

用于空间连接中负载均衡的高效并行自适应分区算法

蔡泽熙

摘要

随着地理信息数据的增加,以及各种收集手段的精细化,GIS(地理信息系统)的数据呈现出数据量大、结构复杂和种类丰富的特点。而GIS的空间连接是比较复杂和耗时的操作,是将两个多边形图层连接为一个新的图层作为输出。为了能高效处理大型空间数据集,需要利用一定的手段来加速执行,例如并行和分布式等。同时,为了提高并行的效率和细粒度,要对有数据倾斜的数据集进行负载均衡的分区,例如使用MPI等技术。本文实现了空间连接算法,同时基于四叉树分区的自适应分区(ADP)技术,对工作负载进行均衡的分区,进一步提高该算法的性能,并且通过一系列实验,证明了算法的正确性和有效性。

关键词: 空间连接; 自适应分区; 并行

1 引言

在人类历史的长河中,几乎所有的活动都发生在地球上,并且与地理和空间位置密切相关。地理信息可以说是我们日常生活中不可或缺的一部分,例如,我们平时出门时使用的导航和定位功能给我们的生活带来了极大的便利。随着计算机技术的发展和普及,以及人类对各种地理信息的认知越来越复杂,各种地理数据的数量呈增长趋势,其结构和类型也越来越丰富,如何有效地处理这些数据是一个需要解决的问题。

地理信息系统(GIS)是一种具有信息系统空间专业形式的数据管理系统,它是一个具有存储和操作功能的计算机系统。其中,空间连接操作在灾害预测、城市规划等场景中发挥着非常重要的作用。它涉及两个空间层,并根据相交、包含、重叠和其他谓词输出所需的结果。然而,由于数据量大和数据倾斜,导致了空间中的负载均衡问题。为了解决这些问题,我们可以对数据进行分区,并使用并行和分布式技术来提高操作效率。

本文主要研究空间连接操作,其中输入是两层地理空间数据,使用的空间连接分区方法是基于四叉树分区的自适应分区(ADP)技术。与现有的分区技术不同,ADP对空间连接工作负载进行分区,而不是单个数据集,以提供更好的负载平衡。

2 相关工作

2.1 使用并行和分布式集群提高空间连接效率

在使用MPI加速算法方面,曹倩倩等^[1]研究了空间拓扑相交关系的并行化,目标是寻找较优的数据并行划分策略,他们发现利用MPI并行库和基于弧段的数据划分策略,可以有效缩短处理大规模GIS数据的时间。Satish Puri等^[2]对裁剪任意多边形进行了研究,并行化实现了Greiner-Hormann算法,成本与基于连续扫描线的多边形裁剪的时间复杂度相匹配,并且设计了一个实用的GIS系统,即MPI-GIS,这是一个分布式多边形覆盖处理系统,用于Linux集群上的真实多边形数据集;他们也提出了利用MPI-Vector-IO来处理大规模数据节点间通信的问题,在Lustre文件系统进行测试,使用

并行空间计算的 MPI 框架支持在数量级较大的数据集上进行并行空间索引和连接操作，让基于 HPC 的 GIS 能够有效地处理不同格式的大型矢量数据集^[3]。

2.2 不同的 GIS 空间连接算法

Jignesh M. Patel 等^[4]提出了一种新的空间连接算法 PBSM，当连接的两个输入都没有连接属性的索引时，该算法十分有效，使用高效的基于计算几何的平面扫描技术来执行连接。Benjamin Sowell 等^[5]评估了不同的空间连接算法，发现在大多数情况下，从头开始重复计算连接结果的性能都优于使用移动对象索引，使用所有空间维度修剪结果的专用连接方法都优于索引嵌套循环方法以及最初仅在一个维度中进行分区的平面扫描方法，高阶算法效应仍然主导着主存的性能，以及简单的空间分区提供了有限的实用性。Jie Yang 等^[6]提出了避免重复技术，通过该技术可以减少跨越多个网格单元的几何图形的不必要重复，同时设计了一个自适应分区算法，较好地解决了负载均衡的问题，基本思想是利用每个候选对的权重来进行决定，并且通过 OpenMP 来实现并行化，这也是本文实现的算法基础。

2.3 GIS 在不同平台上的执行

Yaming Zhang 等^[7]评估了四个计算几何库，以评估它们对大空间数据勘探中各种工作负载的适用性，即 GEOS、JTS、Esri geometry API 和 GeoLite，使用 Spark 上的微和宏基准测试来评估它们的计算效率和内存使用情况。Sujit Beborrtta 等^[8]介绍了不同的众所周知的可扩展无服务器框架，即用于地理空间大数据管理的 Amazon Web Services（AWS）Lambda、Google Cloud Functions 和 Microsoft Azure Functions，讨论了一些常用于分析地理空间大数据的现有方法，并指出了它们的局限性，以及提出的框架在云地理信息系统平台中的适用性。

3 本文方法

3.1 本文方法概述

3.1.1 GIS 中的文件存储

WKT（Well-known text）是一种表示矢量几何对象的文本标记语言，可以用来表示点、线、多边形等几何对象。通常，一个完整图层的所有 WKT 信息存储在一个 csv 文件中，不同的 csv 文件表示不同的图层。图 1 展示了不同几何对象的 WKT 表示。



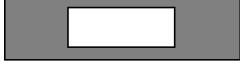
几何对象	几何图形	WKT格式
点		POINT(6 4)
线		LINESTRING(8 19,19 50,36 25)
多边形		POLYGON((1 1,9 1,9 1 9,1 1),(3 3,3 5,5 5,5 3,3 3))

图 1: 常见几何对象的 WKT 表示

3.1.2 最小边界矩形（MBR）

最小边界矩形（minimum bounding rectangle, MBR）是一个二维图形（例如点、线、多边形）的最大范围，即以给定的二维图形所有顶点的最大横坐标、最小横坐标、最大纵坐标、最小纵坐标为边界的矩形。MBR 通常用于大致表示出一个地理对象的位置，在进行空间查询和建立 R 树时需要用到

MBR。图 2展示了一个 MBR 的示例，其中黑边为一个几何对象，红边为其 MBR。

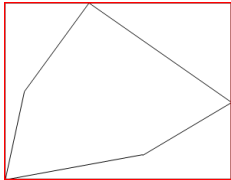


图 2: MBR 示例

3.1.3 R 树

R 树是用于空间数据存储的树形数据结构，树上的每个节点表示为对应几何对象的 MBR，核心思想是将距离较近的节点逐渐聚合。为了保证查询效率，R 树还是一种高度平衡树，类似于 B 树，其叶节点中的索引记录包含指向数据对象的指针^[9]。

