approach supporting an evolving ontology can be expected to require fewer costs for indexing and search.

Privacy of trajectory information. If trajectories of mobile devices can be related to persons, privacy is a very important issue. This particularly applies to tracking protocols and the (distributed) indexing of trajectories.

From a naive perspective, CDR and GRTS may be considered as very first approach to adjust the privacy of trajectory information as they allow reporting an object's movement with defined (in)accuracy. However, in consideration of attacks such as map matching, tailored approaches for managing personal trajectories at defined privacy levels are needed.

A promising idea is to split the trajectory information to multiple shares and to store them at independent providers such that the owner can adjust the privacy level regarding a certain application by the number of shares being revealed to it.

At first, this requires detailed analysis and classification of possible attacks and the definition of respective privacy metrics. Then, a principled approach for the obfuscation of position information that is robust against statistical attacks has to be researched. Finally, to preserve the privacy of trajectories, a spatiotemporal splitting approach that is robust against attacks based on route planning is needed.

Bibliography

[AdBHZ07] Mohammad Ali Abam, Mark de Berg, Peter Hachenberger, and Alireza Zarei. Streaming Algorithms for Line Simplification. In *Proc. of 23rd Symposium on Computational Geometry (SCG '07)*, pages 175–183, Gyeongju, South Korea, June 2007.

[ADL] University of California, Santa Barbara. Alexandria Digital Library (ADL). http://www.alexandria.ucsb.edu/, accessed on 9th July 2009.

[AHPMW05] Pankaj K. Agarwal, Sariel Har-Peled, Nabil H. Mustafa, and Yusu Wang. Near-Linear Time Approximation Algorithms for Curve Simplification. *Algorithmica*, 42(3–4):203–219, July 2005.

[AKK+03] Marcelo Arenas, Vasiliki Kantere, Anastasios Kementsietsidis, Iluju Kiringa, Renée J. Miller, and John Mylopoulos. The Hyperion Project: From Data Integration to Data Coordination. *ACM SIGMOD Record*, 3(3):53–58, September 2003.

[AKS96] Yigal Arens, Craig A. Knoblock, and Wei-Min Shen. Query Reformulation for Dynamic Information Integration. *Journal of Intelligent Information Systems*, 6(2–3):99–130, June 1996.

[AS07] Knarig Arabshian and Henning Schulzrinne. An Ontology-based Hierarchical Peer-to-Peer Global Service Discovery System. *Journal of Ubiquitous Computing and Intelligence (JUCI)*, 1(2):133–144, December 2007.

[AV00] P. K. Agarwal and K. R. Varadarajan. Efficient Algorithms for Approximating Polygonal Chains. *Discrete and Computational Geometry*, 23(2):273–291, February 2000.

[AWEKAS] Automatic Weather Map System (Automatisches Wetterkarten System – AWEKAS). http://www.awekas.at/, accessed on 26th October 2009.

[BC04] Gregory Biegel and Vinny Cahill. A Framework for Developing Mobile, Context-aware Applications. In *Proc. of 2nd IEEE Conf. on Pervasive Computing and Communications (PerCom '04)*, pages 361–365, Orlando, FL, USA, March 2004.

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, Cambridge, UK, 2003.

[BCM06] Dario Bottazzi, Antonio Corradi, and Rebecca Montanari. Context-Aware Middleware Solutions for Anytime and Anywhere Emergency Assistance to Elderly People. *IEEE Communications Magazine*, 44(4):82–90, April 2006.

[BCQ+07] Cristiana Bolchini, Carlo A. Curino, Elisa Quintarelli, Fabio A. Schreiber, and Letizia Tanca. A Data-oriented Survey of Context Models. *ACM SIGMOD Record*, 36(4):19–26, December 2007.

[BCV99] S. Bergamaschi, S. Castano, and M. Vincini. Semantic Integration of Semistructured and Structured Data Sources. *ACM SIGMOD Record*, 28(1):54–59, March 1999.

[BD05] Christian Becker and Frank Dürr. On location models for ubiquitous computing. *Personal and Ubiquitous Computing*, 9(1):20–31, January 2005.

[BDR07] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int'l Journal of Ad Hoc and Ubiquitous Computing (IJAHUC)*, 2(4):263–277, June 2007.

[BKK+01] Reinhard Braumandl, Markus Keidl, Alfons Kemper, Donald Kossmann, Alexander Kreutz, Stefan Seltzsam, and Konrad Stocker. ObjectGlobe: Ubiquitous query processing on the Internet. *VLDB Journal*, 10(1):48–71, 2001.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proc. of 1990 ACM SIGMOD Int'l Conf. on Management of Data*, pages 322–331, Atlantic City, NJ, USA, May 1990.

