分类号：_____ 密级：_____

UDC：_____ 编号：_____

工学硕士学位论文

加密数据的 k 近邻查询算法研究

硕士研究生：辛立杰

指导教师：张志强 教授

学位级别：工学硕士

学科、专业：软件工程

所在单位：计算机科学与技术学院

论文提交日期：2016 年 1 月

论文答辩日期：2016 年 03 月

学位授予单位：哈尔滨工程大学

Classified Index:

U.D.C:

A Dissertation for the Degree of M. Eng

Research on k Nearest Neighbor Query over Encrypted Data

Candidate: Xin Lijie

Supervisor: Prof. Zhang Zhiqiang

Academic Degree Applied for: Master of Engineering

Specialty: Software Engineering

Date of Submission: January, 2016

Date of Oral Examination: March, 2016

University: Harbin Engineering University

哈尔滨工程大学
学位论文原创性声明

本人郑重声明：本论文的所有工作，是在导师的指导下，由作者本人独立完成的。有关观点、方法、数据和文献的引用已在文中指出，并与参考文献相对应。除文中已注明引用的内容外，本论文不包含任何其他个人或集体已经公开发表的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者（签字）：辛云龙

日期：2016 年 3 月 21 日

哈尔滨工程大学
学位论文授权使用声明

本人完全了解学校保护知识产权的有关规定，即研究生在校攻读学位期间论文工作的知识产权属于哈尔滨工程大学。哈尔滨工程大学有权保留并向国家有关部门或机构送交论文的复印件。本人允许哈尔滨工程大学将论文的部分或全部内容编入有关数据库进行检索，可采用影印、缩印或扫描等复制手段保存和汇编本学位论文，可以公布论文的全部内容。同时本人保证毕业后结合学位论文研究课题再撰写的论文一律注明作者第一署名单位为哈尔滨工程大学。涉密学位论文待解密后适用本声明。

本论文（☑在授予学位后即可 □在授予学位 12 个月后 □解密后）由哈尔滨工程大学送交有关部门进行保存、汇编等。

作者（签字）：辛云龙　　导师（签字）：张志强

日期：2016 年 3 月 21 日　　2016 年 3 月 21 日

摘 要

近年来，随着云服务的兴起，许多拥有剩余计算资源、存储资源的公司利用其多余的资源为广大用户提供云服务，例如阿里云、Amazon Relational Database Service(Amzaon RDS)等。一些拥有数据的公司或者个人，介于其计算资源的限制或者从公司整体利益考虑，会选择将其数据托付给提供云服务的公司进行处理。为了保证这些拥有数据的用户的利益，在将数据上传到云服务提供商之前，需要将数据加密，这样可以防止云服务提供商损害其利益。但是云服务提供商在加密的数据上执行查询等操作将会变得困难，所以能够提供一种高效且安全的方法来解决这一问题十分必要。本文主要研究的是云服务提供商如何在加密的数据上完成用户的 k 近邻查询请求，并且保证在这个过程中数据不被云服务提供商窃取。

目前很多研究者针对这一问题做了研究，具体的研究方法可以分为两类。第一类是通过将数据构建成某种特殊的数据结构，然后采取适当的查找策略，完成查找操作。第二类是通过研究加密算法使得密文数据之间的距离与明文数据间距离存在特定关系，这样只需要通过密文的计算便可完成查找操作。在这些方法中主要存在两个问题。第一个问题是安全问题，在计算过程中攻击者能够通过一些特殊的攻击手段获取到明文内容；第二个是效率问题，一些工作主要将研究重点放到了数据安全方面，忽略了查询效率问题。

本文针对以上问题主要从两个方面进行考虑：首先是查询效率，解决查询效率问题对于提高用户体验、节省计算资源都十分必要，所以本文采用了一种基于层次聚类的 k 近邻查询算法来实验 k 近邻的查找。该算法利用剪枝规则能够排除大部分非近邻数据，提高了查询效率。其次是数据安全性的考虑，加密数据中 k 近邻查询主要涉及距离计算，本文利用同态加密算法的同态性来安全地计算两条数据间距离，这种计算距离的方式可以保证数据的安全性。最后将密文计算距离的方法与基于层次聚类的算法结合来实现密文中 k 近邻的查询操作。经实验表明，本文提出的算法具有较高的查询效率。