图 3展示了一个简单的二维矩阵上 R 树的示例，黑色矩形表示原始图形的 MBR，而绿色和红色矩阵表示聚合之后的虚拟矩阵。其中，插入操作是指将要插入的几何对象的 MBR 插入 R 树，并且每次插入后都要保证平衡等特征。搜索操作是指给定一个搜索矩阵，从根节点开始遍历 R 树，如果某节点与搜索矩阵相交，那么就继续往下搜索，输出所有与搜索矩阵相交的节点，即搜索过程中所有比较过的矩阵。

假设图中的橙色框为查询矩阵，根据上述步骤，首先找到其与 R14 和 R15 相交，因此这两个几何对象的 MBR 便在输出队列中。同理，继续往下搜索，对于 R 树的第二层节点，可以发现其与 R10、R11 和 R12 相交。最后对第三层节点进行搜索，可以发现其与 R2、R5 和 R6 相交，因此上述提到的 8 个 MBR 均作为 R 树查询的结果输出，用于后续处理。

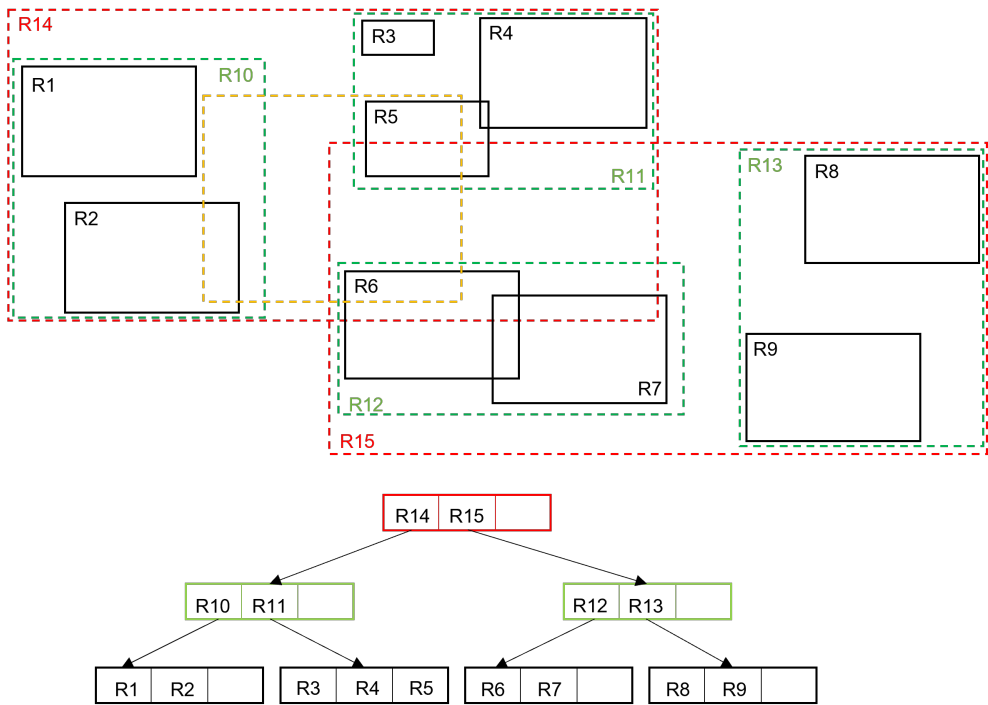


图 3: R 树示例

3.1.4 GEOS

GEOS 是一个 C/C++ 库，常用于 GIS 软件对二维矢量几何进行操作^[10]，同时，该库也提供空间数据索引、基于几何的算法以及空间数据解析。该库提供许多空间模型和功能，在几何方面，包括点、线、多边形等；在谓词方面，包括相交、接触、重叠等；在操作方面，包括并集、距离、交集等。除了上述功能外，还提供各种几何图形、STR 空间索引以及 WKT 的编码器和解码器，因此本文选择 C++ API 来实现空间连接算法，下面将会选择三个较常用的类进行介绍。

GEOS 的 io 类是 JTS 对象与其他格式相互转换的接口。本文需要用 WKTRReader 类将输入的 WKT 语言转换为 geom 类，方便后续使用对应的函数，实现空间连接算法。

GEOS 的 geom 类包含 Geometry 接口层次结构。本文需要用到 Geometry 类和 Envelope 类。其中，Geometry 类由 GeometryFactory 构造和销毁，getEnvelopeInternal 函数返回此 Geometry 的 MBR，getNumPoints 函数返回此 Geometry 的顶点数。Envelope 类定义了二维坐标平面的矩形区域，intersection 函数计算两个 Envelope 的交集，也就是判断两个几何对象的 MBR 是否相交。

GEOS 的 index 类为各种空间索引提供类。本文需要用 STRtree 类，它使用 STR 算法创建仅查询的 R 树，insert 函数将具有给定 Envelope 指定范围的空间项添加到索引，query 函数查询其范围与给定搜索矩阵相交的所有项目的索引，正如在 3.1.3 节举的例子一样。

3.1.5 空间连接算法

空间连接算法在 GIS 中是一个十分常见的操作，其涉及两个图层，这里假设为 R 和 S。在 R 和 S 上，算法根据谓词执行空间连接查询，例如相交、包含和重叠，结果返回候选对集合 (r, s) ，其中 $r \in R$ ， $s \in S$ ，且满足连接谓词^[6]。而空间连接又可以分为两个阶段，分别为过滤阶段和细化阶段。在过滤阶段，根据输入的两个图层，计算它们每个几何对象的 MBR，并且生成候选对集合，这些候选对由跨层几何对象构成，它们的 MBR 具有空间重叠，这样做的好处是减少了细化阶段的工作量。显然，如果两个几何对象的 MBR 重叠，那么它们可能实际上有重叠；而如果两个几何对象的 MBR 没有重叠，那么它们一定不可能重叠。在细化阶段，只需要对这些候选对集合进行操作，而不需要关注整个图层，提高了算法效率。

总的来说，算法的输入为两个几何图层 R 和 S，输出是根据谓词执行的空间连接查询结果，在执行的过程中，目标是生成一个负载均衡和自适应的分区。本算法首先找到所有候选对，然后对候选对进行分区，而不是对每一层的数据进行分区，优点是工作负载感知分区以及减少进程间通信^[6]。下面将会分别介绍这两个阶段的具体实现。

3.2 过滤阶段

在过滤阶段，主要思想是寻找两个图层 MBR 相互重叠的候选对集合，给细化阶段减少工作量。算法 1 为该阶段的伪代码，大致思想是用 R 的 MBR 来建立 R 树，然后用 S 的 MBR 来查询该 R 树，找到候选对集合，并且记录每个候选对的中心点和权重作为属性。中心点的作用是，在细化阶段处理的时候，需要定位该候选对，而定位的依据就是该候选对相交部分的中心点。同理，需要计算两个 MBR 交集的权重，便于后续分区，设两个几何对象的顶点数分别为 m 和 n ，那么权重 $w = (m+n)\log(m+n)$ ^[6]。

Procedure 1 Filter Phase.

