Open Geospatial Data, Software and Standards

SOFTWARE Open Access

GeomDiff — an algorithm for differential geospatial vector data comparison

Atle Frenvik Sveen

Abstract

Diffs, a concept known from source code version control systems such as git, is interesting for geospatial, event-based workflows. We investigate how the native mathematical structure of vector geometries can be utilized in order to create a diffing algorithm tailored to geospatial vector data. Diffing algorithms are a well-researched area which dates to the 1970ies; however, we find that geospatial diffing operations tends to be carried out using generic algorithms combined with a pre- and post-processing step. We created GeomDiff, an algorithm and storage format tailored to geospatial vector data. The creation time, apply/undo time, and patch size of GeomDiff was compared to three other generic algorithms by running an online experiment using 2.5 million real-world geometry pairs from OpenStreetMap. We found that the GeomDiff algorithm performs better than or on-par with the alternatives on point-geometries, and complex geometries with a small (< 500) vertex count. We argue that there are both computation time and storage space improvements to be gained by using a tailored diffing algorithm for geospatial vector data. These promising first results encourages further refinement of the algorithm in order to handle complex geometries efficiently as well.

Keywords: Geospatial data management, Diffing, Event based workflows

Introduction

In computer science, a diff[1] is a set of machine-executable instructions to transform version n of source code or documentation into version $n + 1$ [1]. The concept of diffs is an essential component of source code version control systems such as *git* [2], one of the fundamental building blocks of modern software engineering. Other application areas also take advantage of the diff concept, enabling real-time collaborative editing tools such as Google Docs [3]. The concept of diffs is also important when working with event-based workflows, where messages describing changes are a core component [4]. A recent, geospatial, application of the concept is "Sno" [5] which uses git to apply version control to geospatial data.

The methods used for creating a diff and the format it is stored in will affect both *creation* time, *storage requirements*, and *apply* and *undo* time. These metrics affect the overall performance and requirements of a diff-based workflow. Using a diffing algorithm and diff storage format tailored to geospatial vector data has the possibility to provide an efficient, performant, and reliable event-based workflow for geospatial data management.

Diffing algorithms specifically tailored to geospatial data are rare in the literature. Thus, we implemented "GeomDiff", an algorithm for geospatial diffing. GeomDiff takes advantage of the native mathematical properties of geospatial vector data in order to improve the performance of geospatial event-based workflows.

We review existing literature on diffing algorithms and formats in general and provide an overview of existing solutions for versioning of geospatial vector data using diffs. The concepts used to implement the GeomDiff algorithm is then explained and presented.

To evaluate the proposed algorithm, we perform a large-scale experiment using real-life data in a controlled

[1]Also known as an *edit script*, a *changeset*, a *patch*, or a *delta*

Correspondence: atle.f.sveen@ntnu.no
Norwegian University of Science and Technology, Trondheim, Norway

Springer Open

© The Author(s). 2020 **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

and replicable cloud-based environment. The GeomDiff algorithm is compared to three other approaches to diffing of geospatial vector data in order to investigate how it performs on creation time, apply/undo time, and storage requirements.

Implementation

Motivation

Creating a diff is the process of finding changes between two versions of an object and describing them. Change detection is in itself a topic that is widely studied with regards to Remote sensing and image processing [6], as well as computer science [7]. The term "diff" itself was introduced by Hunt & MacIlroy [8], which described a program which "[…] reports differences between two files, expressed as a minimal list of line changes to bring either file into agreement with the other". The GNU diff program is based on the work carried out by Miller & Myers [9], and Myers [10], who found that the dual problems of finding a longest common sub-sequence of A and B and finding a shortest edit script for transforming A into B are equivalent to finding a shortest/longest path in an edit graph.

The diff program and its iterations are focused on comparing text files and are therefore well suited for tracking revisions to text and computer source code. Variations of this approach serves as a building block of version control systems [2]. Other researchers have focused on creating systems for detecting changes in hierarchically structured information, such as data stored in a database [11] and binary data [12]. Dedicated diff tools and formats have also been created for formats such as JSON [13] and XML [14].

We did not find examples of specialised diffing algorithms for geospatial data in the literature, but some approaches from the industry was identified and are presented in the following. The GeoDiff library [15] aims to simplify vector data management by "keep[ing] track of changes, calculate the differences, merge and consolidate the differences". However, the library seems to focus on changes at a dataset level. A related idea is to apply version control concepts to geospatial vector data. This has been attempted several times by various actors. GeoGit, later renamed GeoGig, was released in 2014 and "allows for decentralized management of versioned geospatial data" [16]. However, an inspection of the source code does not indicate that the project employs diffs but stores each separate change as a new geometry. A more recent approach to version control of geospatial data is Sno, which is built on top of git [5]. This means that this system uses a text-based differ at its core, but presumably with some modifications.

Principles of GeomDiff

Existing diffing algorithms for textual data, binary data, or format-specific algorithms can be applied to geospatial vector data using pre- and post-processing steps. However, geospatial geometries are natively mathematically defined as vectors in N-dimensions. By converting them to text or some other format, we lose the ability to utilize mathematical relations to describe changes in the geometries. This is the main idea behind GeomDiff; to take advantage of the opportunities presented by the mathematical nature of vector geometries in order to create a more efficient algorithm.

Table 1 presents a selection of example geometries in their original and modified state, along with an example of a change script. For point geometries we record the operation (create, delete, or modify) as well as the change to the coordinate expressed as a delta. In order to support reverting a change script, the current value before deletion is recorded in a delete operation.

While a point consists of a single coordinate pair, other geometry types are more complex. These consists of one or more, possibly nested, ordered lists of coordinate pairs. Linestrings are described by a single, ordered list of coordinate pairs, where each coordinate pair describes a vertex. The linestring can be both created and deleted in a similar fashion to a point, but a modification is more complex. Changes to each vertex can be described using the edit script outlined for coordinates, but we need to keep track of the indices of the changed vertices as well, as illustrated in Table 1 (id = 4).