关键词：层次聚类；k 近邻；同态加密；密文数据

Abstract

Recent years, with the rise of the cloud service, the company that has extra computing resources and storage resources provide the cloud service for the vast number of users, such as Ali cloud, Amazon Relational Database Service(Amzaon RDS). Some companies or person that cannot manage their data will send the data to service provider that help them tackle users' request, because they have limit computing resources and want to maximize the benefit of the company. In order to ensure the benefits of these people that have data, the data will be encrypted before uploading to the service provider; this can prevent the service providers to harm these people's interests. However, tackling query request over encrypted data will become difficult, so it is very necessary to provide an efficient and safe method to solve this problem. In this paper, we will study how to help the service provider to tackle the k nearest neighbor query request over encrypted data and prevent the data from being stolen by service provider.

At present, many researchers have done research on this problem. The methods to solve this problem can be divided into two categories. The first category is to construct a particular data structure and then to choose a specific search strategy to get the knn results. The second category is to preserve the relation between ciphertext distance and plaintext distance through proposing new encryption algorithm, thus we can get the results by computing the ciphertext distance. There are two main problems in these methods. The first issue is the security of the data cannot be ensured, some attackers can get the plaintext by choosing specific attack method. The second issue is the query efficiency is low. Some work only focused on how to safely compute the distance between two encrypted data and ignored the query efficiency.

In this paper, we solve this question from two aspects. Firstly, we focus on the query efficiency. It is necessary to improve the user experience and save computing resources through reducing query time. So we propose a k nearest neighbor query algorithm based on hierarchical clustering. The algorithm uses the pruning rules to exclude most of the non-nearest neighbor data and improve query efficiency. Secondly, we focus on data's security. The main problem over encrypted data is how to compute the distance between two ciphertext data. In this paper, we use the properties of homomorphic encryption algorithm to compute

the distance between two encrypted data, this method can ensure the security of data. The combination of these two algorithms constitutes a k nearest neighbor query algorithm over encrypted data based on hierarchical clustering. The experiments show that the algorithm proposed in this paper has high query efficiency.

Keywords: hierarchical Clustering; k nearest neighbor; homomorphic encryption; encrypted data

目 录

第 1 章 绪 论 1
1.1 引言 1
1.2 研究的背景及意义 3
1.3 论文主要研究内容 4
1.4 论文结构安排 5
第 2 章 明文和密文查询算法的相关技术及研究 7
2.1 明文的 k 近邻查询 7
2.1.1 精确 k 近邻查找算法 7
2.1.2 近似 k 近邻查找算法 9
2.2 密文数据查找算法 13
2.3 本章小结 14
第 3 章 基于层次聚类密文 k 近邻算法 17
3.1 概述 17
3.2 基于层次聚类 k 近邻查询方法 18
3.3 密文 k 近邻查询算法 26
3.3.1 密文 k 近邻查询算法流程 26
3.3.2 安全与效率分析 30
3.4 本章小结 32
第 4 章 实验结果及评估 33
4.1 实验环境及实验数据 33
4.2 数据维度对算法性能的影响 33
4.3 基于层次聚类算法在 Color 与 Mnist 数据集上效果评估 37
4.4 本章小结 40
结 论 41
参考文献 43
攻读硕士学位期间发表的论文和取得的科研成果 49
致 谢 51

第 1 章 绪 论

1.1 引言

随着云服务的兴起，越来越多拥有强大计算资源、存储资源的公司用其剩余资源为广大用户提供云服务，例如阿里云、Amazon Relational Database Service(Amzaon RDS)等。一些拥有数据的公司或者个人由于本身资源的有限，就会将其数据外包给这些提供云服务的公司来完成数据的存储、管理以及查询的处理。很明显这种方式增加了资源的利用率，为拥有数据的公司以及他们的客户带来极大的便利。然而在数据外包的同时，数据拥有者担心其外包的的数据被云计算提供商窃取或者其用户的隐私数据遭到泄露，所以在外包时需要对数据进行加密处理。但是数据经过加密处理后，在不将数据泄露给云服务提供商的前提下，数据的计算平均值、查找 k 近邻等操作就会变得困难，所以提出高效且安全的查询措施将十分必要。