Input: Two spatial objects R and S **Output:** Candidate set denoted by C Build Rtree index RI using MBRs of R **for** s_j **in** S **do** **for** r_i **in** $RI.query(s_j.MBR)$ **do** Find the intersection of $r_i.MBR$ and $s_j.MBR$.

Calculate center point of intersection denoted

 by p_{jk} . Calculate weight w_{ij} using weight equation. $C = C \cup tuple(r_i, s_j, p_{ij}, w_{ij})$ **end****end**

图 4 是算法 1 执行的一个示例，假设图层 R 具有 $R1$ 和 $R2$ 两个几何对象，图层 S 具有 $S1$ 、 $S2$ 和 $S3$ 三个几何对象，那么经过算法 1 的执行，输出的候选对为 $(R1, S1)$ 、 $(R1, S2)$ 和 $(R2, S3)$ 。要注意的是，虽然 $R2$ 和 $S3$ 并没有实际相交，但是由于是根据其 MBR 来判断是否相交的，因此也作为一个候选对进入细化阶段。经过算法 1 的处理后，一些确定不相交的几何对象的权重为 0，例如 $R2$ 和 $S1$ ，那么在后面进行分区的时候就不用考虑这些几何对象，提高了算法的效率。在分区的时候，并不是依据整个候选对所处的位置来进行分区的，而是根据中心点所在的单元格进行分区。例如候选对 $R1$ 和 $S2$ ，其相交的部分为 $S2$ 的 MBR，但是 $S2$ 的 MBR 却占据了 4 个单元格，显然其中心点在第三行第三列的单元格中，因此在后续分区的时候，只需要假设该候选对位于第三行第三列的单元格中，而不需要关注整个相交的部分。

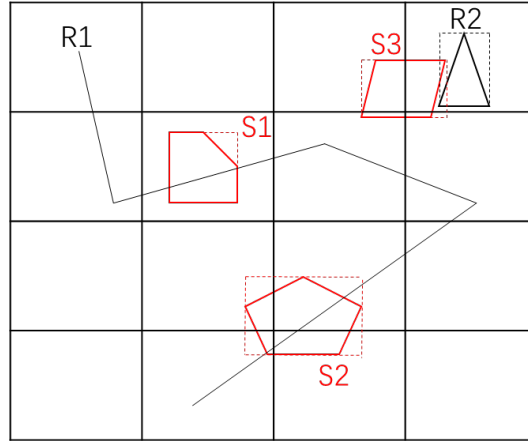


图 4: 算法 1 执行示例

3.3 细化阶段

在细化阶段，主要是多线程对候选对进行分区，并且实现负载均衡的目标，高效进行空间连接查询。算法 2 为该阶段的伪代码，大致思想是将单元格按权重降序进行排序，不断地对权重大的单元格进行再分区，每次都选择权重最大的单元格，并且利用四叉树分区，即将某个大单元格按面积平均分成四个小单元格，再对每个小单元格分配对应的候选对，不断进行上述步骤，直至达到用户想要的分区数。

该算法的输入为算法 1 找到的候选对集合 C 、用户定义要生成的分区数 N 、GlobalMBR、OpenMP 的线程数 P 、暂存四叉树分区结果的队列 Q 、记录当前执行分区的线程数 $counter$ 以及阈值 T 。其中，GlobalMBR 为包含 R 和 S 所有几何对象的 MBR，其权重为所有候选对权重之和；阈值 T 用于避免生成的分区数大于 N ，在本算法中初始化为 $4 \times P$ ，原因是对于 P 个线程，四叉树分区每次最多生成 $4 \times P$ 个子单元格。初始化 G 为 GlobalMBR，其中 G 为按权重降序排序的所有单元格列表；初始化 R 为空

集，其中 R 记录每次线程组执行后分区的子单元格。

当分区数量小于 $N-T$ 时，不断对单元格进行分区， P 个线程中有 $counter$ 个线程进行了四叉树分区，每次循环都初始化为 0。然后，让主线程遍历前 P 个权重最大的单元格。定义 k 为分割因子，用于控制每次执行的精度和并行度^[6]，显然 $k \geq 1$ 。根据上面的分析， G 为按权重降序排序的单元格列表，因此第 0 个单元格为权重最大的单元格。当第 i 个线程候选对的权重 \geq 最大权重的 k 分之一，就执行四叉树分区，此时该单元格将会按面积平均分为四个子单元格，并且通过 “#pragma omp task” 语句在线程组中挑选一个线程执行四叉树分区，需要将原单元格的候选对对应到新的子单元格，判断依据为每个候选对的中心点位于哪个子单元格中。

经过上述操作后，队列 R 存储着所有分区后的子单元格的信息。由于每次迭代任务的数量可能不同，因此 $counter$ 就需要记录有多少个单元格被分区。由于 G 中的单元格是按权重降序排序的，即若第 i 个单元格没分区，那么显然第 $i+1$ 、 $i+2 \dots$ 个单元格就一定不会被分区，因此 $counter-1$ 为最后一个被分区的单元格下标。此时需要更新 G 中的单元格，将 G 中被分区的单元格从列表中删除，并且将 R 中所有子单元格加入 G 中，最后按权重对 G 中的单元格进行降序排序，直到分区数量小于 $N-T$ 为止。

显然，此时大部分的单元格已经大致实现了负载均衡，这时候只需要对 G 中所有权重不为 0 的单元格分别分配一个线程，执行空间连接操作。为叙述方便，本文假设空间连接算法中使用 GEOS 库的 intersects 和 intersection 操作，其中 intersects 用于查找两个几何图形是否相交，intersection 用于查找两个几何图形的相交区域。

Procedure 2 Refinement Phase.