A polygon is even more complex, as it may contain hulls. Thus, a polygon consists of n ordered lists of coordinates, and each of these can be added, deleted, or modified. In addition, each of the vertices in each list can be added, deleted, or modified. However, just as the coordinate edit script is used to represent each change to a linestring, a linestring change script can describe the change to each ring of a polygon (Table 1, id = 5). Using this hierarchical pattern, multi-geometries are handled by adding another layer; multi-point change scripts are lists of point change scripts, multi-linestrings and multi-polygons extends linestring and polygon change scripts in the same manner.

Geospatial data is in many cases represented as a collection of features. A feature is a combination of a geometry and a set of textual or numeric attributes or properties. In order to create a feature patch, the attributes must be handled as well. While this is an important aspect of a geospatial data versioning workflow, the GeomDiff algorithm is not designed to work on features. However, feature attributes are

Table 1 Edits to geometries and their associated change scripts. Ids 1–3 are modification, creation, and deletion of a point, id = 4 is modification of a linestring by inserting, modifying, and deleting vertices. Id =5 is modification of a polygon by modifying one vertex in the shell and deleting the hull. Geometries are described using the WKT format

Id	Geometry Type	Original	Modified	Change Script
1	Point	(10.53 60.10)	(10.52 60.10)	Modify, (-0.01 0)
2	Point	Null	(10.53 60.10)	Create, (10.53 60.10)
3	Point	(10.53 60.10)	Null	Delete, (10.53 60.10)
4	LineString	(1 1, 2 2, 3 3, 4 4)	(0 0, 1 1, 2.5 2.5, 3 3)	{0: Insert, (0 0), 1: Modify, (0.5 0.5) 3: Delete, (4 4)}
5	Polygon	((0 0, 10 0, 5 10, 0 0), (1 1, 2 2, 2 1, 1 1))	((0 0, 10 0, 6 10, 0 0))	{0: Modify, {2: (1, 0)}, 1: Delete, (1 1, 2 2, 2 1, 1 1)}

usually simple key-value pairs that can be represented using formats such as JSON or XML. As previously discussed, specialized diffing algorithms for these formats exists and can be used. An example of this approach is implemented in the attached file FeatureDiffer.cs.

GeomDiff implementation

The main principles described in the previous section are implemented using a generic *Diff* class, as outlined in Listing 1. Here, the value is a generic property, which means that we can represent change scripts for all geometry types using this class. In the case of a point, the *PointDiff* inherits from *Diff* using the *CoordinateDelta* as *TComponent.* Describing changes to a linestring geometry is more complex, as we need to keep track of the vertex indices. This is where the Index and Operation properties on the Diff class are used, as the *LineStringDiff* class uses a *List* < *PointDiff* > as *TComonent*. The same pattern is repeated for the other geometry types.

While the presented class hierarchy represents changes, it does not describe how these changes are detected. In the case of point geometries, the difference is expressed by the change in each coordinate, which is a straightforward mathematical computation. For linestrings and other

```
public abstract class Diff<TComponent>
{
       public int Index {get; set;}
       public TComponent Value {get; set;}
       public Operation Operation {get; set;}

}


public class CoordinateDelta
{
    public double? X { get; set; }
    public double? Y { get; set; }
    public double? Z { get; set; }
}

public class PointDiff: Diff<CoordinateDelta> {}


public class LineStringDiff: Diff<List<PointDiff>> {}
```

Listing 1: C# Class hierarchy used to describe diffs relating to diferent geometry types. The abstract, generic Diff class serves as a basis, and the CoordinateDelta class is the simplest example of a diffed component. A PointDiff is implemented by providing the Diff class the CoordinateDelta. A LineStringDiff is more complex, as it consists of a list of PointDiffs.

```
List<TComponent> PatchList(List<TComponent> components, List<IDiff<TComponent>> diffs)
{
    var patched = new List<TComponent>();

    var numElements = Math.Max(components.Count - 1, diffs.Max(v => v.Index));

    for (var index = 0; index <= numElements; index++)
    {
        //insert all inserts at this index
        var inserts = Util.GetDiffs(index, Operation.Insert, diffs);
        patched.AddRange(inserts.Select(insert => insert.Apply(null)));

        //skip the component if it is marked for deletion
        var delete = Util.GetDiff(index, Operation.Delete, diffs);
        if (delete != null) continue;

        //check if the original list still has elements
        var element = Util.GetAt(index, components);
        if (element == null) continue;

        //find and apply any modifications
        var modify = Util.GetDiff(index, Operation.Modify, diffs);
        patched.Add(modify != null ? modify.Apply(element) : element);
    }

    return patched;
}
```

Listing 2: Simplified version of the PatchList method used for applying a patch, implemented in C#.

sequences of coordinates, we employ a generalized version of the Myers diff algorithm [10], where the input is two lists of components and a function to compare them. By utilizing the recursive strategy presented above, this approach works both for comparing individual vertices in a linestring and for comparing linear rings in a polygon. Thus, diff creation can be implemented by combining this approach with a method for compacting diffs by merging consequent inserts and deletes into modify operations.

Applying a diff to a geometry follows the same recursive pattern. A simple mathematical operation can patch a single coordinate. A list of components (coordinates, linear rings, or geometries) is patched using the *PatchList* method, reproduced in Listing 2. Undo operations use the same method, but a pre-processing step reversing the diff is applied first.

Another important aspect is a storage format for the created diffs. Serializing and deserializing the generated C# objects is an easy solution, but this introduces an unnecessary coupling to a specific implementation. In addition, this is not an efficient approach in terms of storage requirements.

