ADS-BI: Compressed Indexing of ADS-B Data

Sebastian Wandelt, Xiaoqian Sun, and Hartmut Fricke

***Abstract*— The introduction of ADS-B, a satellite-based aircraft tracking technology, and the increasing installation of ADS-B receiver stations around the globe eases the tracking of aircraft, compared with traditional solutions using secondary radar. Given the large scale of ADS-B implementation and the high frequency of data collection, storing and managing ADS-B induced data has become increasingly difficult: The worldwide ADS-B data easily aggregates to several hundreds of terabyte per year, depending on the spatial coverage and temporal resolution. Standard data management solutions do not work well for ADS-B data, since they either require a large uncompressed index structure or cannot be queried efficiently. In this paper, we propose a novel compressed index structure for managing ADS-B data, called ADS-BI. The essential building blocks are spatio-temporal reference partitioning, reordering, and compression. On top of the partitioned, compressed representation, metadata is stored effectively, and exploited during query answering for typical ATM related task such as trajectory adherence evaluation, as well as complexity and safety metrics assessment by only accessing parts of the compressed data as necessary. Our novel index structure is evaluated on worldwide ADS-B data for a week in November 2016. For comparison, we implemented ten standard compression/indexing methods. The experiments reveal that none of these traditional methods can target the sweet spot between a small storage and efficient query answering. Our novel technique provides fast query answering at smallest storage costs. This paper contributes toward efficient handling of the increasing amount of traffic data in air traffic management, and eventually, toward more efficient and safer air transportation.**

***Index Terms*— ADS-B, data compression, data management, aircraft trajectories.**

I. Introduction

WORLDWIDE air transportation is facing an increasing demand for efficiency and safety. Accordingly, a multitude of analysis tasks has emerged, most of which rely on the processing of aircraft positions over time, e.g., complexity science analysis [1], [2], delay propagation [3], conflict resolution [4], topological analysis [5], and general data mining [6]. Analyzing such data often requires large coverage and also snapshots at high resolution. Automatic Dependent Surveillance - Broadcast (ADS-B), an integral component of the Next Generation Air Transportation System (NextGen) and the Single European Sky ATM Research (SESAR) frameworks, was designed in order to overcome this problem. Accordingly, the introduction of ADS-B throughout the last decade, has made it possible for air traffic controllers and researchers to collect large amounts of air traffic data at very high coverage and frequency.

While the increasing amount of data provides exciting opportunities, a critical challenge is to manage the data. At such a large scale of dozens of terabytes, the storage of data alone has already become a problem. However, even more important, efficient query answering, over such data is tremendously complex [7]–[9]: Despite improved hard-/software, the computational requirements for storage and analysis of the bulk of data are steeply increasing. Compression is one key technology to deal with the increasing amount of ADS-B data. Traditional indexing systems, such as relational databases, column stores, and alike, have in common that the index size is usually larger than the uncompressed data. This makes it difficult to employ these systems at a larger scale. Lossless compression [10], on the other hand, which has the goal to reduce the amount of data by exploiting data regularities and similarities, comes usually at the price of non-trivial and slow query answering.

In this paper, we develop ADS-BI, an indexing technique for ADS-B data which aims at a sweet spot between efficient query answering (as provided by indexed systems) and small storage (as provided by compression); the trade-off between both is depicted in Figure 1. Our technique is based on

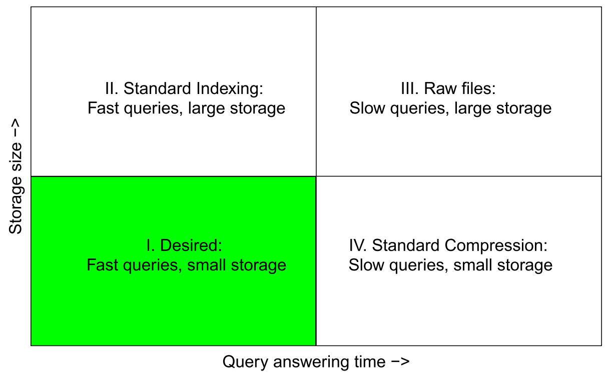

II. Standard Indexing:
Fast queries, large storage
III. Raw files:
Slow queries, large storage
I. Desired:
Fast queries, small storage
IV. Standard Compression:
Slow queries, small storage
Storage size ->
Query answering time ->

Fig. 1. Visualization of trade-off between query answering time (x-axis) and storage size (y-axis). In general, the minimization of both at the same time is desired (Quadrant I of the diagram). Existing solutions only target either of the two goals: Fast retrieval using indexes (Quadrant II) or reducing storage with compression (Quadrant IV).

Manuscript received March 14, 2017; revised November 5, 2017; accepted December 4, 2017. Date of publication February 6, 2018; date of current version November 27, 2018. This work was supported by the Research Fund from National Natural Science Foundation of China under Grant 61601013 and Grant 61650110516. The Associate Editor for this paper was Z. Mao. *(Corresponding author: Xiaoqian Sun.)*

S. Wandelt and X. Sun are with the School of Electronic and Information Engineering, Beihang University, Beijing 100191, China, and also with the Beijing Key Laboratory for Network-Based Cooperative ATM, Beijing 100191, China (e-mail: wandelt@informatik.hu-berlin.de; sunxq@buaa.edu.cn).

H. Fricke is with TU Dresden, 01069 Dresden, Germany (e-mail: hartmut.fricke@tu-dresden.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2017.2784371

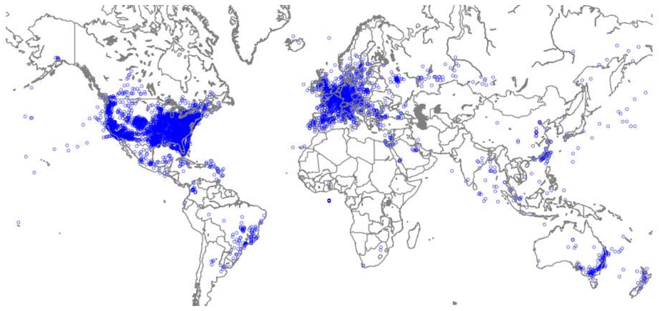


Fig. 2. Snapshot of the data from http://adsbexchange.com for December, 22nd 2016 at 08:38PM. Blue circles represent aircraft positions. In total, 7,173 aircraft were recorded at that time, mostly over US and Europe. Visualization with Matplotlib.

the idea to partition an ADS-B database into a collection of spatio-temporal blocks, which allow us to selectively load relevant blocks during query answering. We develop an efficient column-based compression technique, which significantly reduces the amount of storage per block by exploiting similarities in the input data. Moreover, we design a reference-based partitioning system, which allows us to reuse and exploit frequently reoccurring partition patterns, in order to further reduce the index size and improve query answering time. We compare our approach with ten standard techniques for storage and indexing. Our experimental evaluation shows that ADS-BI is the only technique that provides fast query answering with a small compressed index. Our work contributes towards scalable data management of ADS-B data.

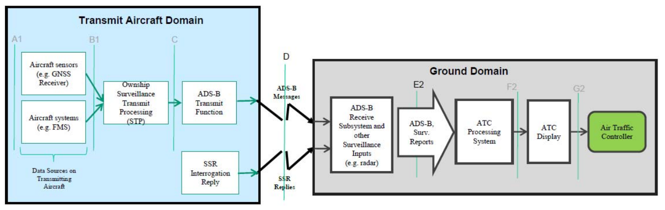
The remainder of our paper is organized as follows. We discuss a data model for ADS-B, together with a set of relevant queries, in Section II. Section III discusses the state of the art in data indexing and the selection of a set of competitors. We introduce our novel indexing technique ADS-BI in Section IV and evaluate it with real-world ADS-B data against competitors in Section V. The paper concludes with Section VI.

II. Data Model and Queries

In Section II-A, we formalize the notion of an ADS-B database as necessary for the definition and implementation of our indexing technique, based on typical Air Traffic Management (ATM), communication, navigation and surveillance (CNS) needs. Referring to the database definition we introduce a set of suitable example queries in Section II-B.

A. ADS-B Database

A database for *ADS-B out* data [EASA CS-ACNS] manages a set of aircraft identification, horizontal position, barometric altitude and quality indicators over time. In Figure 2, we show a snapshot of the worldwide ADS-B data obtained from http://adsbexchange.com. Please note that the majority of ADS-B data position reports is located over Europe and North America, given the current absence of VHF receiver stations in other regions of the world. Once the ADS-B network is fully developed and installed, the system will track flights worldwide, enriching todays secondary surveillance


Transmit Aircraft Domain
A1
B1
C
Aircraft sensors (e.g. GNSS Receiver)
Aircraft systems (e.g. FMS)
Ownship Surveillance Transmit Processing (STP)
ADS-B Transmit Function
SSR Interrogation Reply
Data Sources on Transmitting Aircraft
D
ADS-B Messages
SSR Replies
Ground Domain
E2
F2
G2
ADS-B Receive Subsystem and other Surveillance Inputs (e.g. radar)
ADS-B, Surv. Reports
ATC Processing System
ATC Display
Air Traffic Controller
Fig. 3. Dataflow of combined SSR / ADS-B position reporting for ATC application [source: http://dfs.de].

radar track systems (see Figure 3): Based on the data representation, we derive the following definition.