Input: Candidate collection C , target number of cells N , $GlobalMBR$, maximum OpenMP tasks P , queue R , number of OpenMP tasks $counter$, threshold T

Output: A list of grid cells G

$G = GlobalMBR$

$R = \emptyset$

while number of cells less than $N - T$ **do**

$counter = 0$

#pragma omp parallel num_threads(P)

{

#pragma omp master

{

for $i = 0$ to $P - 1$ **do**

if $G[i].weight \geq G[0].weight/k$ **then**

$counter++$ #pragma omp task $R[i] = \text{Quadtree partition on } G[i]$

end

end

}

}

#pragma omp taskwait $G.delete(0, counter - 1)$

$G = R$ $R = \emptyset$ $G.sort()$

end

$sum = \text{number of cells in } G \text{ whose weight is not } 0$

#pragma omp parallel num_threads(sum) intersects/intersection

图 5 是算法 2 执行的一个示例，假设算法 1 已经找到了所有的候选对，且总权重为 270，目标分区数 N 为 40，OpenMP 最大任务数 P 为 5，阈值 T 为 20，分割因子 k 为 2。在第一次迭代中，只需要对 $GlobalMBR$ 进行分区，假设分区结果如图 5(a) 所示，此时一共有 4 个单元格，按权重降序排序

后每个单元格的权重分别为 90、80、70、30。在第二次迭代中，由于此时的权重最大为 90，而 k 为 2，因此权重大于等于 45 的单元格都要被分区，假设分区结果如图 5(b) 所示，此时一共有 13 个单元格，按权重降序排序后前 5 个单元格的权重分别为 60、32、32、30、22。在第三次迭代中，由于此时的权重最大为 60，而 k 为 2，因此在前 5 个单元格中，权重大于等于 30 的单元格都要被分区，即前 4 个最大的单元格需要分区，假设分区结果如图 5(c) 所示，此时一共有 25 个单元格，按权重降序排序后前 5 个单元格的权重分别为 22、20、20、20、20。由于 $N-T$ 的值为 20，且此时分区数目 25 大于 20，因此停止迭代，此时已经完成负载均衡的目标，只需要给每个分区分配一个处理器进行空间连接查询。注意到有一些权重为 0 的分区，此时可以安全忽略这些分区，因为它不含任何候选对。

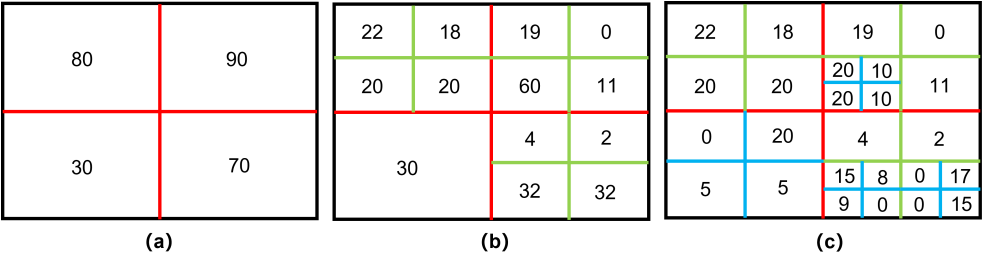


图 5: 算法 2 执行示例

4 复现细节

4.1 与已有开源代码对比

复现过程中没有参考任何相关源代码，对 Jie Yang 等^[6]论文中的“ADAPTIVE PARTITIONING”部分进行了复现。

4.2 实验环境搭建

本实验使用的操作系统为 Ubuntu 20.04，采用的 GEOS 库版本为 3.9.1，使用 C++ API。为了测试空间连接算法的正确性和有效性，本文采用了 TIGER 2015 数据集^[11]，该数据集是从美国人口普查局的 TIGER 文件中提取的。这些 csv 文件的每一行都包含一个 WKT 格式表示的形状，后跟该记录的其它信息，只需要对形状信息进行处理。数据集的信息如表 1 所示，显示了实验所用到的 7 个数据集的描述、文件大小以及每个数据集所包含的几何对象类型和数目。

表 1: 数据集信息

数据集	描述	大小	记录
STATE	州	17MB	56 个多边形
PRIMARYROADS	主要道路	52MB	12396 条线
RAILS	铁路	79MB	1.81 亿条线
AREALM	区域地标	140MB	129K 个多边形
COUNTY	县	149MB	3233 个多边形
ZCTA5	5 位邮政编码制表区	1GB	33144 个多边形
AREAWATER	区域水体	2.1GB	2.3M 个多边形

4.3 界面分析与使用说明

parallel_intersect.cpp: 空间连接算法的并行实现，编译及运行的命令为“g++ parallel_intersect.cpp -o parallel_intersect -lgeos -fopenmp && ./parallel_intersect”。输入的数据集为 TIGER 2015，需要修改代码的第 53 行作为输入的数据集 R 的路径，第 87 行作为输入的数据集 S 的路径，第 269 行作为目标分

区数，第 270 行作为 OpenMP 最大任务数，第 271 行作为阈值，第 272 行作为分割因子。输出分别为算法 1 数据集 R 处理时间、算法 1 数据集 S 处理时间、算法 2 分区时间、算法 2 理论并行 intersects 时间、算法 2 理论并行 intersection 时间，单位为秒，如图 6 所示。

sequential_intersect.cpp:空间连接算法的串行实现,编译及运行的命令为“g++ sequential_intersect.cpp -o sequential_intersect -lgeos && ./sequential_intersect”。输入的数据集为 TIGER 2015，需要修改代码的第 32 行作为输入的数据集 R 的路径，第 51 行作为输入的数据集 S 的路径。输出分别为算法 1 数据集 R 处理时间、算法 1 数据集 S 处理时间、串行 intersects 时间、串行 intersection 时间，单位为秒，如图 7 所示。

```
czx@czx-virtual-machine:~/桌面/代码$ ./parallel_intersect
1.04783
3.62525
21.9938
0.108885
0.115101
```

图 6: 并行空间连接算法的输出示例

```
czx@czx-virtual-machine:~/桌面/代码$ ./sequential_intersect
1.08768
2.86267
100.312
78.4473
```