Thus, we created a binary format for storing diffs, inspired by the WKB (Well Known Binary [17]) format. The format consists of a header, describing the geometry type and dimensions, and the actual diff elements. Writer and reader implementations to convert to and from C# objects are created as part of the implementation.

Benchmark design and implementation

The performance of the GeomDiff algorithm was examined by performing a large number of diff *creation*, *apply*, and *undo* operations on data from OpenStreetMap. In addition, the same operations were performed on the same data, using three other diffing implementations. This ensures that we can perform a statistical analysis to test our hypotheses.

The benchmark was performed in a Microsoft Azure cloud environment. This allows for easy scale-out in order to handle high workloads in parallel. Furthermore, it ensures consistent hardware performance at a reasonable price. The setup consists of a virtual machine running a PostgreSQL/PostGIS database, an Azure Function App, and an Azure storage account for message passing and temporary storage. In addition, NodeJS and Python scripts were developed to import test-data, orchestrate the benchmark, and analyse the results.

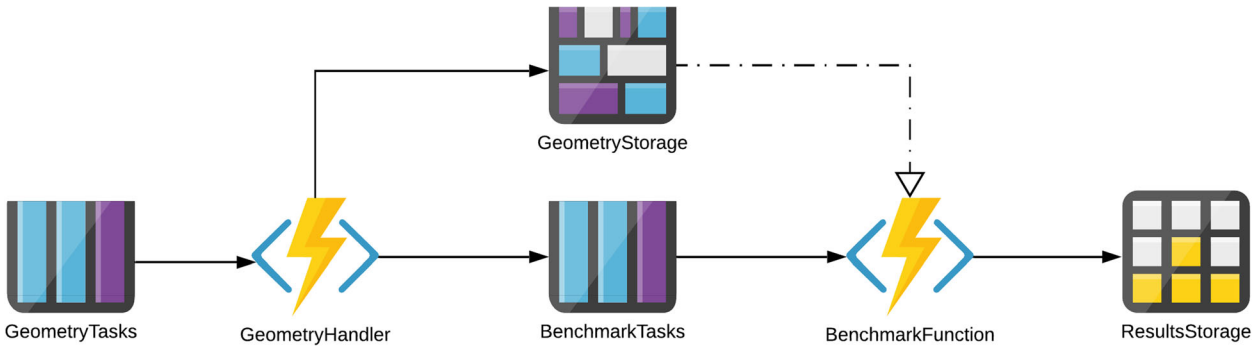

Fig. 1 The Azure pipeline for running the benchmark. The GeometryTasks queue is loaded with one message per geometry pair. The GeometryHandler processes each message, by fetching the two geometries in the pair and storing them in a blob store. It then puts four messages on the BenchmarkTasks queue, one for each algorithm under test. This message has a reference to the blob id of the geometry pair. The benchmark function then loads the geometry pair and executes the three operations (create, apply and, update) using the specified algorithm, while logging time usage. The results are then stored in a table storage for later retrieval and analysis

An extract from the open licensed OpenStreetMap (OSM) dataset [18] was used as test data. A full history extract for Norway was manually downloaded from geofabrik.de and imported to a PostGIS database, using a custom-made script [19]. To prepare the dataset for the benchmark, a series of queries was run to identify geometry pairs. A geometry pair is defined as two versions of the same geometry. In cases where the OSM dataset had more than two versions of a geometry, we chose the minimum and maximum version to represent the two versions. Since the OSM data model stores all linestring and polygon vertices as points, and referencing them from a "ways" table, we selected all points with at least one tag to represent points. Linestrings and polygons was created from the ways table, based on whether they were closed or not. This process created a benchmark dataset consisting of 1,335,489 point-, 813,503 linestring-, and 433,776 polygon-pairs. For each linestring- and polygon pair the *NumVertices* variable (Eq. 1) was computed and stored.

$$NumVertices = avg(numVertices(geometry_{v1}), numVertices(geometry_{v2})) \quad (1)$$

The test pipeline consists of an Azure storage account and a Function App, as depicted in Fig. 1. The data flow in this setup is highly parallelisable and can be scaled to handle increased workloads with minimal effort. The first stage of the pipeline is a storage queue (*GeometryTasks*), populated by a NodeJS script. This queue contains one message per geometry pair in the benchmark dataset, consisting of an id and two version numbers, as well as the geometry type. Attached to this queue is a function app (*GeometryHandler*) which fetches the corresponding geometries from the PostGIS database and stores them in a blob store (*GeometryStorage*). In addition, this message puts four messages on a second queue (*BenchmarkTasks*), with a blob id and a differ name. The second function app (*BenchmarkFunction*) is triggered by the BenchmarkTasks queue and fetches the geometry pair from GeometryStorage and runs the benchmark using the indicated algorithm. On completion the results are saved to a table storage (*ResultsStorage*).

The actual benchmark of the algorithms under test are performed in the *BenchmarkFunction* function app. This is a serverless instance running .NET core code [20]. The actual algorithm implementations are written to conform to a common interface, as depicted in Listing 3, and made available as a NuGet package, which is imported by the *BenchmarkFunction* app.

```
public interface IGeometryDiff
{
    byte[] CreatePatch(IGeometry oldGeom, IGeometry newGeom);
    IGeometry ApplyPatch(IGeometry oldGeom, byte[] patch);
    IGeometry UndoPatch(IGeometry newGeom, byte[] patch);
}
```

Listing 3: IGeometryDiff interface used to implement the diff algorithms

Table 2 Diffing algorithms used in the benchmark

Algorithm	Format	Library	Link
TextDiff	Text	Diff Match Patch	https://github.com/google/diff-match-patch
JsonDiff	JSON	jsondiffpatch.net	https://github.com/wbish/jsondiffpatch.net
BinaryDiff	Binary	Deltaq	https://github.com/jzebedee/deltaq
GeomDiff	Vector Geometry	GeomDiff	https://github.com/atlefren/GeomDiff

The algorithm implementations under test are listed in Table 2. The three other algorithms are based on open source implementations of three different diffing formats; textual data, binary data, and JSON-data. All these algorithms require a pre- and a post-processing step, where the geometry is converted to the appropriate format and back. These algorithms where chosen as they represent existing approaches to handling diffing of geospatial vector data. The Bsdiff algorithm [12], used in the BinaryDiff implementation does not support the undo operation, but we still chose to include it in the benchmark, as it represents another approach to diffing. The pre- and post-processing steps use the open source NetTopologySuite [21] library to convert the geometries into appropriate formats. Well Known Text (WKB) [17], Well Known Binary (WKB) [17], and GeoJSON [22] was chosen as formats to convert to text, binary, and json, respectively.