Definition 1: We define an *ADS-B data point* as a tuple $p = \langle t, lon, lat, fl, ac, s_1, \ldots, s_n\rangle$ which encodes the position longitude (lon) and latitude (lat) and barometric altitude (fl) of an aircraft with unique ID ac at time t. The fields $s_1, \ldots, s_n$ represent additional information about the aircraft/flight. For the remainder of this study, we assign two fields with additional information as follows: The origin airport s_1 of the flight and the destination airport s_2. We refer to the x-component of an ADS-B data point p with $p.x$. A sequence of ADS-B data points $db = [p_1, \ldots, p_m]$ is called an *ADS-B database* (short *ADS-BDB*). We continue with an example for an ADS-BDB as follows.

Example 1: An example ADS-BDB with 30 points is $D = \{p_0, ..p_{29}\}$, where

$$
\begin{aligned}
p0 &= \langle 1, 0.5, 0.5, 9, a1, c1, c2\rangle,\\
p1 &= \langle 1, 9.0, 1.0, 0, a2, c4, c5\rangle,\\
p2 &= \langle 1, 3.0, 7.0, 9, a3, c3, c6\rangle,\\
p3 &= \langle 1, 6.0, 2.0, 0, a4, c8, c9\rangle,\\
p4 &= \langle 1, 9.0, 9.0, 0, a5, c5, c4\rangle,\\
p5 &= \langle 2, 1.0, 1.0, 9, a1, c1, c2\rangle,\\
p6 &= \langle 2, 9.0, 2.0, 3, a2, c4, c5\rangle,\\
p7 &= \langle 2, 4.0, 6.0, 4, a3, c3, c6\rangle,\\
p8 &= \langle 2, 6.1, 3.0, 2, a4, c8, c9\rangle,\\
p9 &= \langle 2, 9.0, 8.0, 3, a5, c5, c4\rangle,\\
p10 &= \langle 3, 1.5, 1.5, 5, a1, c1, c2\rangle,\\
p11 &= \langle 3, 9.0, 4.0, 9, a2, c4, c5\rangle,\\
p12 &= \langle 3, 5.0, 5.0, 0, a3, c3, c6\rangle,\\
p13 &= \langle 3, 6.1, 4.0, 8, a4, c8, c9\rangle,\\
p14 &= \langle 3, 8.8, 6.0, 9, a5, c5, c4\rangle,\\
p15 &= \langle 4, 2.0, 2.0, 3, a1, c1, c2\rangle,\\
p16 &= \langle 4, 9.0, 6.0, 9, a2, c4, c5\rangle,\\
p17 &= \langle 4, 5.0, 5.0, 0, a3, c6, c7\rangle,\\
p18 &= \langle 4, 6.2, 5.0, 9, a4, c8, c9\rangle,\\
p19 &= \langle 4, 8.8, 4.0, 9, a5, c5, c4\rangle,\\
p20 &= \langle 5, 2.4, 1.4, 1, a1, c1, c2\rangle,\\
p21 &= \langle 5, 9.0, 8.0, 3, a2, c4, c5\rangle,\\
p22 &= \langle 5, 4.0, 4.0, 3, a3, c6, c7\rangle,\\
p23 &= \langle 5, 6.3, 7.0, 9, a4, c8, c9\rangle,\\
p24 &= \langle 5, 9.0, 2.0, 3, a5, c5, c4\rangle,
\end{aligned}
$$

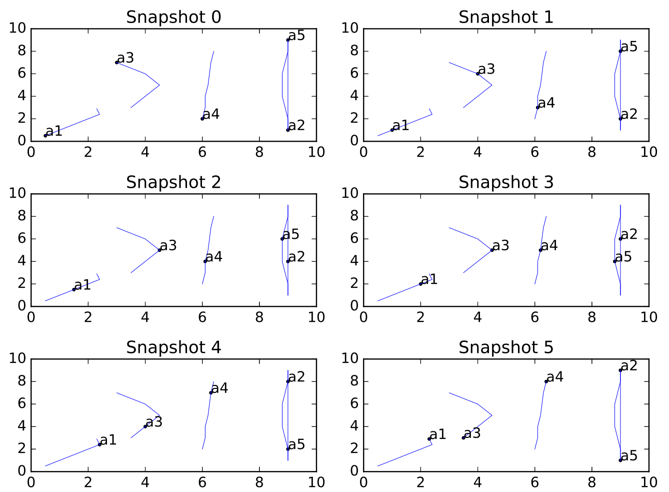


Fig. 4. Visualization of six snapshots from Example 1. Dots represent aircraft and thin blue lines represent their trajectories over time.

$$p25 = \langle 6, 2.5, 1.5, 0, a1, c1, c2\rangle,$$
$$p26 = \langle 6, 9.0, 9.0, 0, a2, c4, c5\rangle,$$
$$p27 = \langle 6, 3.0, 3.0, 9, a3, c6, c7\rangle,$$
$$p28 = \langle 6, 6.4, 8.0, 9, a4, c8, c9\rangle,$$
$$p29 = \langle 6, 9.0, 1.0, 0, a5, c5, c4\rangle$$

This example database represents trajectory information about five aircraft at six consecutive timestamps; a visualization can be seen in Figure 4. It should be noted that real ADS-BDBs may contain far more data fields per aircraft, e.g., the 24-bit aircraft address or a special position indicator (SPI) for navigation accuracy verification or using the data for In Trail Procedures (ITP) [11], following the 1090 MHz Mode S Extender Squitter data link standard, jointly developed by EUROCAE and RTCA. The aircraft positions in space and time are visualized as distinct snapshots in Figure 5. We will use the ADS-BDB from Example 1 as a running example for explaining the intuition of indexing techniques and query answering below.

B. Queries

In the following we define and revisit a collection of typical queries to ADS-B databases, in general aiming at providing interoperable aircraft surveillance data to Air Traffic Control or support non-verbal Controller-Pilot Data Link Communication (CPDLC). These queries are required for visualization purposes and performing the above control and communication tasks, extending to statistical sector complexity evaluation, delay identification, and many more.

Query 1 (Snapshot retrieval):
The first query, a simple, yet one of the most frequently used queries in flight tracking systems, is the retrieval of a temporal snapshot: Given a time snapshot, compute the set of all ADS-B points which belong to that time. The major use case of this query is the input to aircraft situation displays or air traffic flow management (ATFM) benchmark tools [12], [13]. This query only accesses a very limited

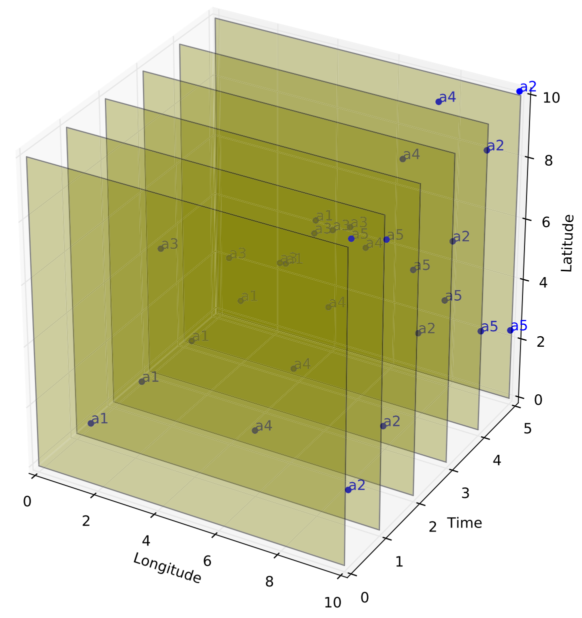


Fig. 5. Three-dimensional layer visualization of the six snapshots from Example 1. Dots represent aircraft positions at a given time.

part of the ADS-B point database, if properly indexed and points are temporally grouped.

A snapshot query for $t = 4$ on Example 1 yields the following five entries:

$$p15 = \langle 4, 2.0, 2.0, 3, a1, c1, c2\rangle,$$
$$p16 = \langle 4, 9.0, 6.0, 9, a2, c4, c5\rangle,$$
$$p17 = \langle 4, 4.5, 5.0, 0, a3, c6, c7\rangle,$$
$$p18 = \langle 4, 6.2, 5.0, 9, a4, c8, c9\rangle,$$
$$p19 = \langle 4, 8.8, 4.0, 9, a5, c5, c4\rangle$$

Query 2 (Bounding box retrieval):
A generalization of Query 1 is the computation of all points inside a bounding box, spanned up by one temporal and two spatial intervals. Such information is useful for visualization of a regional snapshot or evaluation of sector complexity, starting with simple metrics like number of aircraft travelling the sector at time t up to e.g., the sector complexity C5 metric. This metric is based on the mean weighted horizontal separation distance [14].

$$C_5 = \frac{N}{\sum_{1\leq j\leq N}(\frac{\sum_{1\leq j\leq N} w_{ij} * d_{ij}}{\sum_{1\leq j\leq N} w_{ij}})}, \tag{1}$$

where N is the number of aircraft within the sector airspace, d_{ij} is the horizontal separation distance between two aircraft i and j (in nautical miles) and W_{ij} is the associated weighting factor.