图 7: 串行空间连接算法的输出示例

5 实验结果分析

首先，我们对算法 1 进行测试，以说明该算法的有效性。任意选取两个数据集作为输入，例如图层 R 为 RAILS，图层 S 为 COUNTY，输出的部分候选对信息如图 8 所示。以第二行为例，(r112980, s3183) 说明该候选对来自图层 R 的第 112980 个几何对象以及图层 S 的第 3183 个对象，这两个几何对象的 MBR 交集的中心点为 (-122.769, 45.6392)，权重为 16047.1，说明算法 1 可以正确执行。

```
(r112814, s3183), center: (-122.75, 45.6358), weight: 16021.1
(r112980, s3183), center: (-122.769, 45.6392), weight: 16047.1
(r112314, s3183), center: (-122.843, 45.6444), weight: 16073
(r112863, s3183), center: (-122.85, 45.647), weight: 16099
(r176804, s3183), center: (-122.846, 45.6737), weight: 16827.3
(r112643, s3183), center: (-122.869, 45.6926), weight: 16419.4
(r89530, s3183), center: (-123.403, 45.7068), weight: 16168.2
(r89529, s3183), center: (-123.421, 45.7071), weight: 16124.9
(r37630, s3183), center: (-123.383, 45.7108), weight: 16185.5
(r37631, s3183), center: (-123.436, 45.7133), weight: 16150.9
(r107123, s3183), center: (-122.745, 45.7152), weight: 18381.4
(r177492, s3183), center: (-122.745, 45.7152), weight: 18381.4
(r152644, s3183), center: (-122.857, 45.7507), weight: 17279.9
(r152515, s3184), center: (-76.2071, 36.6788), weight: 4858.51
(r152491, s3184), center: (-76.0422, 36.8006), weight: 3904.8
```

图 8: 串行空间连接算法的输出示例

我们还对算法 1 的执行时间进行测量，对每步操作的时间都进行细化。显然，对于输入的数据集 R，需要将 csv 文件的每一行 WKT 信息转化成 geom、计算 geom 的 MBR 以及建立 R 树。经过测试发现，这三部分主要的时间都用在读入文件时，从 WKT 转换到 geom 上，整体趋势都是数据集越大，总时间就更多，如图 9 所示。

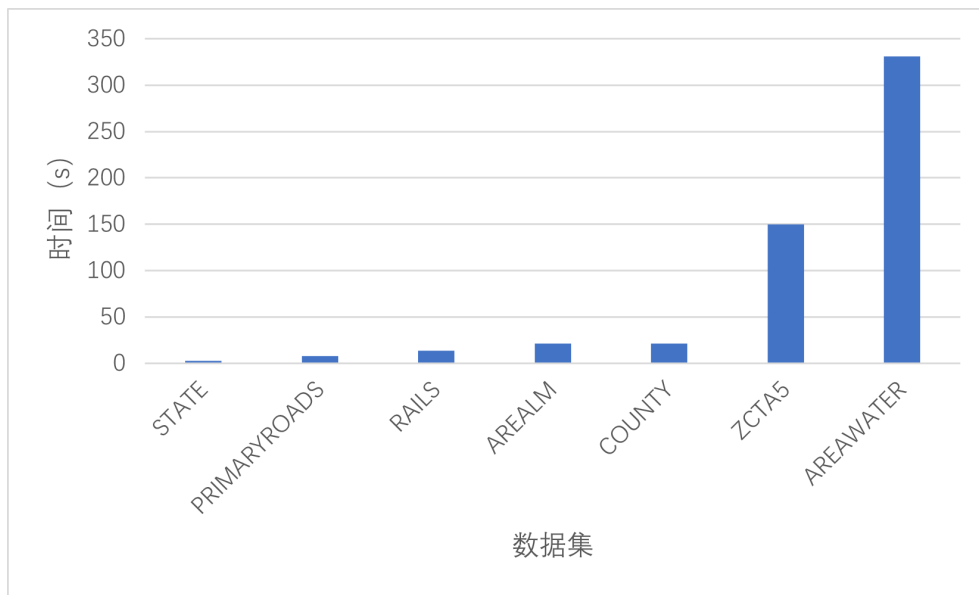


图 9: 数据集 R 将 WKT 转换成 geom 的时间

此时，我们固定数据集 R 为 ZCTA5，改变数据集 S，测量算法 1 中用 S 的 MBR 对 R 的 R 树进行查询以及计算交集所有信息的时间，结果如图 10 所示。可以看到 R 树的查询以及计算交集信息的时间都较少，且计算交集信息的时间随着 R 树查询时间的增加而增加。根据图 9 的 ZCTA5 测量时间，可以得知算法 1 主要时间都用在将 WKT 转换为 geom 上。同时，R 树查询以及计算交集信息的时间并不完全跟数据集 S 的大小有关，推测是和数据的分布有关，因为两个数据集的数据重叠部分越多，需要查询的次数就越多。

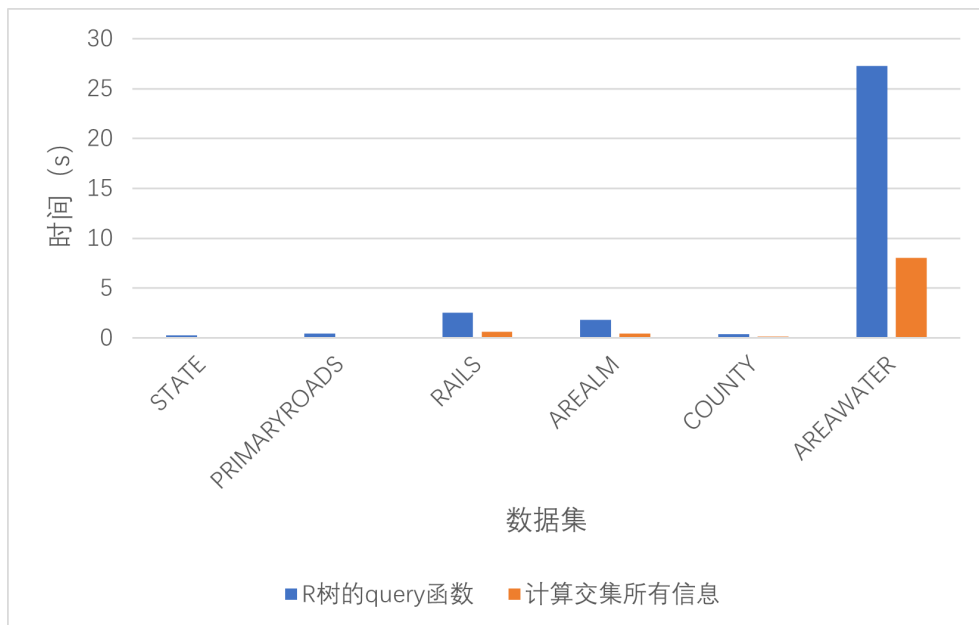


图 10: 数据集 S 在算法 1 的运行时间

我们对算法 2 进行测试，以说明该算法的有效性。同样选取输入的图层 R 为 RAILS，图层 S 为 COUNTY，输出的部分权重信息如图 11 所示。以第三行为例，分区列表 G 的第 29454 个分区的权重为 227757，可以看到每个分区的权重都是大致相等的，实现了负载均衡，说明算法 2 可以正确执行。

