Efficient Tracking of Moving Objects using Generic Remote Trajectory Simplification

Ralph Lange Frank Dürr Kurt Rothermel
Institute of Parallel and Distributed Systems
Universität Stuttgart, Germany
Email: <firstname.lastname>@ipvs.uni-stuttgart.de

***Abstract*—Position information of moving objects plays a vital role in many pervasive applications. Therefore, moving objects databases (MODs), which can manage trajectory data of a number objects, are used in many pervasive systems. A crucial problem with MODs is how to efficiently track a remote object's trajectory in real-time, i.e. how to continuously report the sensed trajectory data to the MOD with minimal effort. For this purpose, we present a prototypical implementation of the Generic Remote Trajectory Simplification (GRTS) protocol, which optimizes storage consumption, processing, and communication costs. Our prototypical system includes a fully functional MOD as well as map-based mobile applications for subnotebooks and smartphones to illustrate the functioning of GRTS.**

I. Introduction

Many pervasive applications rely on information about the current and past positions of moving objects such as vehicles and people. Moving objects' trajectories generally are represented as polylines in time and space where the vertices are time-stamped positions acquired by positioning sensors (e.g. GPS receivers). Many pervasive systems use moving objects databases (MODs) for managing the trajectories of multiple objects at a central database server in real-time.

Reporting every sensed position to a MOD causes high communication costs and may rapidly drain the objects' energy supplies. It also consumes a lot of storage capacity and causes high processing costs for queries. In general, MODs therefore only store an approximation of each trajectory, given by a subset of the sensed positions. This approach can reduce the number of vertices substantially as linear movements yield many redundant vertices, cf. Figure 1.

Computing such a simplified trajectory that does not deviate by more than a certain accuracy bound ϵ from the actual one is a line simplification problem. To optimize the communication between a moving object and a MOD, the simplification should be performed on the object to transmit those positions only that are required as vertices of the simplified trajectory. Still, the simplified trajectory should be available at the MOD in real-time for the use by applications.

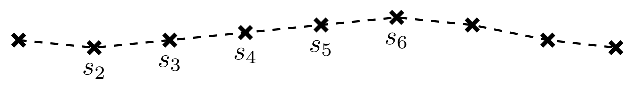


Figure 1. Positions s_3 to s_5 are redundant for line section by s_2 and s_6.

Therefore, we proposed the Generic Remote Trajectory Simplification (GRTS) protocol in [1]. It combines dead reckoning for tracking the current position of a moving object with a line simplification algorithm for computing a simplified trajectory. GRTS achieves 90 to 98% of the reduction performance of retrospective offline reduction with the optimal line simplification algorithm by Imai and Iri [2].

This demo paper is an extension of [1], focusing on the description of a prototypical implementation of GRTS. Therefore, we only give a brief description of GRTS in Section II and explain two novel improvements compared with [1]. Then, we explain the implementation in Section III, before we discuss the demonstrated functionality in Section IV. Finally, a summary is given in Section V.

II. The GRTS Protocol

GRTS is executed independently for each moving object and distinguishes three representations of an object's trajectory, referred to as *actual*, *sensed*, and *simplified trajectory*.

The actual trajectory gives the exact movement of the object over time and thus is an arbitrary continuous function $\vec{a} : \mathbb{R} \mapsto \mathbb{R}^d$ from time to plane ($d = 2$) or space ($d = 3$).

The sensed trajectory is given by the sequence of *sensed positions* $(s_1, s_2, \ldots)$. Each s_i has to two attributes t and $\vec{p}$ where $s_i.\vec{p}$ denotes the position recorded at time $s_i.t$. Two consecutive sensed positions s_i and s_{i+1} define a *spatiotemporal line section* by linear interpolation over $[s_i.t, s_{i+1}.t]$. Therefore, all sensed positions together define a continuous piecewise linear function $\vec{s}(t)$. Geometrically, $\vec{s}(t)$ is a time-monotonous polyline in $\mathbb{R}^{1+d}$ with vertices $s_1, s_2, \ldots$.

$\vec{s}(t)$ generally deviates from $\vec{a}(t)$ due to sensor inaccuracies and time-discrete sensing. GRTS assumes a *maximum sensing deviation* $\delta(t)$ such that $|\vec{s}(t) - \vec{a}(t)| \leq \delta(t)$. It may vary over time, e.g. depending on the dilution of precision (DOP) values reported by the corresponding GPS receiver.