This query again needs only access to a well limited part of the ADS-B-DB. However, compared to Query 1, more sophisticated indexing techniques are required to keep query answering time low, since we now apply multiple selection criteria; and grouping data by snapshots is not efficient for small regions and larger temporal intervals. A bounding box query with $lon = [0, 4]$, $lat = [0, 4]$, and $t = [5, 6]$ on Example 1 yields the following four entries:

$$p20 = \langle 5, 2.4, 2.4, 1, a1, c1, c2\rangle,$$
$$p22 = \langle 5, 4.0, 4.0, 3, a3, c6, c7\rangle,$$
$$p25 = \langle 6, 2.3, 2.9, 0, a1, c1, c2\rangle,$$
$$p27 = \langle 6, 3.5, 3.0, 9, a3, c6, c7\rangle$$

Query 3 (Aircraft retrieval):
This query retrieves all ADS-B data points for a given aircraft (identified by ICAO[1] ID), without any temporal boundaries. This query is valuable, for instance, to identify airline network driven routing strategies serving different airports with one aircraft over time, or to compute the leg-wise and reactionary delay of the same aircraft over time compared to the schedule. Furthermore, it is an important metric to judge economic or ecological optimality for a given flight as the real flown trajectory should be used to determine those often competing costs [15]–[18]. Finally, those queries lay the ground for determining and verifying the actual navigation performance (ANP) of aircraft flying dedicated Area Navigation (RNAV) routes requiring least minimum ANP.
Those data is used to model aircraft collision risk in en-route or approach areas and as such are important metric in safety research [19]. In order to answer this query, the database needs to be scanned entirely. Even if an index exists on the property ICAO ID, we end up with random access over a very large database. While we could group ADS-B data points by their ICAO ID, this would tremendously deteriorate query answering times for spatial and temporal queries. An aircraft retrieval query with $ac = a4$ on Example 1 yields the following entries:

$$p3 = \langle 1, 6.0, 2.0, 0, a4, c8, c9\rangle,$$
$$p8 = \langle 2, 6.1, 3.0, 2, a4, c8, c9\rangle,$$
$$p13 = \langle 3, 6.1, 4.0, 8, a4, c8, c9\rangle,$$
$$p18 = \langle 4, 6.2, 5.0, 9, a4, c8, c9\rangle,$$
$$p23 = \langle 5, 6.3, 7.0, 9, a4, c8, c9\rangle,$$
$$p28 = \langle 6, 6.4, 8.0, 9, a4, c8, c9\rangle$$

Query 4 (OD pair retrieval):
This query asks for all ADS-B data points for a given pair of Origin and Destination (OD) airports in the entire database. This query is crucial to e.g., determine the efficiency of airline networks in terms of in-time / delayed services, connection robustness and fleet utilization [20]. An OD pair retrieval query with $origin = c1$ and $destination = c2$ on Example 1 yields the following entries:

$$p0 = \langle 1, 0.5, 0.5, 9, a1, c1, c2\rangle,$$
$$p5 = \langle 2, 1.0, 1.0, 9, a1, c1, c2\rangle,$$
$$p10 = \langle 3, 1.5, 1.5, 5, a1, c1, c2\rangle,$$
$$p15 = \langle 4, 2.0, 2.0, 3, a1, c1, c2\rangle,$$
$$p20 = \langle 5, 2.4, 2.4, 1, a1, c1, c2\rangle,$$
$$p25 = \langle 6, 2.3, 2.9, 0, a1, c1, c2\rangle$$

[1]http://www.icao.int

III. Standard Solutions

Research on storage and indexing techniques for ADS-B data is rare so far, but problems in similar domains have led to the development of standard data management techniques, we briefly review in the context of our ADS-BDBs in this section. Particularly, we review techniques based on snapshot storage without indexing (Section III-A), database storage (Section III-B), column storage (Section III-C), and spatial point-based indexing (Section III-D). Moreover, other related works are summarized in Section III-E. Based on the discussion of standard solution techniques, we derive a set of ten competitors for our comparative evaluation in Section V.

A. Snapshot Storage

ADS-B data can be naturally grouped by snapshots, for instance, at a resolution of one second (refer to Figure 5 for Example 1). In this case, a single document contains all information about one second; For each second a new document is created. For serialization of data, we can choose different data models. Here, we only discuss the most prevalent one: Relational storage. Storage as relational data has long traditions in many application domains. Intuitively, the representation from Example 1 can be directly translated into a relational snapshot representation by modeling each ADS-B point as a row and columns are possible attributes of points. When answering queries over such a snapshot collection, some queries are rather simple to answer, e.g. those only involving constraints with tight temporal bounds, while others require to parse and scan the whole collection of snapshots. In our study, we consider three approaches for storing snapshot collections:

- **CSV**: Each snapshot is represented as a single Comma-Separated Value (CSV[2]) file.
- **CSVZIP**: Based on CSV, each CSV file in the collection is compressed using ZIP.
- **CSVLZMA**: Based on CSV, each CSV file in the collection is compressed using LZMA.

Expected results: The size using CSVZIP/CSVLZMA should be significantly smaller than using CSV. Query 1 can be executed fast with all these three competitors; all other queries, however, are rather slow due to lack of indexing.

B. Relational Database Storage

Given the relational representation of an ADS-BDB, we can easily create a relational database over all snapshots together. Here, a single table is created, which contains one column for each attribute in the database. The advantage of storing the data in a database system is the ease of accessibility by sophisticated queries, formulated in SQL or variants, and the possibility to create standard indexes on selected parts of the database. When answering queries with a database index, many queries can be efficiently answered. In our study, we consider three approaches for storing in a database:

- **SQLITE3**: Storing all points in a standard SQLite [21] database.

[2]https://tools.ietf.org/html/rfc4180

- **SQLITE3I**: As SQLITE3, but with maximum indexing on all columns.
- **TinyDB**: Storing all points as key-value pairs in the database TinyDB.

Expected results: The size of all three approaches should be larger than CSV. However, query answering should be much faster on all queries; most notably Query 3–4, given the indexes on attributes.

C. Column Store

The general idea of column-wise data processing has long traditions in the database community [22]; yet systems like MonetDB [23] pioneered its usage in modern column-oriented database systems only recently [24]. In our study, we consider the storage of an ADS-BDB using a column store as well:

- **MonetDB**: Storing all points in a standard MonetDB database.

Expected results: The storage size using MonetDB should be similar to that of SQLITE3; and fast query answering for most queries. It should be noted that MonetDB does not have any traditional indexes on the columns, but creates information about columns on the fly during query answering.

D. Point-Based Indexing

Indexing a collection of points is a common task in many Geographical Information System (GIS) applications, and several data structures for efficient spatial indexing have been developed in the past, including Rtrees [25], Quadtrees [26], and BKD trees [27]. In general, the idea is to split multiple dimensions (the plane in case of spatial data) into items of equal frequency, in order to provide efficient search over items. In our study, we consider the following implementation:

- **RTree**: Each ADS-B point is represented as an element in a RTree.

Expected results: While RTrees are superb for spatial indexing, Query 3–4 could be significantly slower. The size of the RTree structure is larger than that of CSV.

E. Others

Previous work on 3D-trajectories, obtained from location-based services on mobile phones, shows that trajectories can be compressed by a factor of 100 or more, if lossy compression is acceptable, by restricting the precision or recall to user-defined error bounds [28]–[30] (see [31], [32] for recent reviews). These approaches have in common that they either discard a significant amount of precision in order to reduce the amount of data [28], [32], or focus on indexing without any attempts for compression [29], [30]. Other work on data compression has shown that the column-wise compression of relational data can significantly reduce the amount of required storage [33], [34]; however, indexing of data is largely ignored, which means that upon access to the data, the whole dataset needs to be decompressed, which is unacceptable. In our study, we consider the following implementations, inspired by [33]:

- **CCZIP**: Each column is compressed separately using ZIP.
- **CCLZMA**: Each column is compressed separately using LZMA.

Expected results: Both competitors, CCZIP and CCLZMA, should require significantly less storage than CSV. For answering all types of queries, it is necessary to decompress the whole dataset, which is expected to be slow.

IV. Novel Approach

Traditional techniques for storing and indexing ADS-B data have disparate foundations: Either they 1) provide efficient access to the data by computing additional index structures and rearrange the data or 2) reduce the amount of storage required at the price of inefficient access to the compressed data. In this paper, we develop a novel indexing technique, ADS-BI, which targets to combine the best of both worlds, compression together with indexing.

A. Overview

Traditional snapshot storage has the advantage of allowing quick access to single snapshots, while column-wise storage of data allows for high compression rates (see our evaluation in Section V). Based on this observation, we propose to split an ADS-BDB into temporal-spatial blocks, similar to R-trees, aiming at a sweet spot for compression by exploiting data locality. Our technique for compression inside blocks essentially follows the idea of column-wise compression (Section IV-B). The partitions are not created uniformly, but depending on the distribution of the data. As we show below, the spatial distribution of data points highly varies even throughout a single day. A uniform partitioning of data will neglect ample compression/load balancing potential. Therefore, we propose a novel adaptive partitioning method. In order to reduce the overhead of managing too many partitions, we introduce a set of principle partition references, which are updated throughout the index creation stage (Section IV-C). Moreover, these partition references enable very efficient pre-filtering during query answering time at rather low additional storage costs. During the time of query answering, based on our block-based storage together with column-compression, we essentially only access what is really required for retrieval of all results (Section IV-D).