Results

For each of the three geometry types in the test dataset we performed hypothesis testing on four metrics:

- Creation time: The time it takes to create a diff given two versions of a geometry.
- Apply time: The time it takes to create version n + 1 of a geometry, given version n and a diff.
- Undo time: The time it takes to roll back to version n of a geometry, given version n + 1 and a diff.
- Patch size: The physical size of the created diff.

We expect the GeomDiff algorithm to exhibit faster creation-, apply- and, undo-time for point, linestring and polygon geometries compared to the other algorithms. In addition, we expect the GeomDiff algorithm to produce smaller patches.

The statistical testing was performed using the following procedure, implemented as a Python script [23]. All statistical tests were performed using a significance level of 0.05. For each metric of each geometry type the recorded data for each of the four algorithms was loaded. First, all errors were counted, recorded (see Table 6), and then removed before further analysis. An error is either an exception thrown by the code, or an instance where the patch did not create the expected result.

Second, a D'Agostino and Pearson's test [24] was applied to check each group for normal distribution. Since none of the groups were normally distributed ($p < 0.05$), a Kruskal-Wallis H-test [25] was then applied to test H_0, that the samples from all algorithms came from the same distribution. Since H_0 was rejected in all cases ($p < 0.05$), we continued with a post hoc test to perform pairwise comparisons between the four algorithms. Using Conover's test [26], we found that none of the pairs where statistically similar ($p < 0.05$). This means that all differences between the mean values for each algorithm are significant.

Point geometries

For point geometries (Table 3), a total of 1,335,489 geometry pairs were checked for each algorithm. Overall, the BinaryDiff algorithm is slower than the fastest algorithm by a factor of 1000 on create and apply. The TextDiff and JsonDiff algorithms show comparable results, apart from patch size. The GeomDiff algorithm produces

Table 3 Benchmark results for point geometries. Best results in each case in bold. The standard deviation of patch size for points using the GeomDiff algorithm is 0, as a point change is described using two doubles. This means that the size of a point patch will always be the size of two doubles and metadata of a fixed size

Algorithm	Create Time (ms)		Apply Time (ms)		Undo Time (ms)		Patch Size (b)	
	Mean	St.dev	Mean	St.dev	Mean	St.dev	Mean	St.dev
TextDiff	0.22	10.92	0.47	15.64	0.32	2.32	54.0	30.0
JsonDiff	0.38	7.21	0.21	2.51	0.16	1.62	184.0	94.0
BinaryDiff	190.88	272.07	67.39	131.74	–	–	168.0	20.0
GeomDiff	**0.03**	1.80	**0.02**	0.58	**0.01**	0.40	**25.0**	0.0

Table 4 Benchmark results for linestring geometries. Best results in each case in bold

Algorithm	Create Time (ms)		Apply Time (ms)		Undo Time (ms)		Patch Size (b)	
	Mean	St.dev	Mean	St.dev	Mean	St.dev	Mean	St.dev
TextDiff	9.01	58.56	1.00	10.98	1.04	4.61	623.44	1733.22
JsonDiff	**2.27**	35.96	1.12	10.08	1.06	8.23	3064.38	9656.37
BinaryDiff	183.47	333.88	57.07	159.81	–	–	**357.16**	635.37
GeomDiff	57.83	3281.33	**0.21**	8.20	**0.19**	5.22	419.63	1355.67

the smallest patch in the shortest time and is also the fastest to apply and undo.

Linestring geometries

For linestring geometries (Table 4), a total of 813,503 geometry pairs were checked for each algorithm. The mean number of vertices is 24, the 99th percentile 236. When it comes to performance, the GeomDiff algorithm is considerably slower to create patches, albeit with a large standard deviation, but it is still the fastest on create and undo time. The JsonDiff algorithm is the fastest to create patches, but the patches created by the JsonDiff algorithm are on average larger than patches created by the BinaryDiff algorithm by a factor of 8.5.

Polygon geometries

For polygon geometries (Table 5), a total of 433,776 polygon pairs with a mean vertex count of 28 (99th percentile 299) were checked. In terms of performance, the polygon dataset exhibits much the same trends as the linestring data. The standard deviations are large, and the BinaryDiff and GeomDiff algorithms are considerably slower than TextDiff and JsonDiff when it comes to create time, but at the same time they produce the smallest patches.

Error counts

The error counts (Table 6) show that the GeomDiff algorithm encountered 22 and 34 create errors, and 33 and 45 patch and undo errors on linestrings and polygons, respectively. The TextDiff algorithm failed to undo 38,480 linestring pairs (5%) and 18,396 (4%) polygon pairs correctly.

For point geometries the rates are close to zero (< 1 ‰) for all metrics.

The create errors for the TextDiff algorithm are all "Invalid URI: The Uri string is too long.". This error originates in the Diff Match Patch library, which uses URL encoding provided by the C# standard library. This shows that the limiting factor for string lengths, and by extension vertex count, are the URL encoding method.