29452	227879
29453	227787
29454	227757
29455	227752
29456	227704
29457	227701
29458	227672
29459	227626
29460	227611
29461	227567
29462	227532

图 11: 算法 2 输出结果

为了便于后续实验的测量，我们规定 6 组输入作为测试，每组输入的信息如表 2 所示，其中总顶点数为两个数据集所有几何对象的顶点数之和。

表 2: 输入数据集的信息

编号	数据集 R	数据集 S	总顶点数	候选对数量	候选对总权重
1	STATE	COUNTY	82979487	5877	1.44853e+09
2	PRIMARYROADS	COUNTY	9981513	32223	8.19832e+08
3	RAILS	AREALM	9958611	58717	1.26435e+08
4	COUNTY	ZCTA5	59680078	88243	3.58289e+09
5	PRIMARYROADS	ZCTA5	54808497	88953	1.23713e+09
6	PRIMARYROADS	AREALM	8984157	99059	4.24169e+08

我们对算法 2 负载均衡部分的执行时间进行测量，规定目标分区数 N 为 10000，OpenMP 最大任务数 P 为 4，阈值 T 为 16，分割因子 k 为 2，结果如图 12 所示。从图中可以看出，删除 G 前 $counter$ 个元素和将 R 的元素赋值给 G 花费时间较少，相对其它两项可以忽略不计。对 G 按权重降序的时候，可以看到每组的时间都比较相近，说明其时间可能与数据集大小和权重关系不大，而与用户定义的目标分区数有关。

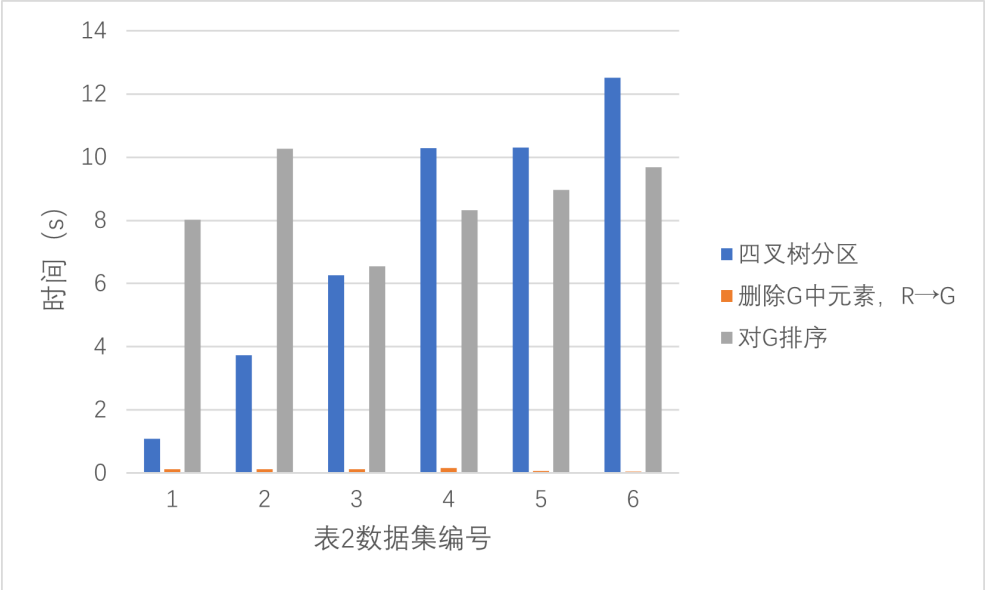


图 12: 不同数据集的算法 2 分区时间

但是，对于第 6 组测试，其数据集大小和候选对总权重比第 4 组和第 5 组小得多，但是算法 2 分区的时间却比它们多，可以知道算法 2 分区的时间跟候选对数量有关，图 13 显示了候选对数量和分区时间的关系。