[BP00] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An In-Building RF-based User Location and Tracking System. In *Proc. of 19th Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM 2000)*, volume 2, pages 775–784, March 2000.

[CC92] W. S. Chan and F. Chin. Approximation of Polygonal Curves with Minimum Number of Line Segments. In *Proc. of 3rd Int'l Symposium on Algorithms and Computation (ISAAC '92)*, pages 378–387, Nagoya, Japan, December 1992.

[CDMF00] Keith Cheverst, Nigel Davies, Keith Mitchell, and Adrian Friday. Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE Project. In *Proc. of 6th Int'l Conf. on Mobile Computing and Networking (MobiCom 2000)*, pages 20–31, Boston, MA, USA, August 2000.

[CEB+09] Nazario Cipriani, Mike Eissele, Andreas Brodt, Matthias Großmann, and Bernhard Mitschang. NexusDS: A Flexible and Extensible Middleware for Distributed Stream Processing. In *Proc. of 13th Int'l Database Engineering and Applications Symposium (IDEAS '09)*, pages 152–161, Cetraro, Calabria, Italy, September 2009.

[CFJ+04] Harry Chen, Tim Finin, Anupam Joshi, Lalana Kagal, Filip Perich, and Dipanjan Chakraborty. Intelligent Agents Meet the Semantic Web in Smart Spaces. *IEEE Internet Computing*, 8(6):69–79, November 2004.

[CJNP04] Alminas Čivilis, Christian S. Jensen, Jovita Nenortaitė, and Stardas Pakalnis. Efficient tracking of moving objects with precision guarantees. In *Proc. of 1st Int'l Conf. on Mobile and*

Ubiquitous Systems (MobiQuitous '04), pages 164–173, Boston, MA, USA, August 2004.

[CJP05] Alminas Čivilis, Christian S. Jensen, and Stardas Pakalnis. Techniques for Efficient Road-Network-Based Tracking of Moving Objects. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(5):698–712, May 2005.

[CL08] Adrian David Cheok and Yue Li. Ubiquitous interaction with positioning and navigation using a novel light sensor-based information transmission system. *Personal and Ubiquitous Computing*, 12(6):445–458, August 2008.

[CPFJ04] Harry Chen, Filip Perich, Timothy W. Finin, and Anupam Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *Proc. of 1st Int'l Conf. on Mobile and Ubiquitous Systems (MobiQuitous '04)*, pages 258–267, Cambridge, MA, USA, August 2004.

[CRHPP09] Ching-Hua Chen-Ritzo, Colin Harrison, Jurij Paraszczak, and Francis Parr. Instrumenting the planet. *IBM Journal of Research and Development*, 53(3):1:1–1:16, May 2009.

[CWOP] Citizen Weather Observer Program (CWOP). http://www.wxqa.com/, accessed on 26th October 2009.

[CWT06] Hu Cao, Ouri Wolfson, and Goce Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *VLDB Journal*, 15(3):211–228, September 2006.

[DA00] Anind K. Dey and Gregory D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *Proc. of CHI 2000 Workshop on the What, Who, Where, When and How of Context-Awareness*, The Hague, Netherlands, April 2000.

[DAS01] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, 16(2-4):97–166, 2001.

[Dey00] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, November 2000.

[DKM+03] Alan Dearle, Graham N. C. Kirby, Ronald Morrison, Andrew McCarthy, Kevin Mullen, Yanyan Yang, Richard C. H. Connor, Paula Welen, and Andy Wilson. Architectural Support for Global Smart Spaces. In *Proc. of 4th Int'l Conf. on Mobile Data Management (MDM '03)*, pages 153–164, Melbourne, Australia, January 2003.

[Dou04] Paul Dourish. What We Talk About When We Talk About Context. *Personal and Ubiquitous Computing*, 8(1):19–30, February 2004.

[DP73] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122, December 1973.

[DPG+08] Frank Dürr, Jonas Palauro, Lars Geiger, Ralph Lange, and Kurt Rothermel. Ein kontextbezogener Instant-Messaging-Dienst auf Basis des XMPP-Protokolls. In *5. GI/ITG KuVS Fachgespräch Ortsbezogene Anwendungen und Dienste*, Nuremberg, Germany, September 2008.

[FC04] Patrick Fahy and Siobhán Clarke. CASS – Middleware for Mobile Context-Aware Applications. In *Proc. of the MobiSys 2004 Workshop on Context Awareness*, Boston, MA, USA, June 2004.