For the GeomDiff algorithm, all create errors are "Timed out after 60000 ms". This is a hard limit built into the GeomDiff library to avoid long-running operations to block for an unreasonable amount of time.

Vertex number effects

For linestring and polygon geometries, the GeomDiff algorithm exhibits an unusually large standard deviation on the Create Time metric. In order to investigate possible causes for this, we identified the upper 99 percentile and removed observations with values higher than this. This is shown in Table 7. We see that by removing 1% of the observations the standard deviation is reduced by two orders of magnitude.

One possible explanation for this is that the create time for the GeomDiff algorithm increases as the number of geometry vertices increase. This explanation is supported by the create failures on 22 linestring and 34 polygon geometries. In these cases, the algorithm ran for 60 s before timing out. Examining the geometries which caused the errors, we find an average vertex count of 1677 and 1576 for linestrings and polygons, respectively. For the top 1 (slowest) percentile, the vertex count averages were 300 and 364. These numbers are both a substantial increase from the full population, which on average has a vertex count of 24 for linestrings and 28

Table 5 Benchmark results for polygon geometries. Best results in each case in bold

Algorithm	Create Time (ms)		Apply Time (ms)		Undo Time (ms)		Patch Size (b)	
	Mean	St.dev	Mean	St.dev	Mean	St.dev	Mean	St.dev
TextDiff	7.53	70.12	1.08	39.82	0.92	7.22	481.37	2023.27
JsonDiff	***3.50***	76.01	1.15	20.80	0.95	10.60	2970.73	15,035.43
BinaryDiff	224.40	571.71	69.11	272.37	–	–	***301.82***	684.04
GeomDiff	118.09	5159.74	***0.39***	79.77	***0.25***	7.02	306.00	1397.86

Table 6 Error counts for the tested algorithms, grouped by geometry type and operation. A create error represents a situation where the algorithm threw an exception during execution, while apply and undo errors represents situations where applying or undoing a diff does not produce the expected geometry

Algorithm	Point			Linestring			Polygon		
	Create	Patch	Undo	Create	Patch	Undo	Create	Patch	Undo
TextDiff	0	1	4	3	3	38,480	1	1	18,396
JsonDiff	0	1	1	0	0	0	0	0	0
BinaryDiff	0	1	–	0	0	–	0	0	–
GeomDiff	0	1	1	22	33	33	34	45	45

for polygons. In other words, large vertex counts seem to indicate long running times.

To further investigate whether the vertex count variable influences create time, we calculated the Pearson correlation coefficient [27] between creation time and vertex count, as shown in Table 8. We see that the correlation change between the whole population and the top 1 percentile is substantial for the GeomDiff algorithm (+ 0.17 / + 0.81), while it is relatively stable or decreasing for the other algorithms (- 0.02 / - 0.01 for the TextDiff algorithm). Thus, we suspect that the vertex count in linestring and polygon geometries affects the creation time for the GeomDiff algorithm significantly, and especially for large numbers of vertices.

By grouping the create time results by vertex count and computing average creation time for each group (Fig. 2 and Fig. 3), we find that all algorithms except the BinaryDiff algorithm show an increase in creation time with increasing number of vertices. However, for the GeomDiff algorithm, there is a sharp increase when exceeding a vertex count of 500, for both linestrings and polygons.

Discussion

Our data shows that the GeomDiff algorithm outperforms the other tested algorithms by a large margin when working with point geometries. It creates the smallest patches in the shortest time and is also fastest at applying and undo patches.

When it comes to more complex geometries (Table 4 and Table 5), the results are more varied. The JsonDiff algorithm is the fastest for creating both linestring- and polygon-patches, while the BinaryDiff algorithm creates the smallest patches. However, the JsonDiff algorithm creates the largest patches, while the BinaryDiff algorithm is the slowest one in both creation and apply time. Moreover, this algorithm does not support the undo operation.

The results for the GeomDiff algorithm with regards to linestrings and polygons are more complex. The algorithm is the fastest on both apply and undo, and it produces patches not much larger than the BinaryDiff algorithm. However, the creation time shows a large variance. Based on our test data, we found that this is related to number of vertices in the diffed geometries. When this exceeds 500 vertices, we see a sharp increase in creation time. In addition, we recorded several occurrences where the algorithm timed out after 60 s for some geometries with large vertex counts.

However, both the mean and 99th percentile of vertex counts in both linestrings and polygons are considerably lower than 500 in the OSM test-dataset. This means that, for datasets comparable in complexity to OSM, the vertex issue is not likely to be major. In addition, diffs are usually created only once, but applied and undone multiple times. Thus, faster apply and undo speeds are more important than creation times. Nevertheless, the fact that the GeomDiff algorithm degrades, and sometimes fails, on geometries with a high vertex count is not ideal. This behaviour is worth determining the cause of and remedy before the algorithm can be considered ready to use in a real-life situation where performance and repeatability is essential.

Table 7 Create time for linestring and polygon geometries with the upper 99 percentile values excluded from the analysis

Algorithm	Linestring		Polygon	
	Mean	St.dev	Mean	St.dev
TextDiff	4.60	14.14	2.72	9.20
JsonDiff	1.12	2.46	0.99	2.33
BinaryDiff	165.05	180.63	180.43	195.48
GeomDiff	2.44	12.68	1.28	7.62

Table 8 Pearson correlation coefficient between creation time and vertex count for the full population and the top 1 percentile

Algorithm	Linestring		Polygon	
	All	Top 1%	All	Top 1%
TextDiff	0.50	0.48	0.45	0.44
JsonDiff	0.30	0.23	0.27	0.16
BinaryDiff	0.01	0.03	0.01	-0.01
GeomDiff	0.26	0.43	0.30	0.48