B. Spatio-Temporal ADS-B Blocks

We propose to store an ADS-BDB as a set of spatial-temporal blocks whose entries are tightly bounded. On one hand, this helps us to significantly speed up query answering times, because we can selectively access blocks which are identified as relevant for a query. On the other hand, we show that the careful partitioning of ADS-B data points into blocks allows us to exploit ample compression potential, compared to pure snapshot storage and uncompressed RTrees. We define a block as a sequence of ADS-B data points within temporal and spatial bounds as follows:

Definition 2: A sequence of ADS-B data points $tr = [p_1, \ldots, p_m]$ is called an *ADS-B block* if for all entries we have that $lon_1 \leq p_i.lon < lon_2$, $lat_1 \leq p_i.lat < lat_2$, and $t_1 \leq p_i.t < t_2$, and all entries are sorted increasingly first by ac and afterwards by t.

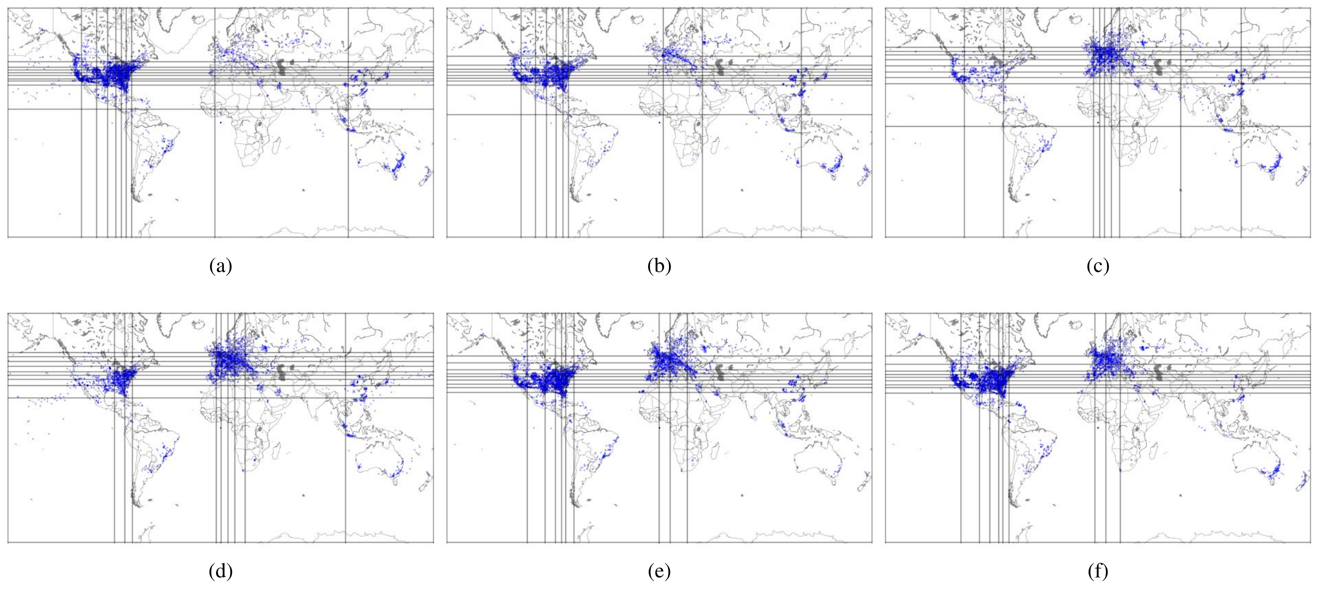


Fig. 6. Visualization of equal-depth clustering. For each of the six snapshots, we obtain a list of latitude and longitude coordinates from the aircraft positions. Both arrays are sorted in an ascending order and the latitude/longitude values are extracted at nine equally distributed positions. These nine latitude/longitude values are then used to split the plane vertically/horizontally into ten rows/columns, yielding 100 blocks. Times are at UTC (Coordinated Universal Time). (a) December 22nd, 2016, 00:00. (b) December 22nd, 2016, 04:00. (c) December 22nd, 2016, 08:00. (d) December 22nd, 2016, 12:00. (e) December 22nd, 2016, 16:00. (f) December 22nd, 2016, 20:00.

Example 2: An ADS-B block for Example 1 and the $lon = [0, 4]$, $lat = [0, 4]$, and $t = [1, 3]$ on Example 1 yields the following entries:

$$p1 = \langle 1, 2, 2, 4, a1, c1, c2\rangle, \quad p3 = \langle 2, 3, 3, 0, a1, c1, c2\rangle$$
$$p5 = \langle 3, 1, 3, 5, a2, c3, c2\rangle$$

From Example 2, we can see that entries for an identical flight often end up grouped together. We can exploit this fact for compression: Many compression techniques work best, if they can exploit locality, i.e., if many repetitions are found within a small sliding window. Based on the findings in previous work, which proposes to compress columns of trajectories separately for index-less storage [33], we compress a block column-wise. The reordering is performed before compression, in order to keep highly similar trajectory points together. Continuing with Example 2, after reordering, we obtain a collection of seven columns, as follows:

Example 3: The ADS-B block from Example 2 yields the following seven column streams.

$$col_t = [1, 2, 3], \quad col_{lon} = [2, 3, 1], \quad col_{lat} = [2, 3, 3],$$
$$col_{fl} = [4, 0, 5], \quad col_{ac} = [a1, a1, a2],$$
$$col_{from} = [c1, c1, c3], \quad col_{to} = [c2, c2, c2]$$

Such columns are compressed efficiently with the standard compressor LZMA, given their similar contexts and repetitive values often occur close to each other (as forced by the reordering). In Section IV-C, we describe how we actually create the block boundaries for an ADS-BDB. So far, we only addressed the compressed storage, but neglected the indexing capabilities. Therefore, below we also show how the partitioning into spatio-temporal blocks further improves query answering times significantly.

C. Partitioning

Given the advantages of splitting an ADS-BDB into spatio-temporal blocks, we discuss the details of how to derive a partitioning of all points into these blocks next. Essentially, we need to deal with three dimensions: Latitude, longitude, and time. For the dimension time we chose the simple solution of splitting an ADS-BDB into equally spaced intervals of a predefined number of minutes. The rationale is that, according to our experiments in Section V, the number of entries is rather constant over time. For now, we assume that we have split an ADS-BDB into temporal intervals of T minutes and deal with each interval separately.

The spatial distribution of entries in a temporal interval changes significantly throughout the course of a day, as well as, over longer time. The former is caused by peaks of flights during day times and night flight bans, while the latter is induced by an ever growing travel demand and an improved coverage of regions with ADS-B receivers. In Figure 6, we visualize the distribution of points throughout a single day at a resolution of four hours. It can be seen that major locations of aircraft vary significantly: At night time (UTC), the majority of flights is found over Northern America, while at day time the mass of flights shifts towards Europe. With this observation, we conclude that there exists no best static partitioning into blocks; but we need to adapt the partitioning over time, in order to obtain a reasonable effect of load balancing.

We compute the spatial partitioning for a given set of points as follows. First, we create a separate list of all latitude (longitude) entries for the set of points. This list is sorted in an ascending order and the latitude (longitude) values are extracted at $X - 1$ equally distributed indexes. These $X - 1$ latitude/longitude values are then used to split the plane