[FGNS00] Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli, and Markus Schneider. A Data Model and Data Structures for Moving Objects Databases. In *Proc. of 2000 ACM SIGMOD Int'l Conf. on Management of Data*, pages 319–330, Dallas, TX, USA, May 2000.

[FireEagle] Yahoo! Inc. Fire Eagle. http://fireeagle.yahoo.net/, accessed on 22nd April 2010.

[FLR07] Tobias Farrell, Ralph Lange, and Kurt Rothermel. Energy-efficient Tracking of Mobile Objects with Early Distance-based Reporting. In *Proc. of 4th Int'l Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous '07)*, Philadelphia, PA, USA, August 2007.

[GdAD06] Ralf Hartmut Güting, Victor Teixeira de Almeida, and Zhiming Ding. Modeling and querying moving objects in networks. *VLDB Journal*, 15(2):165–190, June 2006.

[GDR09] Lars Geiger, Frank Dürr, and Kurt Rothermel. On Contextcast: A Context-Aware Communication Mechanism. In *Proc. of IEEE Int'l Conf. on Communications (ICC '09)*, Dresden, Germany, June 2009.

[GGMT99] Luis Gravano, Hétor García-Molina, and Anthony Tomasic. GlOSS: Text-Source Discovery over the Internet. *ACM Transactions on Database Systems (TODS)*, 24(2):229–264, June 1999.

[GKD97] Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: An Information Integration System. In *Proc. of 1997 ACM SIGMOD Int'l Conf. on Management of Data*, pages 539–542, Tucson, AZ, USA, May 1997.

[GKM+07] Joachim Gudmundsson, Jyrki Katajainen, Damian Merrick, Cahya Ong, and Thomas Wolle. Compressing spatio-temporal trajectories. In *Proc. of 18th Int'l Symposium on Algorithms and Computation (ISAAC '07)*, pages 763–775, Sendai, Japan, December 2007.

[GPS-Perf] U.S. Department of Defense (DoD). Global Positioning System Standard Positioning Service Performance Standard, October 2001.

[GPZ05a] Tao Gu, Hung Keng Pung, and Daqing Zhang. A Peer-to-Peer Overlay for Context Information Search. In *Proc. of 14th Int'l Conf. on Computer Communications and Networks (ICCCN '05)*, pages 395–400, San Diego, CA, USA, October 2005.

[GPZ05b] Tao Gu, Hung Keng Punga, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Application*, 28(1):1–18, January 2005.

[GS05] Ralf Hartmut Güting and Markus Schneider. *Moving Objects Databases*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2005.

[GWPZ04] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang. An Ontology-based Context Model in Intelligent Environments. In *Proc. of Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS '04)*, pages 270–275, San Diego, CA, USA, January 2004.

[Haa97] Zygmunt J. Haas. The Routing Algorithm for the Reconfigurable Wireless Networks. In *Proc. of 6th Int'l Conf. on Universal Personal Communications (ICUPC '97)*, pages 562–566, San Diego, CA, USA, October 1997.

[HB01] Jeffrey Hightower and Gaetano Borriello. Location Systems for Ubiquitous Computing. *IEEE Computer*, 34(8):57–66, August 2001.

[HBZ+06] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. CarTel: A Distributed Mobile Sensor Computing System. In *Proc. of 4th Int'l Conf. on Embedded Networked Sensor Systems (SenSys '06)*, pages 125–138, Boulder, CO, USA, October 2006.

[HGNM08] Nicola Hönle, Matthias Großmann, Daniela Nicklas, and Bernhard Mitschang. Preprocessing Position Data of Mobile Objects. In *Proc. of 9th Int'l Conf. on Mobile Data Management (MDM '08)*, pages 41–48, Beijing, China, April 2008.

[HHS+99] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The Anatomy of a Context-Aware Application.

In *Proc. of 5th ACM/IEEE Int'l Conf. on Mobile Computing and Networking (MobiCom '99)*, pages 59–68, August 1999.

[HIM+04] Alon Y. Halevy, Zachary G. Ives, Jayant Madhavan, Peter Mork, Dan Suciu, and Igor Tatarinov. The Piazza Peer Data Management System. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(7):787–798, July 2004.

[HKBE07] Andreas Hub, Stefan Kombrink, Klaus Bosse, and Thomas Ertl. TANIA – A Tactile-Acoustical Navigation and Information Assistant for the 2007 CSUN Conference. In *Proc. of 22nd Int'l Technology and Persons with Disabilities Conference (CSUN '07)*, Los Angeles, CA, USA, March 2007.