The simplified trajectory is a piecewise linear function $\vec{u}(t)$, analogous to $\vec{s}(t)$, with a reduced number of vertices $u_1, u_2, \ldots$. GRTS aims at minimizing the number of vertices

Published in Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops '10), pp. 829-831. Mannheim, Germany. March 2010.
© IEEE 2010
http://dx.doi.org/10.1109/PERCOMW.2010.5470553

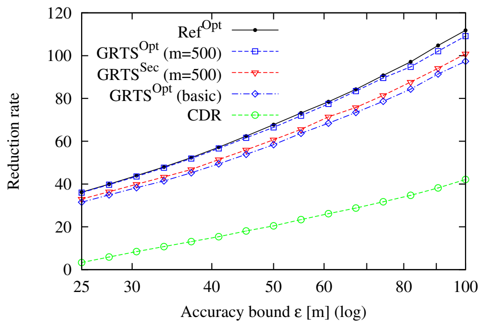
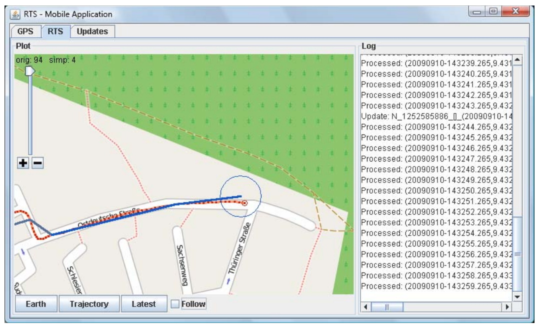


Figure 2. Reduction rate depending on ϵ.



```
Processed: (20090910-143239.265,9.431
Processed: (20090910-143240.265,9.431
Processed: (20090910-143241.265,9.431
Processed: (20090910-143242.265,9.431
Processed: (20090910-143243.265,9.432
Update: N_1252585886_[]_(20090910-14
Processed: (20090910-143244.265,9.432
Processed: (20090910-143245.265,9.432
Processed: (20090910-143246.265,9.432
Processed: (20090910-143247.265,9.432
Processed: (20090910-143248.265,9.432
Processed: (20090910-143249.265,9.432
Processed: (20090910-143250.265,9.432
Processed: (20090910-143251.265,9.432
Processed: (20090910-143252.265,9.432
Processed: (20090910-143253.265,9.432
Processed: (20090910-143254.265,9.432
Processed: (20090910-143255.265,9.432
Processed: (20090910-143256.265,9.432
Processed: (20090910-143257.265,9.432
Processed: (20090910-143258.265,9.433
Processed: (20090910-143259.265,9.433
```

Figure 3. Java Swing-based GUI.

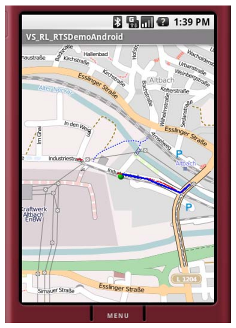


Figure 4. Android GUI.

of $\vec{u}(t)$ 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

$$\forall\, t \in [s_1.t, t_\mathrm{C}] \;:\; |\vec{u}(t) - \vec{a}(t)| \le \epsilon$$

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

For this purpose, the GRTS protocol combines *linear dead reckoning* (LDR) with line simplification. *LDR* is a simple but efficient protocol for tracking the current position of a moving object [3]. The object and the MOD share a linear prediction on the object's future movement given by a past sensed position and a velocity vector for the presumed direction and speed. Each time the deviation between the object's actual position and the predicted position *impends* to reach ϵ, the object transmits a new prediction to the MOD.

Line simplification refers to a multitude of algorithmic problems on approximating a given polyline by a simplified one with fewer vertices. Trajectory simplification belongs to the min-# problems of minimizing the number of vertices of the simplified polyline with respect to a given accuracy bound. There exist several algorithms for this problem such as the Douglas-Peucker algorithm [4], the Section Heuristic [1], and the optimal algorithm by Imai and Iri [2]. The latter reduces the min-# problem to the shortest-path problem.

GRTS combines LDR and line simplification as follows: The moving object maintains a *sensing history* $\mathbb{S}$ of the last sensed positions and adds each new sensed position to it. In the present version of GRTS, the size $|\mathbb{S}|$ is bounded by a parameter m, in contrast to [1]. Each time $|\mathbb{S}|$ exceeds m, an arbitrary predetermined line simplification algorithm is executed on $\mathbb{S}$, resulting in a simplified trajectory segment with vertices $\mathbb{U} \subseteq \mathbb{S}$. The first simplified line section, given by the first two vertices of $\mathbb{U}$, is stored for the next update message of LDR and $\mathbb{S}$ is cleared up to the second vertex of $\mathbb{U}$. If the current sensed position causes LDR to send an update, three steps are performed: First, a simplification $\mathbb{U}$ for $\mathbb{S}$ is computed and completely added to the update message. Then, a new velocity vector for LDR is determined based on the last sensed positions and added to the update message. Finally, the update message is sent to the MOD.

The simplified trajectory $\vec{u}(t)$ is divided into three parts, cf. Figure 5. The first part is created in an append-only fashion and stored at the MOD only. The third part is given by the prediction of LDR. The second part is the simplification $\mathbb{U}$ of $\mathbb{S}$ at the time of the last update message.