可以采取两种技术对数据进行加密：一种是对称加密技术，另一种是非对称加密[1]技术。对称加密属于单密钥加密算法，即加密与解密使用同一把密钥或者可以根据加密密钥推出解密密钥。对称加密使用的密钥需要完全保密，所以对称加密的安全性不仅取决于使用的加密算法，还和密钥的管理有关。对称加密主要有 DES、3DES、IDEA 和 AES 等。对称加密的优点主要包括加密速度和解密速度很快，密钥通常来说相对较短，明文的长度和密文长度在通常情况下相同，并且对于硬件实现也非常容易，使用硬件进行加密和解密速度更快。但是对称加密技术的缺点也非常明显，由于其在加密过程和解密过程中使用的密钥是相同的，所以对传输密钥的安全性要求非常高，由于对称加密这一特点造成在发送方传输密钥过程中传输的效率特别低，需要有一个安全通道才可以将密钥进行安全的传输。另外，当有大量密钥的时候，密钥的管理也是非常大的问题，假如有 n 个人用对称加密算法进行通信，那么每个人都需要保存 n-1 个人的密钥，当 n 较大时，密钥的生成、传输、备份、更新等将会变得非常复杂。所以对称加密方式不适合本文要解决的问题，如果要把密钥和加密数据传输到同一个云端，数据的安全性就不能得到保证。

在非对称加密技术中包含两个密钥，一个是可以向大家公开的密钥，即公钥，另外一个是需要保密的密钥，即私钥，只有需要解密的人知道。非对称加密中的私钥不能通过公钥推出，或者不能在很短的时间内通过公钥计算出来，使得攻击者的攻击成本要大

于其获取到的数据的成本。比较典型的非对称加密算法主要有 RSA，ECC 等。非对称加密就如同我们发电子邮件一样，我们的邮件账号大家都知道，大家可以通过我们的账号向自己发送邮件，但是当我们接收到邮件的时候只有我们输入正确的密码才可以看到邮件的内容。这里邮件账号就如同公钥，密码如同私钥。非对称加密的提出可以有效解决对称加密的缺点，对于对称加密中密钥的管理问题，在非对称加密中只需要分发公钥就可以。传统的加密算法为了对信息保密才进行加密，只能实现信息的保密性。而非对称加密不仅具有保密信息的能力，还能够用来做数字签名。数字签名和传统的加密解密不一样，传统的用法是任何人用公钥进行加密，接收人用私钥解密查看内容。而在数字签名技术中签名的人用其私钥对数据签名，然后其他人用其公钥来验证其签名是否有效。因此，非对称加密不仅可以加密，还拥有认证的功能。当然非对称加密也有其自身的优缺点，其优点主要包括密钥的分发比较容易，密钥的管理简单，假如有 n 个人互相通信，n 个人只要产生 2n 个密钥就可以完成任务，对于后来加入的人，密钥的数量是线性增长的。而在对称加密中需要产生C_n^2个密钥，随着加入的人的数量增多密钥的数量是以指数级增长的。非对称加密的缺点主要有密文的长度一般情况下要长于明文，在同一级别的安全强度下，非对称加密的密钥位数要长一些，且加密解密时间相对较慢。

1978 年，Rivest 等人[2]提出了同态加密，同态加密是指在密文数据上经过一系列计算，将计算得到结果进行解密正好与明文上计算得到的结果相等。例如可以这样描述非对称加密算法 RSA 的乘法同态性：对于两条明文 m_1,m_2，它们相应的密文为 $E(m_1)$,$E(m_2)$，如果 $E(m_1)\cdot E(m_2)=E(m_1\cdot m_2)$,那么称其具有乘法同态性。从这个等式可以看出，将 $E(m_1\cdot m_2)$ 进行解密正好等于明文的 $m_1\cdot m_2$，而 $E(m_1)\cdot E(m_2)$可以看做与明文的 $m_1\cdot m_2$ 对应。如果把同态加密算法应用到云计算中，可以让云服务提供商直接在密文上进行计算而不泄露原始数据，但是由于某些加密算法只具有加法同态性或者乘法同态性，在进行密文计算的时候需要采取一定的策略才能保证即能获得准确的结果又可以避免数据泄露。Yousef Elmehdwi 等人[3]提出了 SkNN(Secure k-Nearest Neighbor Algorithm)算法，其算法的主要思想是利用 Paillier Cryptosystem 算法[4]的加法同态性和乘法同态性来计算两条数据的距离。Yousef Elmehdwi 等人发现对于计算 a,b 两个数的乘积，可以通过$a*b=(a+r_a)*(b+r_b)-a*r_b-b*r_a-r_a*r_b$得到。这里$r_a$和$r_b$分别代表两个随机数，通过这样拆解 SP 在计算$a*b$的时候 a 和 b 的信息都不会泄露。满足加法同态和乘法同态性的加密算法有很多种，2009 年 Craig Gentry[5]提出了一种全同态加密方案，但是这种方案非常耗时，Yao Bin 等人指出对于 Craig Gentry 提出的全同态加密只适合理论研究，不能应用于大数据领域。在数据量非常多的情况下，同态加密算法的选择应该以计算过程简便、加密解