[HKL+99] Fritz Hohl, Uwe Kubach, Alexander Leonhardi, Kurt Rothermel, and Markus Schwehm. Next Century Challenges: Nexus – An Open Global Infrastructure for Spatial-Aware Applications. In *Proc. of 5th ACM/IEEE Int'l Conf. on Mobile Computing and Networking (MobiCom '99)*, pages 249–255, Seattle, WA, USA, August 1999.

[HKTG06] Marios Hadjieleftheriou, George Kollios, Vassilis J. Tsotras, and Dimitrios Gunopulos. Indexing spatiotemporal archives. *VLDB Journal*, 15(2):143–164, June 2006.

[HMEZ+05] Sumi Helal, William Mann, Hicham El-Zabadani, Jeffrey King, Youssef Kaddoura, and Erwin Jansen. The Gator Tech Smart House: A Programmable Pervasive Space. *IEEE Computer*, 38(3):50–60, March 2005.

[HS94] John Hershberger and Jack Snoeyink. An O(n log n) Implementation of the Douglas-Peucker Algorithm for Line Simplification. In *Proc. of 10th Symposium on Computational Geometry*, pages 383–384, Stony Brook, NY, USA, June 1994.

[HSP+03] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-Awareness on Mobile Devices – The Hydrogen Approach. In *Proc. of 36th Hawaii Int'l Conf. on System Sciences (HICSS '03)*, pages 10–19, January 2003.

[HW08] Mordechai (Muki) Haklay and Patrick Weber. OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive Computing*, 7(4):12–18, October 2008.

[II88] Masao Iri and Hiroshi Imai. *Computational Morphology*, chapter Polygonal Approximations of a Curve – Formulations and Algorithms, pages 71–86. North-Holland Publishing Company, 1988.

[InstaMapper] InstaMapper LLC. http://www.instamapper.com/, accessed on 22nd April 2010.

[iPhone3GS] Apple Inc. iPhone 3GS Technical Specifications. http://www.apple.com/iphone/specs.html, accessed on 21th October 2009.

[ISO-19762-5] International Organization for Standardization and International Electrotechnical Commission. ISO/IEC 19762-5:2008 Information technology – Automatic identification and data capture (AIDC) techniques – Harmonized vocabulary – Part 5: Locating systems, June 2008.

[ISO-9075] International Organization for Standardization and International Electrotechnical Commission. ISO/IEC 9075:2008 Information technology – Database languages – SQL, 2008.

[JTS] Vivid Solutions Inc. Java Topology Suite (JTS). http://sourceforge.net/projects/jts-topo-suite/, accessed on 24th June 2009.

[KAP07] Haridimos Kondylakis, Anastasia Analyti, and Dimitris Plexousakis. Quete: Ontology-Based Query System for Distributed Sources. In *Proc. of 11th East European Conf. on Advances in Databases and Information Systems (ADBIS '07)*, pages 359–375, Varna, Bulgaria, September 2007.

[KB03] Christian Kray and Jörg Baus. A survey of mobile guides. In *Proc. of Workshop on HCI in Mobile Guides (MGUIDES) at MobileHCI '03*, Udine, Italy, September 2003.

[KHM08] Raddad Al King, Abdelkader Hameurlain, and Franck Morvan. Ontology-Based Data Source Localization in a Structured

Peer-to-Peer Environment. In *Proc. of 12th Int'l Database Engineering and Applications Symposium (IDEAS '08)*, pages 9–18, Coimbra, Portugal, September 2008.

[Kle00] Jon Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proc. of 32nd ACM Symposium on Theory of Computing (STOC 2000)*, pages 163–170, Portland, OR, USA, May 2000.

[KMK+03] Panu Korpipää, Jani Mäntyjärvi, Juha Kela, Heikki Keränen, and Esko-Juhani Malm. Managing Context Information in Mobile Devices. *IEEE Pervasive Computing*, 2(3):42–51, July 2003.

[Latitude] Google Inc. Google Latitude. http://www.google.com/latitude, accessed on 22nd April 2010.

[LCC+05] Anthony LaMarca, Yatin Chawathe, Sunny Consolvo, Jeffrey Hightower, Ian Smith, James Scott, Tim Sohn, James Howard, Jeff Hughes, Fred Potter, Jason Tabert, Pauline Powledge, Gaetano Borriello, and Bill Schilit. Place Lab: Device Positioning Using Radio Beacons in the Wild. In *Proc. of 3rd Int'l Conf. on Pervasive Computing (Pervasive '05)*, pages 116–133, May 2005.