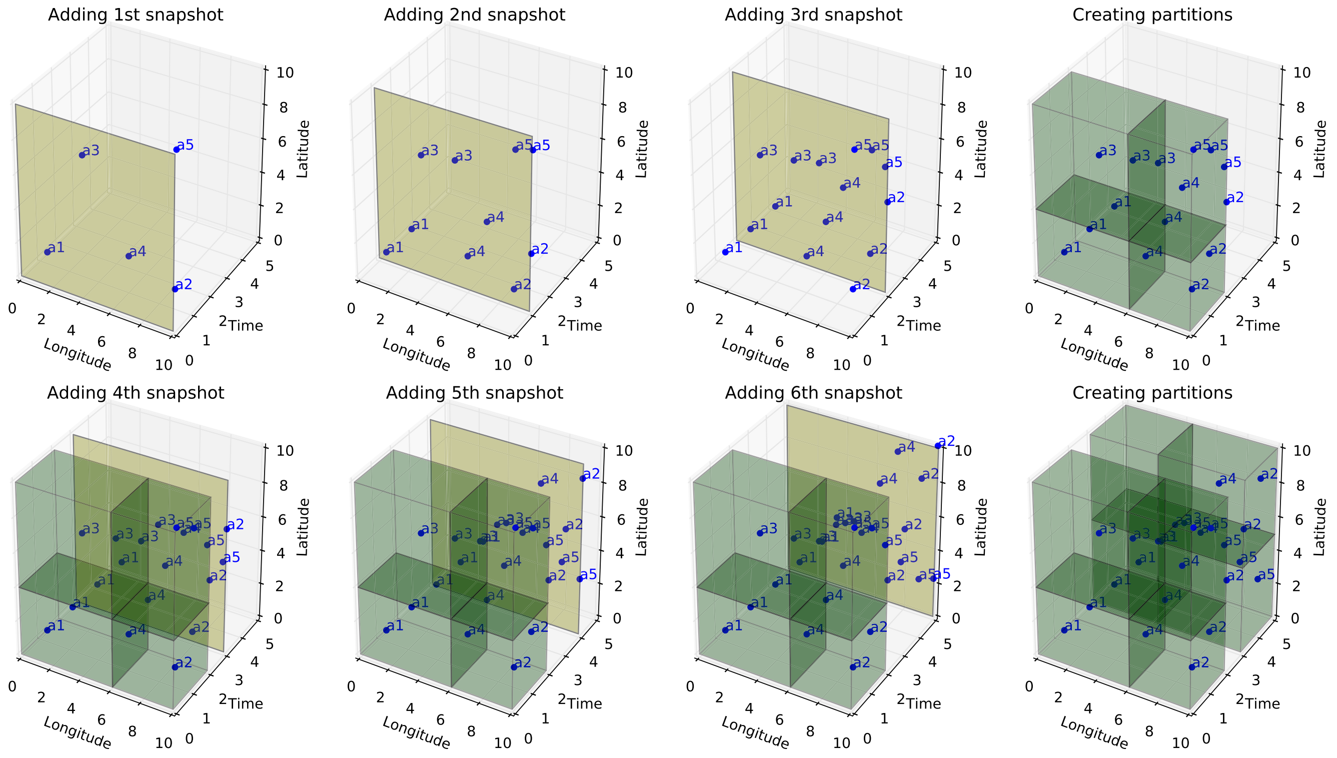


Fig. 7. Visualization of index creation for the six snapshots from Example 1. Dots represent aircraft, planes visualize snapshots and boxes model spatio-temporal aggregations.

vertically/horizontally into X rows/columns, yielding $X * X$ blocks.[3]

The naive approach is to compute a new such partitioning for each temporal interval in an ADS-BDB. Such a creation of a new partitioning every few minutes creates an undesirable property for indexing: Given the changing partitioning structure over time, it becomes increasingly difficult to lookup points in longer time intervals, because we need to treat each temporal interval separately. Moreover, as we show below, it is beneficial to attach metadata to a partitioning in order to reduce query answering times. Such metadata requires additional storage, and thus we would like to avoid repetitive metadata, to limit hampering the effectiveness of our compression technique. Therefore, we introduce the concept of a reference partitioning. Essentially we keep track of each partitioning obtained for previous temporal intervals and check the similarity of the currently obtained partitioning. In order to reduce computation time, we design an easy-to-compute heuristic for estimating the similarity: Pairwise sum of differences for the list of latitude (longitude) breakpoints. If the sum is smaller than a given threshold, we use the existing partitioning; otherwise a new partitioning is created. In Section V, we explore the effect of different similarity thresholds and give a recommendation on which one to use.

[3]In fact, an additional block is derived for entries without latitude/longitude information and stored separately. For the sake of simplicity, we neglect this additional block in our remaining description. All experiments, were implemented with this additional block for entries without spatial information, to correctly reflect all entries in an ADS-BDB.

In Figure 7, we visualize the process of creating the partitions of the database from Example 1.

In order to efficiently answer queries over ADS-BDB blocks, we propose to store metadata for blocks. Without such metadata, we would have to decompress and scan all blocks during query answering; a process which does not scale up well for larger ADS-BDBs. Therefore, we attach to each partition reference metadata which can help to decide quickly which blocks, using that reference, is relevant for answering a query. Besides the latitude/longitude breaks inducing the partition, we keep track of the following:

- A mapping from Aircraft ID (issued by ICAO) to reference blocks in which this aircraft has been seen.
- A mapping from origin airport to reference blocks in which an aircraft with this origin has been seen.
- A mapping from destination airport to reference blocks in which an aircraft with this destination has been seen.

D. Query Answering

Below, we show how this metadata is used for efficient query answering over the compressed point database; by discussing each query type separately.

Query 1 (Snapshot retrieval): First, we select all boxes that intersect the snapshot plane, defined by a temporal point. For each column selected, we decompress the result-relevant columns of the box and stream results.

Query 2 (Bounding box retrieval): Similar to answering the Query 1, we first select all boxes that intersect the query bounding box. For each column of these boxes, we decompress the column of the box and stream results. During the result streaming, we need to post-filter all entries to make sure only entries within the exact query boundaries are returned.

Query 3 (Aircraft retrieval): We first find all reference bounding boxes that contain the given aircraft ID. Next, we iterate all boxes using this reference box, decompress and parse their entries. Only those entries which match the aircraft ID are returned.

Query 4 (OD pair retrieval): We first find all reference bounding boxes that contain an aircraft with the given origin and the given destination. Afterwards, we traverse all boxes which are referencing these reference boxes. Next, we decompress all these boxes, and parse the entries. Only those entries which match the origin and destination airport are returned.

V. Evaluation

Our empirical evaluation is structured as follows. First, we introduce the evaluation dataset and the setup of experiments in Section V-A. We perform a sensitivity analysis of the parameters for our novel ADS-B index structure ADS-BI in Section V-B. In Section V-C, we report on the results of comparative experiments against the state-of-the-art methods.

A. Datasets and Setup

For the evaluation of our algorithms against state-of-the-art techniques, we use publicly available data from the website https://www.adsbexchange.com. There, data is stored in snapshots of single days; each of them is encoded as a JSON file [35]; and all snapshots within a day are zipped. We evaluate all techniques against three types of data sets, covering distinct periods of time as follows:

- **Hour**: This dataset consists of 60 one-minute snapshots taken on November 16th, 2016 between 3 PM and 4 PM. In total, this database contains 410,466 entries at a total uncompressed storage of 146.5 MB. This is the smallest dataset in our experiment, and it is used for identifying weak competitors which do no scale up to larger datasets.
- **Day**: This dataset consists of 24*60 one-minute snapshots taken on November 16th, 2016 between 00:00 AM and 23:59 PM. In total, this database contains 7,209,949 entries at a total uncompressed storage of 2.6 GB.
- **Week**: This dataset consists of 7*24*60 one-minute snapshots taken from November 13th-19th, 2016 between 00:00 AM and 23:59 PM. In total, this database contains 52,388,243 entries at a total uncompressed storage of 18.9 GB.

The properties of these three temporal periods were found to be representative based on comparison against a longer period of time: July–December 2016. In Figure 8, we show the number of aircraft per snapshot according to different resolutions; the number is usually between 5,000 and 10,000 aircraft per minute. Moreover, in Figure 9 we report the size of uncompressed snapshots; usually between 1–3 MB/minute. All experiments in this study were executed on a server with 32 cores and 320 GB RAM, running Fedora 21 (Linux 4.1.13-100.fc21.x86_64). In Figure 10, we report

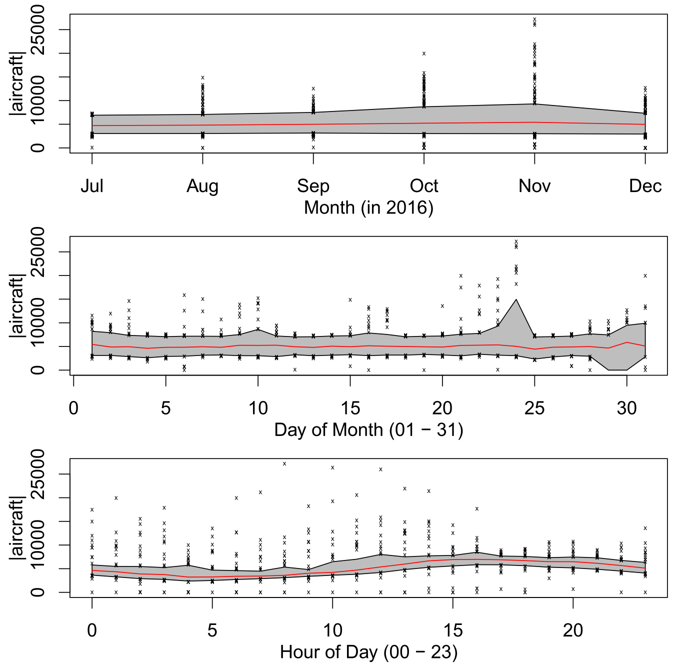


Fig. 8. Number of aircraft in snapshots between July 1st, 2016 and December 31st, 2016. The variation of the number of aircraft is rather small with a few exceptions. We show the monthly variation (top), daily variation (center), and hourly variation (bottom). The grey area covers data points within [5%, 95%] of the values; outliers are visualized as dots. The red line represents the median value.