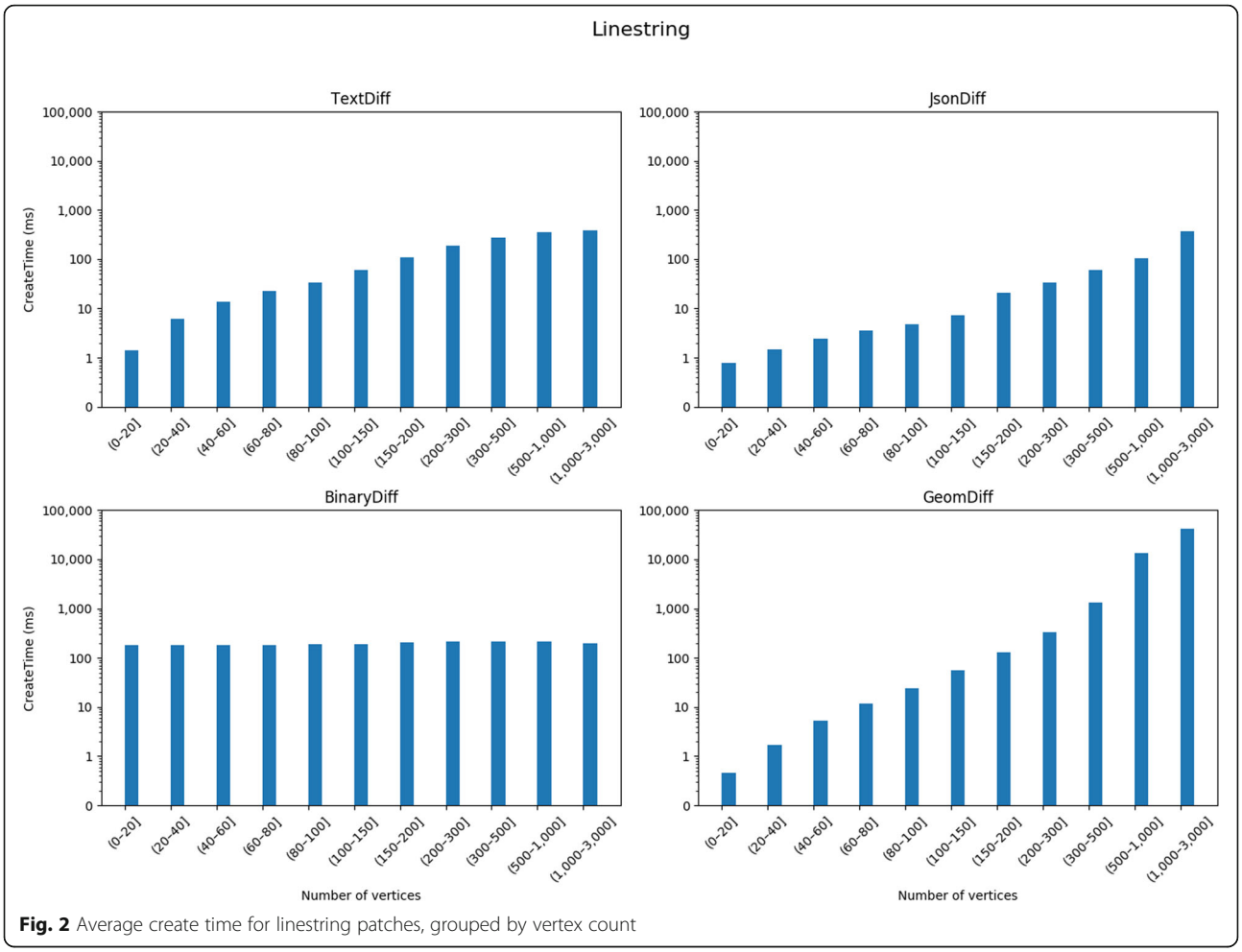

Fig. 2 Average create time for linestring patches, grouped by vertex count

The error rates are low for all algorithms, except for the TextDiff undo algorithm. One possible explanation for these errors are floating-point issues. Since the TextDiff algorithm uses the text based WKT format as an intermediary step, it is possible that some rounding errors have been introduced when the undo operation is applied. However, we have not investigated this issue further.

One shortcoming of our experiment is that the test-dataset did not include multi geometries. This is because the OSM dataset does not contain multi points and multi linestrings, and that multi polygons were considered too time-consuming to create from the OSM data format. However, we suspect that multi-geometries will show results similar to or worse than linestrings and polygons. Since multi-geometries adds more layers to the recursive hierarchy of components, more time will be spent traversing this hierarchy.

The use of a commercial cloud platform as the testbed for our experiment allowed us to test a large number of operations in parallel at a reasonable price. This would have been costly and complex to achieve using on-site hardware. However, each execution of a function app runs on an instance. This instance runs multiple concurrent executions in parallel, which mean that executions may compete for the same CPU resource [28]. This may affect the performance of each execution, compared to running them in complete isolation. However, we argue that the large amount of geometries tested will mitigate this issue and spread the effect evenly.

Conclusions

We have shown that efficient diffing algorithms for geospatial vector data can be created by taking advantage of the native mathematical properties of the data. The GeomDiff algorithm performs comparable to, or better than, the three generic diffing algorithm we have compared it to. However, it suffers from performance degradation as the vertex count increases. In many situations this will not pose a problem, but it is a serious shortcoming that should be addressed.