[LCG+09] Ralph Lange, Nazario Cipriani, Lars Geiger, Matthias Großmann, Harald Weinschrott, Andreas Brodt, Matthias Wieland, Stamatia Rizou, and Kurt Rothermel. Making the World Wide Space Happen: New Challenges for the Nexus Context Platform (Work-in-Progress Paper). In *Proc. of 7th IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom '09)*, pages 300–303, Galveston, TX, USA, March 2009.

[LDR08a] Ralph Lange, Frank Dürr, and Kurt Rothermel. Online Trajectory Data Reduction using Connection-preserving Dead Reckoning. In *Proc. of 5th Int'l Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous '08)*, Dublin, Ireland, July 2008.

[LDR08b] Ralph Lange, Frank Dürr, and Kurt Rothermel. Scalable Processing of Trajectory-Based Queries in Space-Partitioned Moving Objects Databases. In *Proc. of 16th ACM SIGSPATIAL Int'l Conf. on Advances in Geographic Information Systems (ACM GIS '08)*, pages 270–279, Irvine, CA, USA, November 2008.

[LDR10a] Ralph Lange, Frank Dürr, and Kurt Rothermel. Efficient Tracking of Moving Objects using Generic Remote Trajectory Simplification (Demo Paper). In *Proc. of 8th IEEE Int'l Conf. on Pervasive Computing and Communications Workshops (PerCom Workshops '10)*, Mannheim, Germany, March 2010.

[LDR10b] Ralph Lange, Frank Dürr, and Kurt Rothermel. Indexing Source Descriptions based on Defined Classes. In *Proc. of 14th Int'l Database Engineering and Applications Symposium (IDEAS '10)*, Montreal, QC, Canada, August 2010. Accepted for publication, to appear.

[LFDR09] Ralph Lange, Tobias Farrell, Frank Dürr, and Kurt Rothermel. Remote Real-Time Trajectory Simplification. In *Proc. of 7th IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom '09)*, pages 184–193, Galveston, TX, USA, March 2009.

[LFG+03] José Antonio Cotelo Lema, Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli, and Markus Schneider. Algorithms for Moving Objects Databases. *The Computer Journal*, 46(6):680–712, 2003.

[Li03] Chen Li. Using Constraints to Describe Source Contents in Data Integration Systems. *IEEE Intelligent Systems*, 18(5):49–53, September 2003.

[LKNZ08] Liqian Luo, Aman Kansal, Suman Nath, and Feng Zhao. Sharing and Exploring Sensor Streams over Geocentric Interfaces. In *Proc. of 16th ACM SIGSPATIAL Int'l Conf. on Advances in*

Geographic Information Systems (ACM GIS '08), pages 3–12, Irvine, CA, USA, November 2008.

[LLN03] Michael Liljenstam, Jason Liu, and David M. Nicol. Development of an Internet Backbone Topology for Large-scale Network Simulations. In *Proc. of 2003 Winter Simulation Conference*, pages 694–702, New Orleans, LA, USA, December 2003.

[LNR02] Alexander Leonhardi, Christian Nicu, and Kurt Rothermel. A Map-based Dead-reckoning Protocol for Updating Location Information. In *Proc. of Int'l Parallel and Distributed Processing Symposium (IPDPS '02)*, pages 193–200, April 2002.

[LR01] Alexander Leonhardi and Kurt Rothermel. A Comparison of Protocols for Updating Location Information. *Cluster Computing: The Journal of Networks, Software Tools and Applications*, 4(4):355–367, October 2001.

[LR02] Alexander Leonhardi and Kurt Rothermel. Architecture of a Large-scale Location Service. In *Proc. of 22nd Int'l Conf. on Distributed Computing Systems (ICDCS '02)*, pages 465–466, Vienna, Austria, July 2002.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. of 22th Int'l Conf. on Very Large Data Bases (VLDB '96)*, pages 251–262, Mumbai (Bombay), India, September 1996.

[LWG+09] Ralph Lange, Harald Weinschrott, Lars Geiger, André Blessing, Frank Dürr, Kurt Rothermel, and Hinrich Schütze. On a Generic Uncertainty Model for Position Information. In *Proc. of 1st Int'l Workshop on Quality of Context (QuaCon '09)*, pages 76–87, Stuttgart, Germany, June 2009.

[MapTracks] Tinderhouse Limited. Map My Tracks. http://www.mapmytracks.com/, accessed on 22nd April 2010.