密次数少为准则，否则会带来非常大的时间开销。

1.2 研究的背景及意义

由于云计算的灵活性和可扩展性，使其越来越受到关注。云计算模型中涉及到的参与者主要有三个，数据拥有者、用户、云服务提供商(Service Provider 简称 SP)。

数据拥有者拥有数据，但是缺乏对数据分析的能力。数据拥有者可以是企业或者政府部门，例如交通规划部门、物流企业等，也可以是个人，例如个人开发一个 App，但是由于其没有独立的服务器或者当用户量激增的时候由于资金的限制不能购买更多的硬件设备，这时 App 开发者可以将其数据移植到云上，云服务可以为其提供更好的扩展性。但是带来便利的同时，也会带来安全性的问题。我们假设云服务提供商不会主动向外泄露数据，但是对数据的内容好奇，如果数据拥有者的数据含有大量用户的隐私信息或者具有重要的价值，直接放到云端无疑会为用户和数据拥有者带来损失，所以在放到云端托管之前需要对数据进行加密处理，以防攻击者和 SP 获得明文数据。

用户主要是使用数据的个人、企业等 。例如交通规划部门想要通过 SP 的分析获取各个街道的车流量信息，个人可能需要通过 SP 查询其附近的餐厅信息，或者在交友网站中查询和其有相似背景的异性信息等。用户的查询信息也需要进行加密处理，个人的地理位置信息是非常私密的信息，如果攻击者发现某个用户经常在同一地点进行查询，并且了解其查询内容，那么不法分子可以通过分析用户的行为获取更多用户的信息，为其不法活动提供更多的机会。

SP 主要是拥有强大的计算资源和存储资源的公司，例如阿里巴巴、亚马逊、微软等。在阿里巴巴内部有非常多的服务器，在满足公司日常公司需求的情况下，仍然有很多剩余资源，如果把这些资源用来做云计算，不仅解决了资源浪费问题，还能为公司带来可观的收益。在云计算模型中我们假设 SP 是半诚实可信的，即其能够根据算法流程进行计算，但是对数据比较好奇，可能会保留一部分中间计算结果。

数据拥有者在外包数据时必须采取有效的手段或者方法使服务提供商不能获悉数据的具体内容，也不能依据用户的查询来推测出用户的身份。解决这些问题最常用的方法就是给数据加密。数据拥有者在外包数据前先将数据加密，用户提交请求时也同样需要对查询请求加密。服务提供商在处理数据的时候接触到的数据全是密文，这样就使得用户的隐私和数据拥有者的利益得到了保护。数据加密的处理带来安全性的同时也带来了处理上的困难，服务提供商必须根据加密的数据和加密的查询请求为用户返回精确的

查询结果，但是在密文上处理数据不同于明文，例如在密文上计算两条数据的距离，可能需要某种方式的解密再加密操作，或者直接在密文上计算，第一种方式安全性差一些，第二种方式效率较低。所以提出一项合理的方案解决该问题非常必要。

所以这里存在的主要问题是数据以加密的方式存储在云端时，服务提供商如何完成用户的查询请求。现在有很多技术解决加密数据上的范围查询(Range query)、聚合查询(Aggregate query)等其他查询请求，但是对于某些高级查询请求，例如 k 近邻查询请求，则没有适当的解决方案，一则是不适用，二则效率较低。所以本文基于此主要研究了加密数据上的 k 近邻查询请求，第四章中实验结果表明，本文方法有较高的查询效率。

1.3 论文主要研究内容

本文主要研究云端加密数据的 k 近邻查询问题。在云端加密数据的 k 近邻查询模型中主要涉及三个参与者。数据拥有者将其数据发送到 SP 端，SP 主要负责加密数据的计算、查询，用户向 SP 提交查询请求 Query, SP 根据 Query 计算出离其最近的 k 条数据，在此计算过程中需要保证如下几点：

第一：用户的 Query 需要加密，在计算过程中不能泄露给 SP。

第二：用户最终必须获得 Query 的精确 k 近邻。

第三：在用户端需要避免大量计算，过于繁重的计算会导致用户的体验下降。

第四：数据拥有者的数据以及在计算时过程中的中间结果不能泄露给 SP.