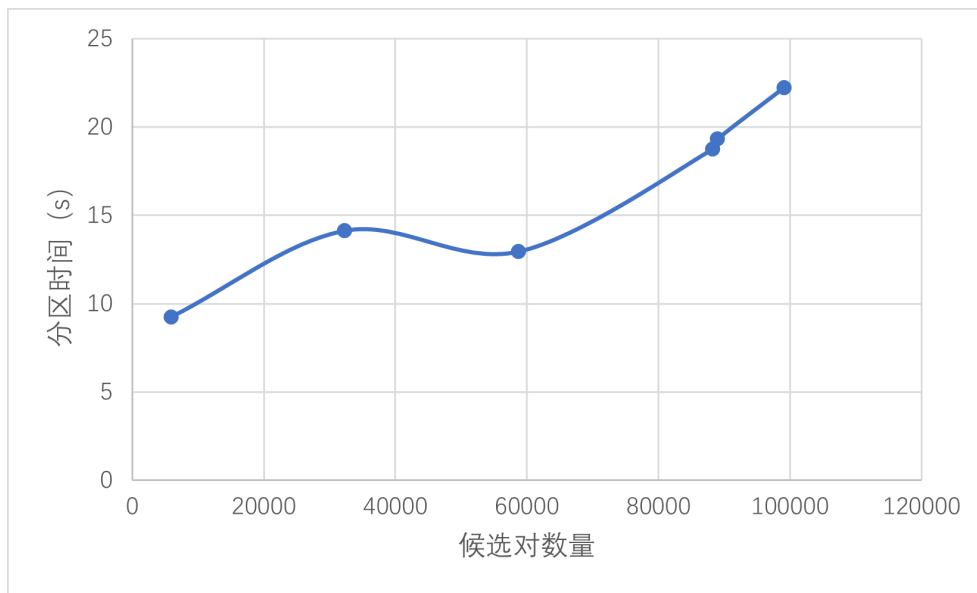


图 13: 候选对数量和分区时间的关系

在空间连接操作部分，我们对 `intersects` 和 `intersection` 两个函数进行测试，比较这两个函数的运行时间。通过算法 1 找到所有候选对后，使用串行算法对每个候选对进行空间连接操作，结果如图 14 所示。虽然 `intersects` 只判断两个几何对象是否相交，而 `intersection` 需要返回相交的部分，但是 `intersects` 的时间并不总是小于 `intersection`。当候选对总权重小于 $8e+08$ 的时候，`intersection` 的时间小于 `intersects`，例如图中的第 3 和第 6 组；而当候选对权重大于 $8e+08$ 的时候，`intersection` 的时间大于 `intersects`。同时，这两个操作的时间跟数据集大小和候选对数量的关系较小，跟候选对总权重大致成正比。查看源代码发现，`intersects` 函数需要计算两个几何对象之间的拓扑关系，需要用到 DE-9IM 模型，猜测该部分的计算需要花费一定的时间。

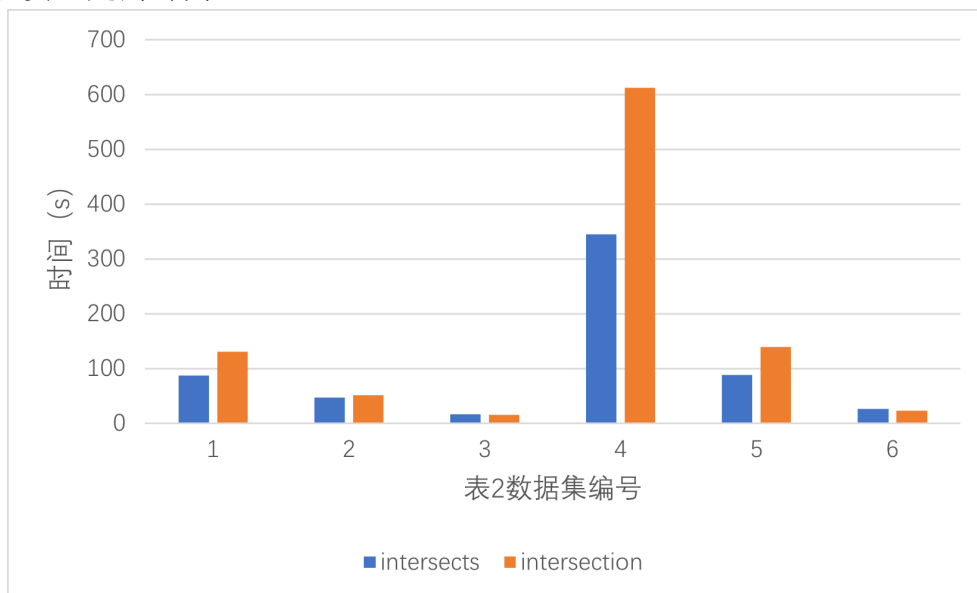


图 14: 不同数据集的 `intersects` 和 `intersection` 时间

我们对算法 2 负载均衡的有效性进行说明，分别对上述两个空间连接操作串行和并行执行，规定目标分区数 N 为 2000，OpenMP 最大任务数 P 为 4，阈值 T 为 16，分割因子 k 为 2，结果如图 15 所示。从图中可以看出，在分区数为 2000 的时候，两种操作并行的时间比串行的时间都提升了许多，说明负载均衡算法是有效的，可以正确对数据负载进行分区，提高算法效率。

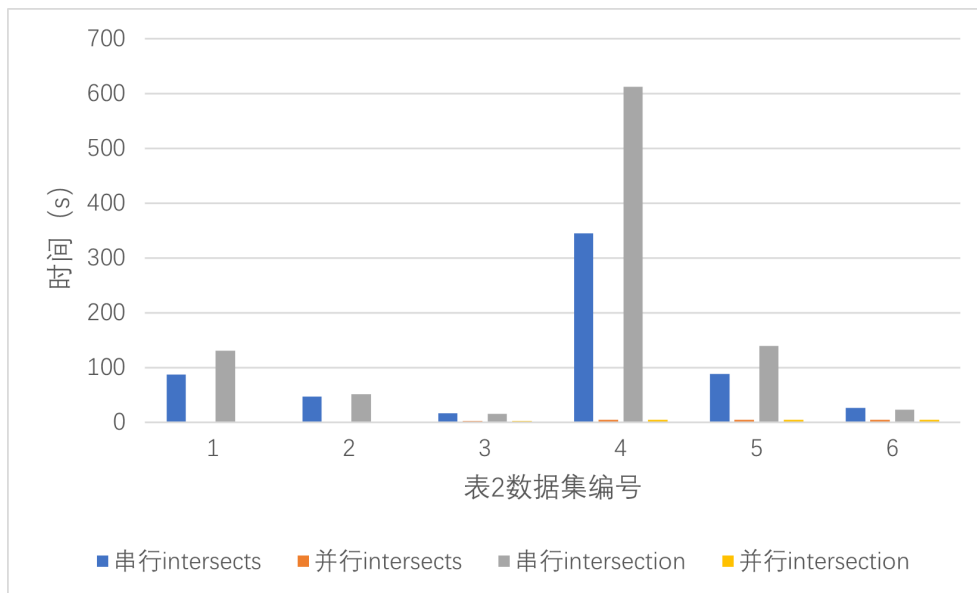


图 15: 不同数据集 intersects 以及 intersection 的串行和并行时间比较

为了说明分区数对并行执行时间的影响，以第 3 组数据为例，即输入为 RAILS 和 AREALM，我们分别改变目标分区数 N 为 200 和 20000，其它条件跟上面一样，测量分区数对负载均衡以及空间连接时间的影响，结果如图 16 所示，纵坐标为时间的对数刻度。显然分区数并不是越多越好，在该数据集中分区数为 200 的执行效率最高，而当分区数为 20000 时，其分区时间比串行执行空间连接都要长不少。但是对于不同的数据集，经过测试后发现，分区数也不是越少越好，因为此时可能会导致每个处理器分配到的候选对相对较多，而每个处理器执行空间连接的次数也会较多，相对于分区花费更多的时间，因此如何选择最合适的分区数也是可以深入研究的课题。