[MdB04] Nirvana Meratnia and Rolf A. de By. Spatiotemporal Compression Techniques for Moving Point Objects. In *Proc. of 9th Int'l*

Conf. on Extending Database Technology (EDBT '04), pages 765–782, Heraklion, Crete, March 2004.

[ME01] Pratap Misra and Per Enge. *Global Positioning System: Signals, Measurements and Performance*. Ganga-Jumuna Press, 2001.

[MFHH02] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, December 2002.

[MGA03] Mohamed F. Mokbel, Thanaa M. Ghanem, and Walid G. Aref. Spatio-Temporal Access Methods. *IEEE Data Engineering Bulletin*, 26(2):40–49, June 2003.

[MLF+08] Emiliano Miluzzo, Nicholas D. Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B. Eisenman, Xiao Zheng, and Andrew T. Campbell. Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application. In *Proc. of 6th ACM Conf. on Embedded Networked Sensor Systems (SenSys '08)*, pages 337–350, Raleigh, NC, USA, November 2008.

[MMR+08] Rohan Narayana Murty, Geoffrey Mainland, Ian Rose, Atanu Roy Chowdhury, Abhimanyu Gosaint, Josh Berst, and Matt Welsh. CitySense: An Urban-Scale Wireless Sensor Network and Testbed. In *Proc. of IEEE 2008 Conf. on Technologies for Homeland Security*, pages 583–588, Waltham, MA, USA, May 2008.

[NFL04] Felix Naumann, Johann-Christoph Freytag, and Ulf Leser. Completeness of integrated information sources. *Information Systems*, 29(7):583–615, October 2004.

[NGMW08] Daniela Nicklas, Matthias Großmann, Jorge Mínguez, and Matthias Wieland. Adding High-level Reasoning to Efficient Low-level Context Management: a Hybrid Approach. In *Proc.*

of 6th IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom '08), pages 447–452, Hong Kong, China, March 2008.

[NGS+01] Daniela Nicklas, Matthias Großmann, Thomas Schwarz, Steffen Volz, and Bernhard Mitschang. A Model-Based, Open Architecture for Mobile, Spatially Aware Applications. In *Proc. of 7th Int'l Symposium on Advances in Spatial and Temporal Databases (SSTD '01)*, pages 117–135, Redondo Beach, CA, USA, July 2001.

[NM04] Daniela Nicklas and Bernhard Mitschang. On building location aware applications using an open platform based on the NEXUS Augmented World Model. *Software and Systems Modeling*, 3(4):303–313, December 2004.

[NNCB03] Marian (Misty) Nodine, Anne Hee Hiong Ngu, Anthony Cassandra, and William G. Bohrer. Scalable Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth(TM). *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(5):1082–1098, September 2003.

[NOTZ03] Wee Siong Ng, Beng Chin Ooi, Kian-Lee Tan, and Aoying Zhou. PeerDB: A P2P-based System for Distributed Data Sharing. In *Proc. of 19th Int'l Conf. on Data Engineering (ICDE '03)*, pages 633–644, Bangalore, India, March 2003.

[NR07] Jinfeng Ni and Chinya V. Ravishankar. Indexing Spatio-Temporal Trajectories with Efficient Polynomial Approximations. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(5):663–678, May 2007.

[OA00] Robert J. Orr and Gregory D. Abowd. The Smart Floor: A Mechanism for Natural User Identification and Tracking. In *Extended Abstracts on Human factors in Computing Systems (CHI 2000)*, pages 275–276, The Hague, The Netherlands, April 2000.

[OBB00] M. Ouzzani, B. Benatallah, and A. Bouguettaya. Ontological Approach for Information Discovery in Internet Databases. *Distributed and Parallel Databases*, 8(3):367–392, July 2000.

[OCS06] Sarah Olofsson, Veronica Carlsson, and Jessica Sjölander. The friend locator: supporting visitors at large-scale events. *Personal and Ubiquitous Computing*, 10(2–3):84–89, April 2006.

[OpenCyc] Cycorp, Inc. OpenCyc. http://www.opencyc.org/, accessed on 1st December 2009.

[OpenGIS] Open Geospatial Consortium (OGC), Inc. OpenGIS(R) Implementation Specification for Geographic information – Simple feature access – Part 1: Common architecture, October 2006.

[OQO] OQO, Inc. http://www.oqo.com/, accessed on 7th October 2009.

[OSM] OpenStreetMap (OSM). http://www.openstreetmap.org/, accessed on 20th May 2010.

[OWL] World Wide Web Consortium (W3C). OWL Web Ontology Language: Reference (W3C Recommendation), February 2004.