第五：为了防止数据拥有者和 SP 或者其他攻击者根据用户的查询 Query 推理用户的行为和隐私信息，计算过程中需要保证返回给用户的 k 条数据的 ID 不能泄露给数据拥有者和 SP.

本文针对以上几点要求做了如下研究：

首先，由于考虑到查询效率的问题，我们对如何提高 k 近邻查询效率做了研究。k 近邻查询算法主要分为两类：近似 k 近邻和准确 k 近邻。由于上面第二点的要求，本文只考虑准确 k 近邻查询算法，但是在要求放宽的情况下，近似 k 近邻算法也是很好的选择，可以非常显著地提高查询效率。本文提出了一种基于层次结构查找 k 近邻的方法。该算法利用 k-means 算法将所有数据分成 m 份，即 k-means 算法中 k 值设为 m，这样聚类后的数据就会产生 m 份，对于 m 份中的每一份数据再依次分裂成 m 份，直到叶子节点不再满足分裂要求则停止分裂。算法中规定每个叶子节点内包含数据数量的最大值和最小值。当节点中数据数量大于最大值时，则节点继续分裂。若分裂后获得的节点内的

数据数量小于规定的最小值，则将该节点与其同时分裂产生的其他节点进行合并。若节点内的数据数量小于最大值，大于最小值，则该节点为叶子节点。基于层次结构的查询可以采用剪枝规则进行剪枝，剪枝的过程中可以有效的剪掉大部分不满足结果的数据，这样可以显著的提高查找的效率。

其次是针对于安全性的考虑。对于加密数据的计算可以采用同态加密算法。由于同态加密算法具有加法同态性和乘法同态性，所以可以安全地计算两条数据的距离而不泄露数据自身的内容。本文主要采用 Paillier 加密算法在云端计算两条数据的欧式距离，为了保证计算产生的中间结果和返回给用户数据的 ID 不泄露给 SP, 本文需要两个 SP(SP_A,SP_B)相互合作计算 k 近邻。数据拥有者首先将数据按照层次聚类的方式聚类，在聚类后利用加密算法进行加密，然后将加密的数据发送到 SP_A，将解密的密钥放到 SP_B，这里假设 SP_A 与 SP_B 不会相互合作解密数据。在数据拥有者发送过数据后，数据拥有者的任务至此结束，在以后用户查询的过程中不再进行参与。用户提交查询请求时，将 Query 加密发送到 SP_A，SP_A 与 SP_B 利用同态加密算法的加法同态性和乘法同态性安全地计算 Query 与加密数据的距离。在这个过程中可以保证数据不会泄露给任意一个 SP，从而保证数据的安全性。用户在收到数据后只需非常少的计算即可还原数据的真实值，从而避免了用户端繁重的计算，并且又保证了结果的准确性。

1.4 论文结构安排

本文主要研究的是在加密数据上查找 k 近邻的问题。根据问题的性质和对各个参与者的相关的要求，本文从两方面考虑来解决密文 k 近邻问题。首先考虑查找效率问题，基于此本文提出了基于层次聚类的 k 近邻查询算法，该算法为一种查找策略，不仅适用于明文 k 近邻查找，同样适用于密文 k 近邻查找。该算法与密文中安全计算距离的算法相结合，即可完成查找任务。然后是从数据安全方面考虑，基于此本文结合基于层次聚类的 k 近邻查询算法设计了一种即可以安全计算两条数据距离，又能够安全返回结果的算法，本文称该算法为基于层次聚类的密文 k 近邻查询算法。第四章的实验结果证明了该方法的可行性和实用性。本文的结构安排如下：

第 1 章：绪论。第一章首先探讨了云端加密数据查找算法的意义，然后分析了几种加密方案的优缺点以及对问题的适用性，最后简要的介绍下本文的主要研究内容。

第 2 章：明文和密文查询算法的相关技术及研究。第二章主要介绍了下以前研究人员的相关研究成果。由于云端加密数据 k 近邻查询算法即涉及到明文 k 近邻查询算法又

涉及到加密的部分，所以第二章首先从准确和近似两方面介绍了明文 k 近邻查询算法的研究成果，然后又介绍了最近密文查询算法的研究成果。

第 3 章：基于层次聚类的密文 k 近邻算法。第三章介绍了本文提出的解决密文 k 近邻查询的算法。本章首先介绍了基于层次聚类的 k 近邻查询算法，然后介绍了密文 k 近邻算法的大致流程，最后对算法的效率和安全性做了分析。