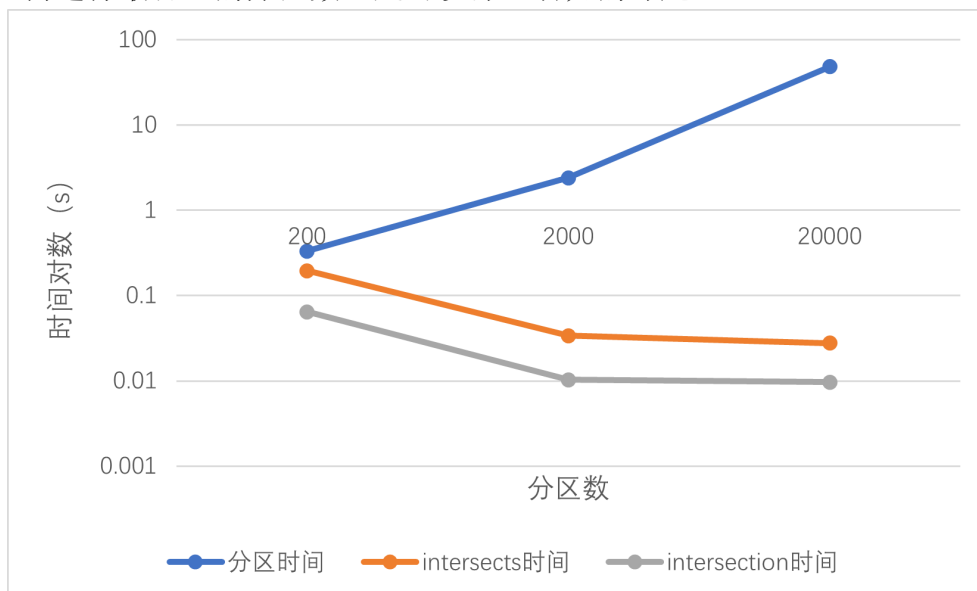


图 16: 不同分区数的分区时间、intersects 以及 intersection 时间比较

6 总结与展望

本文对地理信息系统中的空间连接算法进行了实现和优化，解决了数据负载不均衡的问题，并且通过并行技术，提升了算法的运行效率。空间连接算法分为过滤阶段和细化阶段，输入为两个图层，输出为空间连接操作的结果。我们使用的输入图层为 csv 文件，其包含的 WKT 信息是 GEOS 库不同函数运行的基础。

在过滤阶段，对输入的两个数据集进行处理，找到所有候选对集合，这些候选对由跨层几何对象组成，它们的 MBR 具有重叠部分。在细化阶段，实现了自适应分区技术，根据候选对的权重，通过四叉树分区实现了负载均衡，即分区后每个处理器的权重大致相等。最后给每个权重不为 0 的分区分配一个处理器，每个处理器之间是并行处理的，并且只需要处理它自己分到的候选对，处理器间不会相互干扰。

我们对实现的算法进行了测试，以说明算法的正确性和有效性。对于算法 1，我们发现时间主要用于将 WKT 信息转换为 geom，整体趋势是随着数据集的增大，时间也会不断增加。对于算法 2 的分区部分，我们发现运行时间跟候选对数量成正比，时间主要用于四叉树并行分区。对于空间连接操作，我们对比了 intersects 和 intersection 函数，发现运行时间跟候选对总权重成正比。最后我们测试了算法的有效性，发现并行执行比串行执行效率提高了很多，实现了负载均衡的目标。

该算法还有以下问题可以进行研究和探讨。在进行空间连接操作的时候，我们只对比了 intersects 和 intersection 操作，并没有比较其它的空间连接操作，以后可以对其它空间连接操作进行分析。这两个操作的运行时间也存在着一定的差异，我们只是大致分析了源代码，后续可以仔细研究源代码，对每个部分进行时间测试，找到耗时最大的地方。

实现算法 2 分区部分的时候，我们使用了分割因子 k，在本文的实验中我们都将其设置为 2，并没有设置为其它的值，在之后可以改变不同的 k 值，测试算法 2 的时间。对于不同的数据集输入，也可能会有不同的最优 k 值。同理，目标分区数 N 的值设置为太小或太大都不一定是最好的，不同的数据集也存在着不同的最佳分区数。如何自适应找到不同数据集输入的最优 k 值和 N 值是后续值得研究的一个问题。

参考文献

- [1] 曹倩倩, 李文杰. 空间拓扑相交关系计算算法并行化研究[J]. 天津理工大学学报, 2016, 32(5): 12-15.
- [2] PURI S, PRASAD S K. A parallel algorithm for clipping polygons with improved bounds and a distributed overlay processing system using mpi[C]//2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. 2015: 576-585.
- [3] PURI S, PAUDEL A, PRASAD S K. MPI-Vector-IO: Parallel I/O and partitioning for geospatial vector data[C]//Proceedings of the 47th International Conference on Parallel Processing. 2018: 1-11.
- [4] PATEL J M, DEWITT D J. Partition based spatial-merge join[J]. ACM Sigmod Record, 1996, 25(2): 259-270.
- [5] SOWELL B, SALLES M V, CAO T, et al. An experimental analysis of iterated spatial joins in main memory[J]. Proceedings of the VLDB Endowment, 2013, 6(14): 1882-1893.
- [6] YANG J, PURI S. Efficient parallel and adaptive partitioning for load-balancing in spatial join[C]//2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). 2020: 810-820.
- [7] ZHANG Y, ELDAWY A. Evaluating computational geometry libraries for big spatial data exploration

[C]//Proceedings of the Sixth International ACM SIGMOD Workshop on Managing and Mining Enriched Geo-Spatial Data. 2020: 1-6.

- [8] BEBORTTA S, DAS S K, KANDPAL M, et al. Geospatial serverless computing: Architectures, tools and future directions[J]. ISPRS International Journal of Geo-Information, 2020, 9(5): 311.
- [9] GUTTMAN A. R-trees: A dynamic index structure for spatial searching[C]//Proceedings of the 1984 ACM SIGMOD international conference on Management of data. 1984: 47-57.
- [10] GEOS contributors. GEOS coordinate transformation software library[A/OL]. Open Source Geospatial Foundation. 2021. <https://libgeos.org/>.
- [11] ELDAWY A, MOKBEL M F. Spatialhadoop: A mapreduce framework for spatial data[C]//2015 IEEE 31st international conference on Data Engineering. 2015: 1352-1363.