[PJ99] Dieter Pfoser and Christian S. Jensen. Capturing the Uncertainty of Moving-Object Representations. In *Proc. of 6th Int'l Symposium on Advances in Spatial Databases (SSD '99)*, pages 111–131, Hong Kong, China, July 1999.

[PJT00] Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. Novel Approaches to the Indexing of Moving Object Trajectories. In *Proc. of 26th Int'l Conf. on Very Large Data Bases (VLDB 2000)*, pages 395–406, Cairo, Egypt, September 2000.

[PPS06a] Michalis Potamias, Kostas Patroumpas, and Timos Sellis. Amnesic Online Synopses for Moving Objects. In *Proc. of 15th ACM Int'l Conf. on Information and Knowledge Management (CIKM '06)*, pages 784–785, Arlington, VA, USA, November 2006.

[PPS06b] Michalis Potamias, Kostas Patroumpas, and Timos Sellis. Sampling Trajectory Streams with Spatiotemporal Criteria. In *Proc. of 18th Int'l Conf. on Scientific and Statistical Database Management (SSDBM '06)*, pages 275–284, Vienna, Austria, July 2006.

[Proton] Semantic Knowledge Technologies (SEKT) Project. PROTON Ontology (PROTo ONtology). http://proton.semanticweb.org/, accessed on 9th July 2009.

[PS01] Evaggelia Pitoura and George Samaras. Locating Objects in Mobile Computing. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(4):571–592, July 2001.

[PSJ06] Mindaugas Pelanis, Simonas Saltenis, and Christian S. Jensen. Indexing the Past, Present, and Anticipated Future Positions of Moving Objects. *ACM Transactions on Database Systems (TODS)*, 31(1):255–298, March 2006.

[Pug90] William Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Communications of the ACM*, 33(6):668–676, June 1990.

[Ran94] James Rankin. GPS and Differential GPS: An Error Model for Sensor Simulation. In *Position Location and Navigation Symposium*, pages 260–266, April 1994.

[RCC92] David A. Randell, Zhan Cui, and Anthony G. Cohn. A Spatial Logic based on Regions and Connection. In *Proc. of 3rd Int'l Conf. on Knowledge Representation and Reasoning (KR '92)*, pages 165–176, Cambridge, MA, USA, October 1992.

[RDD+03] Kurt Rothermel, Dominique Dudkowski, Frank Dürr, Martin Bauer, and Christian Becker. Ubiquitous Computing – More than Computing Anytime Anyplace? In *Proc. of 49th Photogrammetric Week*, Stuttgart, Germany, September 2003.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network.

In *Proc. of 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01)*, pages 161–172, San Diego, CA, USA, August 2001.

[RHC+02] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, October 2002.

[RHD+10] Stamatia Rizou, Kai Häussermann, Frank Dürr, Nazario Cipriani, and Kurt Rothermel. A System for Distributed Context Reasoning. In *Proc. of 6th Int'l Conf. on Autonomic and Autonomous Systems (ICAS '10)*, pages 84–89, Cancun, Mexico, March 2010.

[Rid07] Kirstin Ridley (Reuters). Global mobile phone use to pass 3 billion. http://uk.reuters.com/article/idUKL2712199720070627, June 2007.

[RJK03] T. Risch, V. Josifovski, and T. Katchaounov. *Functional Approach to Data Management – Modeling, Analyzing and Integrating Heterogeneous Data*, chapter Functional Data Integration in a Distributed Mediator System. Springer, 2003.

[RMCM03] Anand Ranganathan, Robert E. McGrath, Roy H. Campbell, and M. Dennis Mickunas. Use of ontologies in a pervasive computing environment. *The Knowledge Engineering Review*, 18(3):209–220, September 2003.

[Rot03] Jörg Roth. Accessing Location Data in Mobile Environments – The Nimbus Location Model. In *Proc. of Workshop on Mobile and Ubiquitous Information Access at MobileHCI '03*, pages 256–270, Udine, Italy, September 2003.

[RTA05] Dimitrios Raptis, Nikolaos Tselios, and Nikolaos Avouris. Context-based Design of Mobile Applications for Museums: A Survey of Existing Practices. In *Proc. of 7th Int'l Conf. on Human Computer Interaction with Mobile Devices and Services*

(MobileHCI '05), pages 153–160, Salzburg, Austria, September 2005.

[SAW94] Bill N. Schilit, Norman Adams, and Roy Want. Context-Aware Computing Applications. In *Proc. of IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '94)*, pages 89–101, Santa Cruz, CA, USA, 1994.