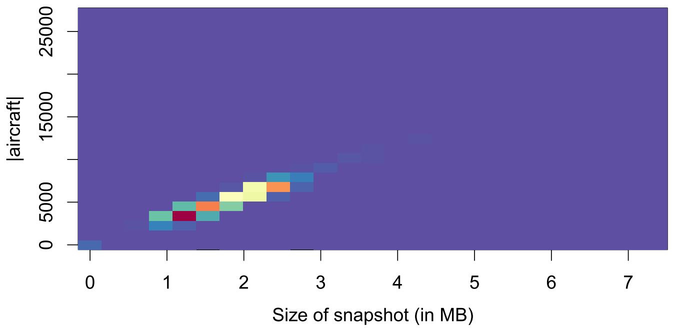


Fig. 9. 2D kernel density estimation for the size of a snapshot against the number of aircraft listed. Most snapshots are within 1–3 MB and cover 3,000-10,000 aircraft. The size of a snapshot grows approx. linearly with the number of aircraft.

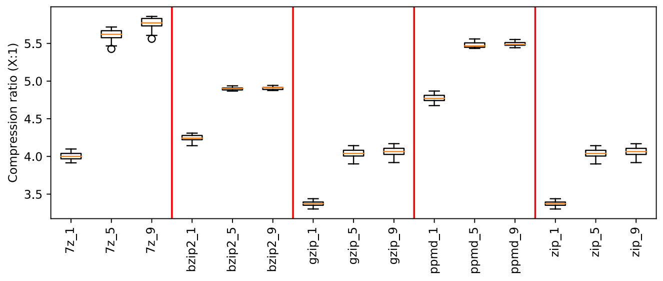


Fig. 10. Boxplots of compression ratios for different compression techniques/parameters (varied from 1 = fast to 9 = best compression. 7z allows for the highest compression ratios; the differences between parameter 5 and 9 are neglectable for 7z.

the compression ratios obtained by different compression method/parameter settings over 10 randomly chosen snapshots. The parameters are standardized from 1 (fastest

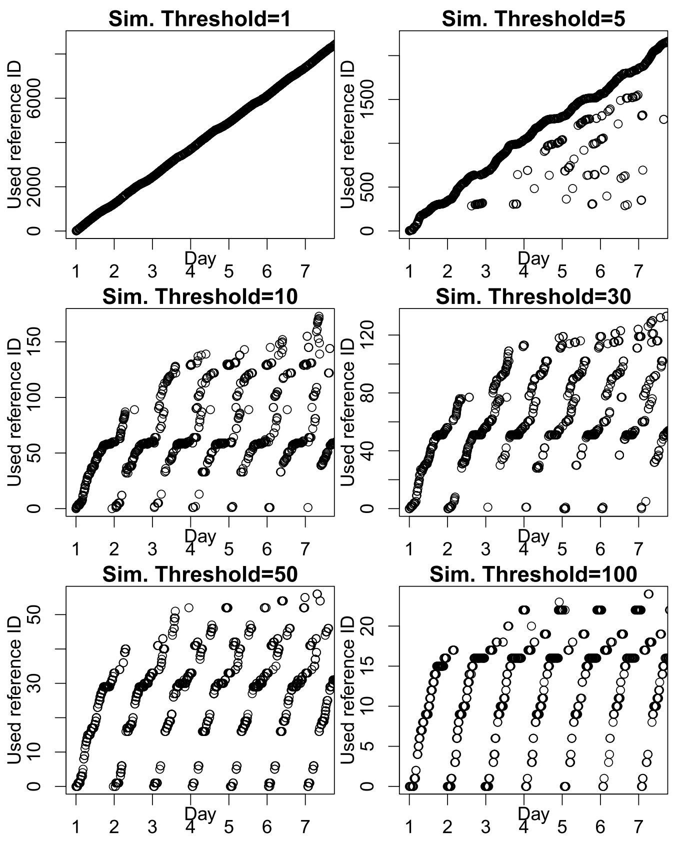


Fig. 11. Evaluation of the references used for processing one week (seven days) with different similarity thresholds. For small thresholds, new snapshots often get assigned new references, and similarities are not exploited. When increasing the threshold, the repetitive pattern, as induced by daily evolution of ADS-B data can be clearly identified. Particularly, for a threshold of 100, we can see that separate days often make use of the same references at the same time of the day.

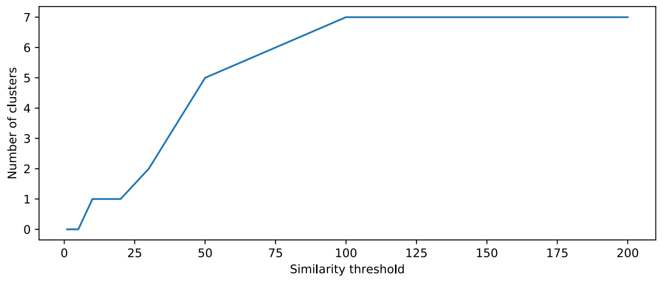


Fig. 12. The number of median minute clusters for a random week of ADS-B data and varying similarity thresholds.

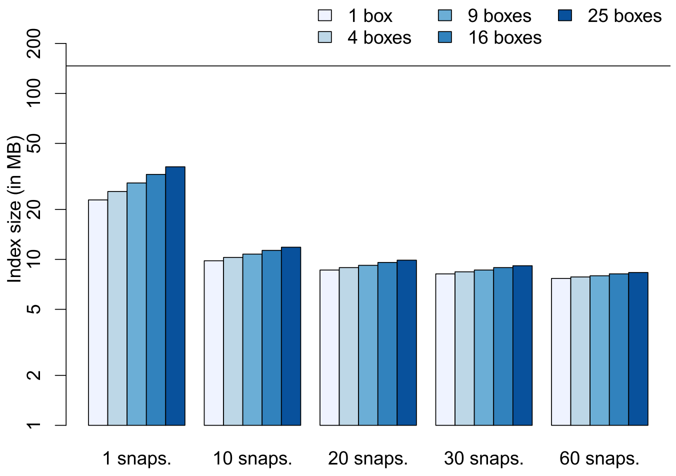


Fig. 13. Sensitivity analysis for the number of temporal snapshots aggregated together (1,10,20,30,60) and the number of total boxes split (1,4,9,16,25). With an increasing number of snapshots, our technique can better exploit the compressibility of data. Increasing the number of spatial boxes slightly deteriorates the compression effect.

compression) to 9 (smallest size). We can see that 7z (using lzma compression) provides the best compression results.

B. Parameter Sensitivity Analysis

First, we perform experiments to derive a useful recommendation for the reference similarity threshold. In Figure 11, we report the number of references for different similarity thresholds. We observe that the similarity threshold has a crucial impact on the reuse of references throughout the computation of an index. Using a small number (less than 10) for the threshold, a new reference is created for each snapshot. When the threshold is increased (larger than approx. 50), we can clearly observe the reuse of references throughout the daily cycles, i.e., during the same time at different days the probability of reuse is very high.

We further evaluate the possibility to select a similarity threshold, based on a sample of the data. The idea is to compress the data for a given time span, e.g. one complete week with 7 days of data. In this case a reasonable similarity threshold should induce a partitioning such that the minute values associated to each reference block are clustered into 7 blocks. The rationale is that for 7 days of the week one should observe 7 repetitions of (approximately) the same flight patterns every day. An indication of that pattern is clearly visible in Figure 11. Essentially, we compute clusters along horizontal lines in Figure 11, one for each reference ID. Figure 12 reports the median number of minute clusters with a varying similarity threshold. We can see that a similarity threshold larger than 100 yields 7 clusters as expected. Therefore, in the remainder of this study, we set the reference similarity threshold to 100, since we want to maximize the reuse of references.

Next, we discuss the impact of the temporal aggregation length and the number of boxes inside an aggregation on index size and query answering times. Figure 13 reports the effect of different combinations of both thresholds on the index size: The number of snapshots per temporal aggregation is set to $[1, 10, 20, 30, 60]$ minutes and the number of boxes (number of horizontal boxes times number of vertical boxes) is set to $[1, 4, 9, 16, 25]$, respectively. The index size is reduced significantly with a growing number of snapshots per temporal aggregation, since we can exploit more similarities among entries for the same aircraft. Increasing the number of boxes slightly increases the storage requirements. This can be explained with the observation that a compression model always starts from the scratch and needs to be learned based on the data. When splitting the data into boxes, the compression model needs to be learned repetitively for each subset, while

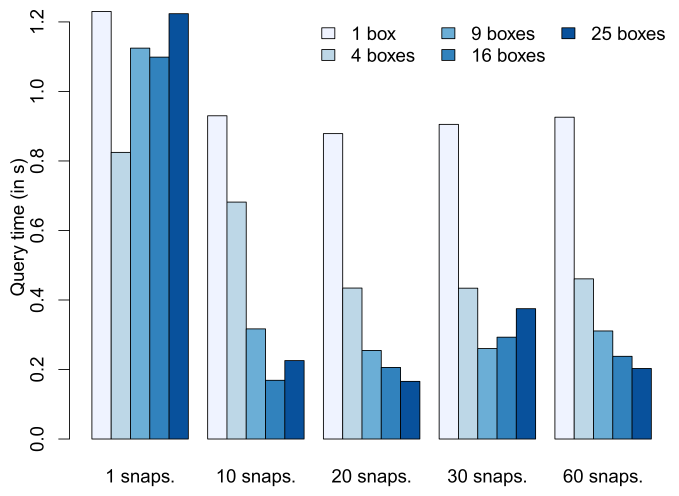


Fig. 14. Sensitivity analysis for the number of temporal snapshots aggregated together (1,10,20,30,60) and the number of total boxes split (1,4,9,16,25). With an increasing number of snapshots, query answering times are reduced significantly. Similarly, increasing the number of spatial boxes often reduces query answering times.