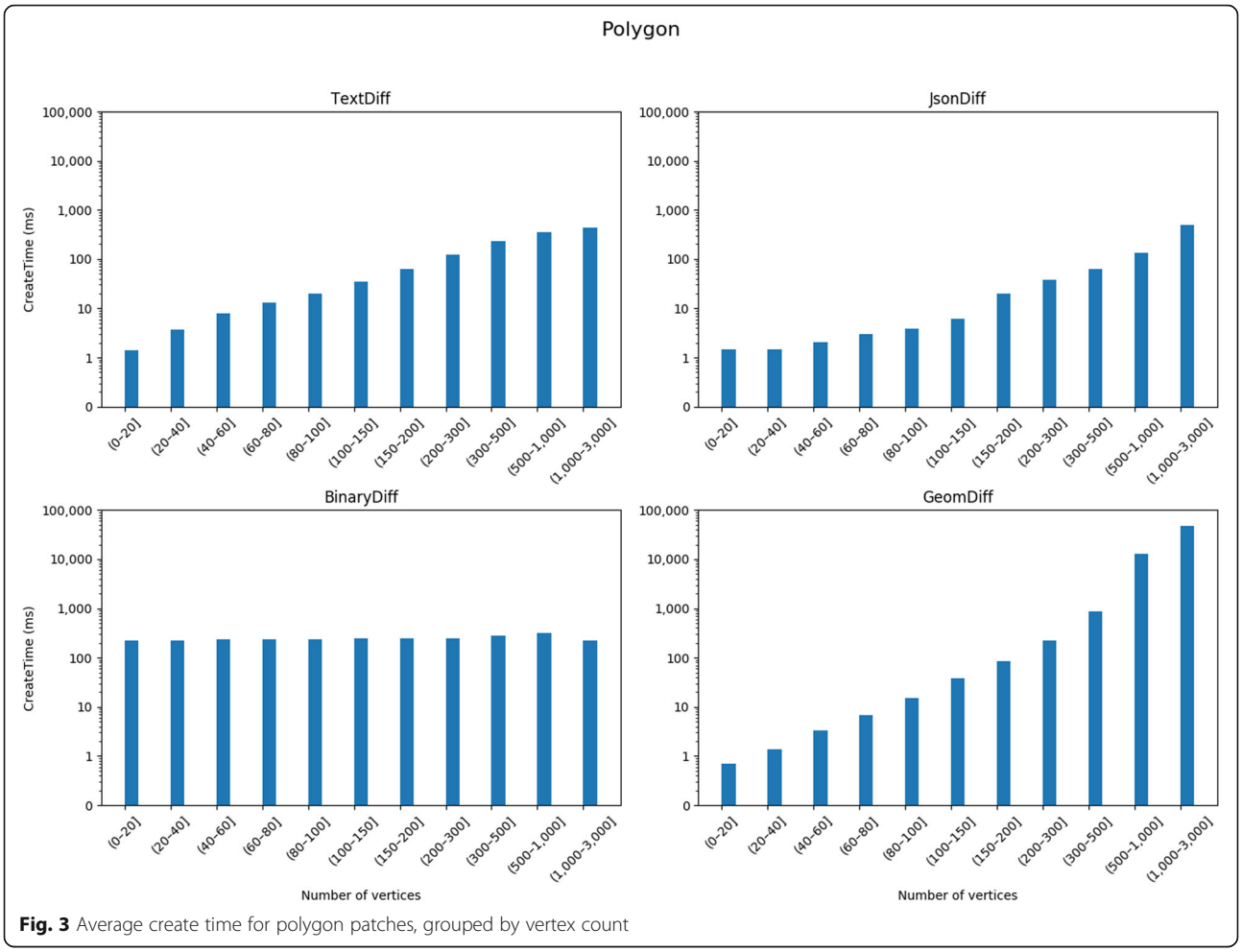

Fig. 3 Average create time for polygon patches, grouped by vertex count

Geospatial diffing formats have several use-cases. Storage of spatiotemporal data is one example. In the "object change" model described by Worboys [29], significant storage reductions can be achieved by storing each change as a diff as opposed to storing the complete, changed version. Geospatial diffs will also be a key concept when designing a system that uses the principles of Event Sourcing [30] to handle geospatial data.

We have not found any published geospatial diffing algorithms in the literature. However, we found some examples from the industry. We suspect that this indicates that if work on this topic has been carried out, it has been done in the industry. Another possibility is that when the need for geospatial diffing has occurred, generic diffing algorithms have been found sufficient. In the light of our findings we question this conclusion, as we have shown that it is possible to create tailored algorithms for geospatial diffing that outperforms generic algorithms.

With the increasing amount of geospatial data being collected, created, and updated we foresee an increased demand for efficient strategies for storage and processing. Event sourcing and object change models are an interesting approach to this challenge. Since geospatial diffing algorithms are a key aspect of these techniques, we encourage more research into this field.

Availability and requirements

- **Project name:** GeomDiff
- **Project home page:** https://github.com/atlefren/GeomDiff
- **Operating system(s):** Platform independent
- **Programming language:** C#
- **Other requirements:** .NET Standard 2.0 compatible .NET implementation
- **License:** BSD-3-Clause
- **Any restrictions to use by non-academics:** No restrictions apart from those imposed by the license.

Abbreviations

JSON: JavaScript Object Notation; OSM: OpenStreetMap; WKB: Well Known Binary; WKT: Well Known Text; XML: Extensible Markup Language

Acknowledgements
The author would like to thank Dr. Birgitte H. McDonagh, Dr. Alexander S. Nossum, and Dr. Terje Midtbø for valuable input during manuscript preparation.

Author's contributions
A.F.S is the sole author of this manuscript, and carried out the implementation of the GeomDiff library, all supporting scripts, performed the statistical analysis, and wrote the manuscript. The author read and approved the final manuscript.

Funding
This work was supported by Norkart AS and The Research Council of Norway [grant number: 261304 l].

Availability of data and materials
The data generated by the benchmark pipeline is available at https://github.com/atlefren/geomdiff_article_results
The OpenStreetMap data used by the benchmark are available at http://download.geofabrik.de [24]
The derived OpenStreetMap geometry pairs used in the benchmark are available from the corresponding author on reasonable request.