With the above algorithm, GRTS adds a vertex to $\vec{u}(t)$ at least every m sensed positions, even if the object does not move for a long time. Therefore, we extended GRTS by a compression approach for $\mathbb{S}$.

Figure 2 shows the average reduction rate of different simplification algorithms depending on ϵ for more than 250 GPS traces from the OpenStreetMap project [5]. Ref$^\mathrm{Opt}$ denotes the maximum possible reduction computed *offline* by the optimal line simplification by Imai and Iri [2]. GRTS$^\mathrm{Sec}$ denotes GRTS combined with the Section Heuristic described in [1]. GRTS$^\mathrm{Opt}$ refers to GRTS combined with the algorithm by Imai and Iri. The label *basic* refers to the variant of GRTS described in [1], whereas $m = 500$ denotes the improved version described above. The improved version outperforms the basic variant of GRTS$^\mathrm{Opt}$, even with the simple Section Heuristic, despite the limitation of $\mathbb{S}$.

III. GRTS-based MOD System Prototype

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. Next, we describe each of these components.

Mobile component is executed by each object being tracked using a subnotebook or smartphone with a GPS receiver.

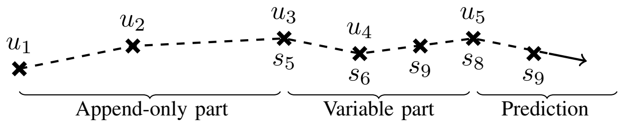


Figure 5. The three parts of $\vec{u}(t)$.

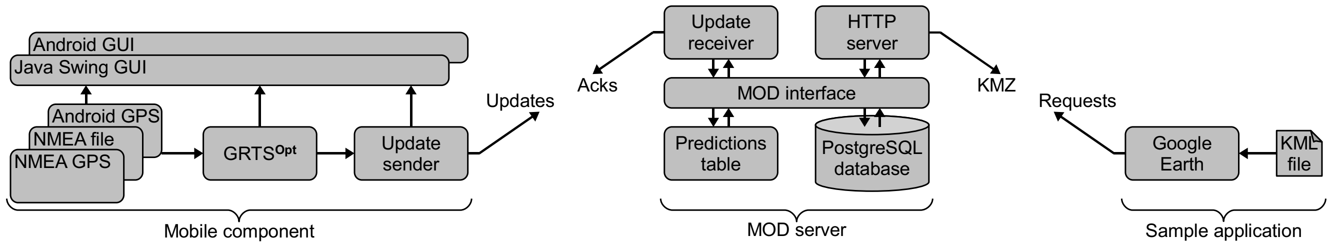


Figure 6. System architecture.

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^{Opt} is an implementation of the GRTS algorithm with the optimal line simplification algorithm by Imai and Iri [1].

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. For clock synchronization between the moving object and the MOD server, the sent time and previous round trip times are included into each message.

To illustrate the functioning of GRTS, we implemented two GUIs for different devices: Figure 3 is a screenshot of the Java Swing-based GUI. It shows sensed and the simplified trajectory drawn on the map of the OpenStreetMap project [5]. The large circle depicts the accuracy bound ϵ. The small circle illustrates the maximum sensing deviation $\delta(t)$. Figure 4 analogously is 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.

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 that executes 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.

IV. Demonstrated Functionality

The complete MOD system is demonstrated using a laptop executing the MOD server and Google Earth and several smartphones and a subnotebook executing the mobile component. NMEA file is used to replay different NMEA traces from bicycle tours around Stuttgart, Germany.

The map-based visualization on the mobile devices allows to follow the GRTS algorithm and protocol and to test different parametrizations of GRTS. Moreover, by comparing the trajectories shown on the mobile devices with the trajectories shown in Google Earth, it is possible to visually verify that the simplification is performed in real-time.

V. Summary

In this demo paper, we presented an implementation of the GRTS protocol [1], which optimizes storage consumption, processing, and communication costs for tracking moving objects' trajectories. In terms of reduction performance, GRTS achieves up to 98% compared to retrospective offline reduction with the optimal line simplification algorithm.

Acknowledgments

The work described in this paper was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center (SFB) 627.

References

[1] R. Lange, T. Farrell, F. Dürr, and K. Rothermel, "Remote Real-Time Trajectory Simplification," in *Proc. of 7th PerCom*, Galveston, TX, Mar. 2009.

[2] M. Iri and H. Imai, *Computational Morphology*. North-Holland Publishing Company, 1988, ch. Polygonal Approximations of a Curve – Formulations and Algorithms, pp. 71–86.

[3] A. Leonhardi and K. Rothermel, "A Comparison of Protocols for Updating Location Information," *Cluster Computing*, vol. 4, no. 4, pp. 355–367, Oct. 2001.

[4] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Canadian Cartographer*, vol. 10, no. 2, pp. 112–122, Dec. 1973.

[5] http://www.openstreetmap.org/.