[SBG99] Albrecht Schmidt, Michael Beigl, and Hans-Werner Gellersen. There is more to context than location. *Computers & Graphics Journal*, 23(6):893–902, December 1999.

[SBGP04] Adam Smith, Hari Balakrishnan, Michel Goraczko, and Nissanka Priyantha. Tracking moving devices with the cricket location system. In *Proc. of 2nd Int'l Conf. on Mobile systems, applications, and services (MobiSys '04)*, pages 190–202, June 2004.

[Sch02] Albrecht Schmidt. *Ubiquitous Computing – Computing in Context*. PhD thesis, Lancaster University, Lancaster, UK, November 2002.

[SMLN+03] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking*, 11(1):2003, February 2003.

[SPARQL] World Wide Web Consortium (W3C). SPARQL Query Language for RDF (W3C Recommendation), January 2008.

[ST94] Bill N. Schilit and Marvin M. Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5):22–32, September 1994.

[SUMO] Articulate Software. Suggested Upper Merged Ontology (SUMO). http://www.ontologyportal.org/, accessed on 9th July 2009.

[SWCD98] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Querying the Uncertain Position of Moving Objects. *Lecture Notes in Computer Science (LNCS)*, 1399:310–337, June 1998.

[TCS+06] Goce Trajcevski, Hu Cao, Peter Scheuermann, Ouri Wolfson, and Dennis Vaccaro. Online Data Reduction and the Quality of History in Moving Objects Databases. In *Proc. of 5th ACM Int'l Workshop on Data Engineering for Wireless and Mobile Access (MobiDE '06)*, Chicago, IL, USA, June 2006.

[TDSC07] Goce Trajcevski, Hui Ding, Peter Scheuermann, and Isabel F. Cruz. BORA: Routing and Aggregation for Distributed Processing of Spatio-Temporal Range Queries. In *Proc. of 8th Int'l Conf. on Mobile Data Management (MDM '07)*, pages 36–43, Mannheim, Germany, May 2007.

[TJ07] Dalia Tiešytė and Christian S. Jensen. Recovery of Vehicle Trajectories from Tracking Data for Analysis Purposes. In *Proc. of 6th European Congress and Exhibition on Intelligent Transport Systems and Services*, Aalborg, Denmark, June 2007.

[TRV98] Anthony Tomasic, Louiqa Raschid, and Patrick Valduriez. Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 10(5):808–823, September 1998.

[Var96] Kasturi R. Varadarajan. Approximating Monotone Polygonal Curves Using the Uniform Metric. In *Proc. of 12th Symposium on Computational Geometry*, pages 311–318, Philadelphia, PA, USA, May 1996.

[vSHHT98] Maarten van Steen, Franz J. Hauck, Philip Homburg, and Andrew S. Tanenbaum. Locating Objects in Wide-Area Systems. *IEEE Communications Magazine*, 36(1):104–109, January 1998.

[WDC+04] Xiaohang Wang, Jin Song Dong, ChungYau Chin, Sanka, Ravipriya Hettiarachchi, and Daqing Zhang. Semantic Space:

An Infrastructure for Smart Spaces. *IEEE Pervasive Computing*, 3(3):32–39, July 2004.

[Wei91] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991.

[WHFG92] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems (TOIS)*, 10(1):91–102, January 1992.

[Wintec] Wintecronics Ltd. http://www.wintec.com.tw/, accessed on 7th October 2009.

[WSCY99] Ouri Wolfson, A. Prasad Sistla, Sam Chamberlain, and Yelena Yesha. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases*, 7(3):257–287, July 1999.

[XECA07] Xiaopeng Xiong, Hicham G. Elmongui, Xiaoyong Chai, and Walid G. Aref. PLACE*: A Distributed Spatio-temporal Data Stream Management System for Moving Objects. In *Proc. of 8th Int'l Conf. on Mobile Data Management (MDM '07)*, pages 44–51, Mannheim, Germany, May 2007.

[YCDN07] Juan Ye, Lorcan Coyle, Simon Dobson, and Paddy Nixon. Ontology-based models in pervasive computing systems. *The Knowledge Engineering Review*, 22(4):315–347, December 2007.

[Zog09] Jean-Marie Zogg (u-blox AG). Essentials of Satellite Navigation (Compendium). http://www.u-blox.com/, February 2009.

[ZZL07] Jianjun Zhang, Gong Zhang, and Ling Liu. GeoGrid: A Scalable Location Service Network. In *Proc. of 27th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS '07)*, Toronto, Ontario, Canada, June 2007.

Index