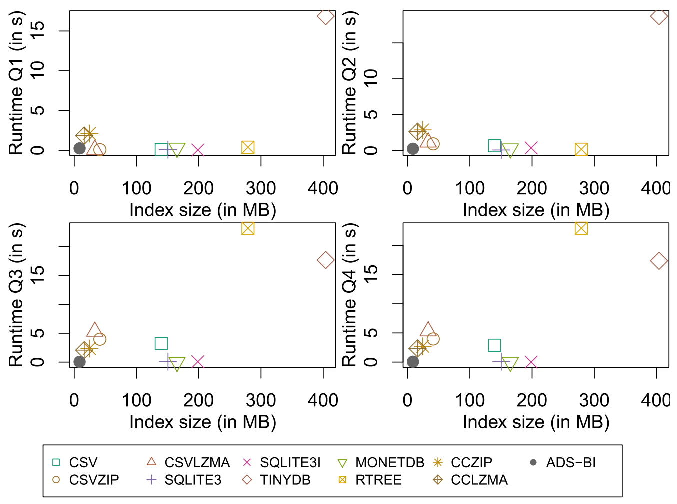


Fig. 15. Comparison of the index size versus running time of four queries for our technique against 10 competitors, based on the hourly dataset. Our method computes the smallest index, while providing competitive query answering times.

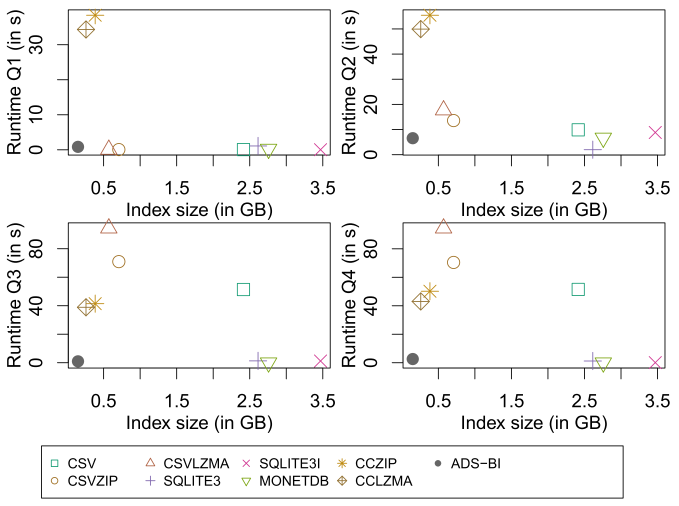


Fig. 16. Comparison of the index size versus running time of four queries for our technique against eight remaining competitors, based on the daily dataset. Our method computes the smallest index, while providing competitive query answering times.

all entries in one box allow for an even shorter encoding of entries, based on learned statistics.

Figure 14 reports exemplarily the effect of both thresholds on query answering times for Query 2 (Bounding box retrieval). Generally, increasing the number of boxes reduces the query answering time, since data can be pruned more efficiently. The running time for the case with single snapshots is particularly long, since we need to perform many random accesses during query answering, one for each snapshot and column. In the remainder of this study, we set number of snapshots per temporal aggregation to 30 and the number of boxes to 9, to address a sweet spot between small index size and short query answering times.

C. Comparative Evaluation

Figure 15 reports the index size and median query answering times for all four types of queries on the smallest dataset: A single hour of ADS-B data. In total, we compare ten competitors with our novel index structure ADS-BI. We can observe a wide range of index sizes: Between 9 MB for ADS-BI and more than 400 MB for TinyDB. The query answering times for most methods are rather short; usually within one second for all types of queries. TinyDB is slow for all types of queries, while the RTree is slow for Query 3 and Query 4. The weakness of RTree for these two query types can be explained by the fact that these two queries do not involve spatial selection (which is the strength of RTrees). In our remaining experiments, we do not evaluate TinyDB and RTree further, given their obvious limitations for our use case. Please note that even additional indexing was implemented on top of an Rtree, the required storage would still be tremendous.

In Figure 16, we report the results for ADS-B of a complete day. In total, we compare eight competitors with ADS-BI. A wide range of index sizes were measured. ADS-BI requires the smallest storage space (0.16 GB). All other competitors either require much more storage (e.g., SQLite3 with 2.8 GB) or suffer from slow query answering (e.g., CCZIP/CCLZMA with more than 40 seconds for Query 3 and Query 4. Given the poor performance of CSVZIP and CSVLZMA on most query types, we do not evaluate these two further.

Figure 17 presents the experimental results for the largest dataset in our study: A set of ADS-B data for a whole week, at 18.9 GB of uncompressed storage. We observe that our novel indexing technique ADS-BI is now the only method that performs fast query answering at a small amount of storage: The index is only 1.1 GB large, which translates into a compression ratio of 16:1. We conclude that none of the competitors exploits the advantages of indexing and compression in a way as ADS-BI does.

Finally, we report how much of the storage for the weekly data is required for the metadata, given a specific similarity

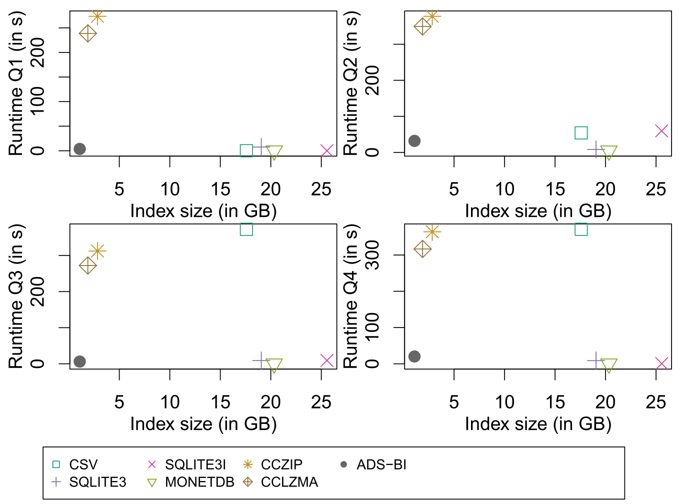


Fig. 17. Comparison of the index size versus running time of four queries for our technique against six remaining competitors, based on the weekly dataset. Our method computes the smallest index, while providing competitive query answering times.

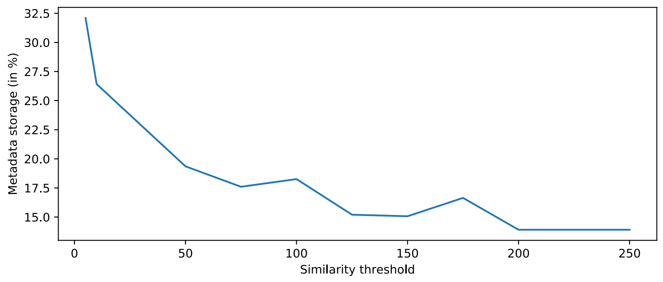


Fig. 18. Fraction of metadata storage in the complete compressed representation with varying similarity thresholds. With an increasing similarity threshold, the percentage of metadata storage reduces significantly, given that fewer reference blocks are stored.

threshold, in Figure 18. With an increasing threshold, less reference blocks are required, and thus the overall storage for the metadata is reduced.

VI. CONCLUSIONS

We presented a new technique ADS-BI for lossless compression and indexing of ADS-B data. Standard indexing techniques neglect ample compression potential, while standard compression techniques do not allow for efficient indexing. Our technique combines the best of both worlds, notably requiring the smallest amount of storage among all competitors, despite its excellent query response times. We evaluate ADS-BI against ten traditional competitors on a collection of four selected query types. Each query type reflects a different view on the data, including snapshot retrieval, bounding box queries, aircraft tracking, and flight route mining. We show that ADS-BI successfully combines compact storage with efficient query answering. Our work on ADS-B data compression and indexing makes it feasible to handle large amounts of air traffic data even on commodity hardware. We discuss the major limitations of our study and how it induces future work directions as follows:

1) In our study, we have addressed four simple query types. In real-world applications, there are more complicated queries, for instance, involving triangulation or separation distance events. These new queries might induce different access patterns and many blocks have to be extracted. In this case, the design of different partition strategies becomes necessary, in order to avoid full data set traversal.
2) In a similar direction, we split the space always with complete cuts either along the spatial or the temporal domain. This might lead to uneven distribution of flights along blocks. In general, more flexible types of cuts could further improve the compression ratio as well as the ability to perform load balancing during the performing query answering process. This will particularly become important, if multiple threads are used for query answering, since the goal would be to keep all threads equally loaded until the end of the computation.
3) We have performed our experiments on data from ADS-B exchange. There are many other data sources available, including commercial and institutional data. The data included in other streams might be slightly different, depending on the attached metadata and the temporal resolution. It would be interesting to evaluate the efficiency of our methodology on other data.
4) We did not compare our results against lossy compressors, since, in general, we do not want to loose accuracy for storing the data: The required level of accuracy depends on the analysis task at hand. If it is clear that there is no analysis task which requires all data at highest resolution, lossy compression techniques could be used to reduce the necessary storage further.
5) In our study, we have evaluated two criteria for comparison only: Query answering time and storage size. Additional criteria could include main memory usage during query answering and also the time required for compression.