Competing interests
The author declare that they have no competing interests.

Received: 11 May 2020 Accepted: 26 June 2020
Published online: 10 July 2020

References

1. Raymond ES. The jargon file, version 4.4.7. 2003 [cited 2019 Nov 11]. Available from: http://catb.org/jargon/html/D/diff.html.
2. Ruparelia NB. The history of version control. SIGSOFT Softw Eng Notes. 2010; 35(1):5–9.
3. D'Angelo G, Di Iorio A, Zacchiroli S. Spacetime Characterization of Real-Time Collaborative Editing. Proc ACM Hum-Comput Interact. 2018;2(CSCW):41 1–41:19.
4. Michelson B. Event-Driven Architecture Overview. Patricia Seybold Group [Internet]. 2006;2. Available from: http://www.customers.com/articles/event-driven-architecture-overview/.
5. Coup R. Sno, our new open source tool for distributed data versioning. Koordinates Blog 2020. Available from: https://koordinates.com/blog/sno-our-new-open-source-tool-distributed-data-versioning/.
6. Singh A. Review article digital change detection techniques using remotely-sensed data. Int J Remote Sens. 1989;10(6):989–1003.
7. Cho J, Ntoulas A. Chapter 45 - Effective Change Detection Using Sampling. In: Bernstein PA, Ioannidis YE, Ramakrishnan R, Papadias D, editors. VLDB '02: Proceedings of the 28th International Conference on Very Large Databases. San Francisco: Morgan Kaufmann; 2002 [cited 2020 Mar 18]. p. 514–25. Available from: http://www.sciencedirect.com/science/article/pii/B9781558608696500524.
8. Hunt JW, MacIlroy MD. An algorithm for differential file comparison. Bell Laboratories Murray Hill; 1976. (Bell Laboratories Computing Science). Report No.: #41.
9. Miller W, Myers EW. A file comparison program. Software: Practice and Experience. 1985;15(11):1025–40.
10. Myers EW. AnO (ND) difference algorithm and its variations. Algorithmica. 1986;1(1):251–66.
11. Chawathe SS, Rajaraman A, Garcia-Molina H, Widom J. Change Detection in Hierarchically Structured Information. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data. New York: ACM; 1996 [cited 2019 Aug 27]. p. 493–504. (SIGMOD '96). Available from: http://doi.acm.org/10.1145/233269.233366.
12. Percival C. Naive differences of executable code. 2003 [cited 2020 Jan 15]. Available from: http://www.daemonology.net/bsdiff/.
13. Nottingham M, Bryan P. JavaScript Object Notation (JSON) Patch. 2013 [cited 2020 Mar 17]. Available from: https://tools.ietf.org/html/rfc6902.
14. Urpalainen J. An Extensible Markup Language (XML) Patch Operations Framework Utilizing XML Path Language (XPath) Selectors. 2008 [cited 2020 Mar 18]. Available from: https://tools.ietf.org/html/rfc5261.
15. Razmjooei S. Tracking, calculating and merging vector changes with Input and QGIS - Lutra Consulting. Lutra Consulting. 2019 [cited 2020 Mar 20]. Available from: https://www.lutraconsulting.co.uk/blog/2019/11/23/input-geodiff/.
16. Duffy L. Boundless GeoGit: A Different Approach to Geospatial Data Management. POB. 2014 [cited 2020 Mar 18]. Available from: https://www.pobonline.com/articles/100258-boundless-geogit-a-different-approach-to-geospatial-data-management?v=preview.
17. ISO. ISO/IEC 13249–3:2016 Information technology — Database languages — SQL multimedia and application packages — Part 3: Spatial. 5th ed. 2016. 1328 p.
18. Haklay M, Weber P. OpenStreetMap: user-generated street maps. IEEE Pervasive Computing. 2008;7(4):12–8.
19. Atle Frenvik Sveen. node-osm-read. 2020. Available from: https://github.com/atlefren/node-osm-read.
20. Sveen AF. BenchmarkFunction. 28.01.20202 [cited 2020 Jan 28]. Available from: https://github.com/atlefren/GeometryDiffBenchmark.
21. NetTopologySuite. 2019. Available from: https://github.com/NetTopologySuite/NetTopologySuite.
22. Gillies S, Butler H, Daly M, Doyle A, Schaub T. The GeoJSON Format. 2016 [cited 2020 Mar 19]. Available from: https://tools.ietf.org/html/rfc7946.
23. Sveen AF. geodiff_stats. 2020 [cited 2020 Apr 10]. Available from: https://github.com/atlefren/geodiff_stats.
24. D'agostino R, Pearson ES. Tests for departure from normality. Empirical results for the distributions of b2 and √b1. Biometrika. 1973;60(3):613–22.
25. Kruskal WH, Wallis WA. Use of ranks in one-criterion variance analysis. J Am Stat Assoc. 1952;47(260):583–621.
26. Conover WJ, Iman RL. On multiple-comparisons procedures. Los Alamos Sci Lab Tech Rep LA-7677-MS. 1979;1:14.
27. Lee Rodgers J, Nicewander WA. Thirteen ways to look at the correlation coefficient. Am Stat. 1988;42(1):59–66.
28. Shilkov M. From 0 to 1000 Instances: How Serverless Providers Scale Queue Processing. Binaris Blog. 2018 [cited 2020 May 8]. Available from: https://blog.binaris.com/from-0-to-1000-instances/.
29. Worboys M. Event-oriented approaches to geographic phenomena. Int J Geogr Inf Sci. 2005;19(1):1–28.
30. Fowler M. Event Sourcing. martinfowler.com. 2005 [cited 2020 Apr 16]. Available from: https://martinfowler.com/eaaDev/EventSourcing.html.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

► Convenient online submission
► Rigorous peer review
► Open access: articles freely available online
► High visibility within the field
► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com