REFERENCES

[1] M. Prandini, L. Piroddi, S. Puechmorel, and S. L. Brazdilova, "Toward air traffic complexity assessment in new generation air traffic management systems," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 3, pp. 809–818, Sep. 2011.
[2] J. Wang, H. Mo, F. Wang, and F. Jin, "Exploring the network structure and nodal centrality of china's air transport network: A complex network approach," *J. Transp. Geogr.*, vol. 19, no. 4, pp. 712–721, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0966692310001328
[3] N. Pyrgiotis, K. M. Malone, and A. Odoni, "Modelling delay propagation within an airport network," *Transp. Res. C, Emerg. Technol.*, vol. 27, pp. 60–75, Feb. 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0968090X11000878
[4] L. Pallottino, E. M. Feron, and A. Bicchi, "Conflict resolution problems for air traffic management systems solved with mixed integer programming," *IEEE Trans. Intell. Transp. Syst.*, vol. 3, no. 1, pp. 3–11, Mar. 2002.
[5] X. Sun, S. Wandelt, and F. Linke, "On the topology of air navigation route systems," *Proc. Inst. Civil Eng.-Transp.*, vol. 170, no. 1, pp. 46–59, 2017.
[6] M. C. R. Murca, R. DeLaura, R. J. Hansman, R. Jordan, T. Reynolds, and H. Balakrishnan, "Trajectory clustering and classification for characterization of air traffic flows," in *Proc. 16th AIAA Aviation Technol., Integr., Oper. Conf.*, 2016, p. 3760.
[7] R. DeLaura *et al.*, "Multi-scale data mining for air transportation system diagnostics," in *Proc. 16th AIAA Aviation Technol., Integr., Oper. Conf.*, 2016, p. 3761.

[8] J. Zhang, F.-Y. Wang, K. Wang, W.-H. Lin, X. Xu, and C. Chen, "Data-driven intelligent transportation systems: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1624–1639, Dec. 2011.

[9] M. Zanin, "The reasonable effectiveness of data in ATM," in *Proc. 3rd SESAR Innov. Days*, 2013, pp. 2–5.

[10] P. G. Howard, "The design and analysis of efficient lossless data compression systems," Tech. Rep., 1993.

[11] *Global Operational Data Link Document (GOLD)*, 2nd ed., ICAO, Montreal, QC, Canada, 2013.

[12] G. Enea *et al.*, "Fuel burn estimation modeling for ATM benchmark applications," in *Proc. 12th USA/Eur. Air Traffic Manage. Res. Develop. Seminar*, 2017, pp. 1–10.

[13] N. Bienert and H. Fricke, "Real-time wind uplinks for prediction of the arrival time and optimization of the descent profile," in *Proc. 3rd ENRI Int. Workshop ATM/CNS*, Tokyo, Japan, 2013, pp. 1–6.

[14] J. Djokic, H. Fricke, M. Schultz, and C. Thiel, "Air traffic complexity as a safety performance indicator," *Sci. Military J.*, vol. 4, no. 2, p. 20, 2009.

[15] J. Rosenow, H. Fricke, and M. Schultz, "Air traffic simulation with 4D multi-criteria optimized trajectories," in *Proc. Winter Simul. Conf.*, 2017.

[16] J. Rosenow, M. Lindner, and H. Fricke, "Impact of climate costs on airline network and trajectory optimization: A parametric study," *CEAS Aeron. J.*, vol. 8, no. 2, pp. 371–384, 2017.

[17] H. Fricke, C. Seiß, and R. Herrmann, "Fuel and energy benchmark analysis of continuous descent operations," *Air Traffic Control Quart.*, vol. 23, no. 1, pp. 83–108, 2015.

[18] S. Groth, J. Rosenow, and H. Fricke, "Aviation-induced nitrogen oxide emissions and their effect on the energy budget of the earth-atmosphere system," in *Proc. 7th Int. Conf. Res. Air Transp. (ICRAT)*, 2016.

[19] M. Vogel, K. Schelbert, H. Fricke, and T. Kistan, "Analysis of airspace complexity factors' capability to predict workload and safety levels in the TMA," in *Proc. 10th USA/Eur. Air Traffic Manage. R&D Seminar*, Chicago, IL, USA, 2013, pp. 1–10.

[20] M. Lindner, S. Foerster, J. Rosenow, and H. Fricke, "Aircraft fleet mix optimization in airline networks under ecological and soft time window constraints," in *Proc. 7th Int. Conf. Res. Air Transp. (ICRAT)*, Philadelphia, PA, USA, Jun. 2016.

[21] C. Newman, *SQLite (Developer's Library)*. Indianapolis, IN, USA: SAMS, 2004.

[22] S. Khoshafian, G. P. Copeland, T. Jagodis, H. Boral, and P. Valduriez, "A query processing strategy for the decomposed storage model," in *Proc. 3rd Int. Conf. Data Eng.*, Washington, DC, USA, 1987, pp. 636–643. [Online]. Available: http://dl.acm.org/citation.cfm?id=645472.655555

[23] P. A. Boncz and M. L. Kersten, "MIL primitives for querying a fragmented world," *VLDB J.*, vol. 8, no. 2, pp. 101–119, Oct. 1999. [Online]. Available: http://dx.doi.org/10.1007/s007780050076

[24] D. J. Abadi, P. A. Boncz, and S. Harizopoulos, "Column-oriented database systems," *Proc. VLDB Endowment*, vol. 2, no. 2, pp. 1664–1665, Aug. 2009. [Online]. Available: http://dx.doi.org/10.14778/1687553.1687625

[25] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: An efficient and robust access method for points and rectangles," *ACM SIGMOD Rec.*, vol. 19, no. 2, pp. 322–331, 1990.

[26] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Inform.*, vol. 4, no. 1, pp. 1–9, 1974.

[27] O. Procopiuc, P. K. Agarwal, L. Arge, and J. S. Vitter, "Bkd-tree: A dynamic scalable kd-tree," in *Proc. Int. Symp. Spatial Temporal Databases*, 2003, pp. 46–65.

[28] J. Muckell, J.-H. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. S. Ravi, "SQUISH: An online approach for GPS trajectory compression," in *Proc. COM.Geo*, 2011, pp. 13:1–13:8. [Online]. Available: http://doi.acm.org/10.1145/1999320.1999333

[29] P. Cudre-Mauroux, E. Wu, and S. Madden, "TrajStore: An adaptive storage system for very large trajectory data sets," in *Proc. ICDE*, 2010, pp. 109–120. [Online]. Available: http://dblp.uni-trier.de/db/conf/icde/icde2010.html#Cudre-MaurouxWM10

[30] V. Botea, D. Mallett, M. A. Nascimento, and J. Sander, "PIST: An efficient and practical indexing technique for historical spatio-temporal point data," *GeoInformatica*, vol. 12, no. 2, pp. 143–168, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10707-007-0030-3

[31] J. Muckell, P. W. Olsen, Jr., J.-H. Hwang, C. T. Lawson, and S. S. Ravi, "Compression of trajectory data: A comprehensive evaluation and new approach," *GeoInformatica*, vol. 18, no. 3, pp. 435–460, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10707-013-0184-0

[32] J. Muckell, J.-H. Hwang, C. T. Lawson, and S. S. Ravi, "Algorithms for compressing GPS trajectory data: An empirical evaluation," in *Proc. GIS*, 2010, pp. 402–405. [Online]. Available: http://doi.acm.org/10.1145/1869790.1869847

[33] S. Wandelt and X. Sun, "Efficient compression of 4D-trajectory data in air traffic management," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 2, pp. 844–853, Apr. 2015.

[34] S. Wandelt, X. Sun, and Y. Zhu, "Lossless compression of public transit schedules," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 11, pp. 3075–3086, Nov. 2016.

[35] D. Crockford, *The Application/JSON Media Type for JavaScript Object Notation (JSON)*, document 4627, 2006.

Sebastian Wandelt received the Ph.D. degree in computer science from Hamburg University of Technology. He is a Professor with the School of Electronic and Information Engineering, Beihang University, and Beijing Key Laboratory for Network-Based Cooperative ATM, Beijing. His research interests include transportation systems, scalable data management, and compressing/searching large collections of objects.

Xiaoqian Sun received the Ph.D. degree in aerospace engineering from Hamburg University of Technology in 2012. She is an Associate Professor with the School of Electronic and Information Engineering, Beihang University, and Beijing Key Laboratory for Network-Based Cooperative ATM, Beijing. Her research interests mainly include air transportation networks, multi-modal transportation, and multi-criteria decision analysis.

Hartmut Fricke studied aeronautics and astronautics at, and also received the Ph.D. degree from, Technische Universität (TU) Berlin. He finished the Habilitation on integrated collision risk modeling for airborne and ground-based systems in 2001. Since 2001, he has been Professor of aviation technologies and logistics with TU Dresden. His research interests include aircraft trajectory optimization and benchmarking, safety assessment techniques in ATM, and innovative airport planning.