第 4 章：实验结果及评估。主要介绍了实验所采用的数据集、实验环境以及在实验中的参数设置情况。分别对本文的方法和其他人的方法做了相应的评估。

最后总结了本文方法的贡献、算法的不足和未来的研究方向。

第 2 章 明文和密文查询算法的相关技术及研究

对于加密数据查询处理算法的研究可以分为明文查询算法和密文查询算法两大类。例如在密文上的 k 近邻查询算法，遇到的主要问题是数据在加密的情况下如何高效查找 k 近邻和计算两条密文的距离并且不将明文泄露给 SP。明文查询算法的研究主要是解决效率问题，密文查询算法的研究主要是为了解决安全问题。这两点可以分别研究，最终予以适当的结合。

Hu Haibo 等人[6]利用在明文上采用的 R 树来处理密文上面的查询操作。Yao Bin 等人[7]通过建立 Voronoi 图来处理在二维数据上的最近邻查询操作。这两种方案都可以显著提高在密文上查询处理数据的效率，能够很好的适应大数据上查询处理的要求。因此可以将明文上采用的特殊数据结构应用到密文，来减少密文数据查询操作的时间。

2.1 明文的 k 近邻查询

明文上查询算法的研究可以分为两类，一类是准确查找，一类是近似查找。准确查找的算法时间复杂度相对较高，但返回的数据质量相对较好。近似查找返回的结果与采用的近似查找算法和数据的分布都有关系，如果数据均匀分布，近似算法能够返回相对满意的结果，如果数据分布不均匀，对于某些特殊查询会导致不能返回结果或者返回的结果不满足用户的要求。近似算法非常好的特点是查找时间复杂度极低，能够应用到大数据上。

2.1.1 精确 k 近邻查找算法

精确 k 近邻查找算法有很多，这些算法在数据库，文本检索，排序等应用中具有非常重要的作用。有一些精确 k 近邻查找算法需要使用一些特殊的数据结构[8,9]，还有一些用到了空间索引，例如 kd 树，R 树等[10-12]。

k 近邻查找策略可以分为最佳优先 k 近邻查找[13]和深度优先 k 近邻查找。最佳优先 k 近邻查找就是用最佳优先的策略来遍历搜索层次，类似于 A^*算法。在算法中的每一步，算法首先访问的是离查询对象最近的节点（例如 R 树中的节点）。实现这种策略可以使用优先队列，优先队列中存储的是访问过的节点，并且按着离查询对象距离的远近来排序，当要进行下一步搜索的时候，首先取出优先队列中第一个元素，取出以后判断是不是叶子节点，如果是叶子节点则进行标记放入候选 k 近邻中，如果不是叶子节点，则计

算查询对象与该元素孩子的距离，然后按照优先队列的要求将该节点的孩子插入到优先队列中并将该元素出队列。依次按照上述步骤进行搜索直到满足要求为止。

假设在候选 k 近邻中已经有 k 条数据，并且可以确定在优先队列中的元素以及元素的孩子节点都没有满足 k 近邻要求的情况下，可以将优先队列清空，这一步骤相当于剪枝。这样避免了将所有元素与查询对象进行计算，提高了效率。对于是否能够剪枝取决于我们用的数据结构和查询对象到节点的距离，第三章会讨论具体的剪枝规则。

深度优先查找策略可以应用到任何基于层次聚类的索引上[14]。基于层次聚类的基本思想是首先将数据先划分为丛，然后这些丛再进行聚类变成另外一个丛，直到所有数据聚合成一棵完整的树。在查找过程中深度优先算法会维护一个候选 k 近邻的大顶堆，算法首先从根节点进行查找，然后每访问一个节点进行判断是否是叶子节点，如果是则计算查询对象和叶子节点中心的距离，如果计算后的距离小于堆中最大的距离则更新堆，如果节点非叶子节点则重复这一算法过程。在深度优先查找过程中会用到下文介绍的剪枝规则，剪枝规则可以排除掉许多非 k 近邻点，节省不必要的查找时间。

何洪辉等人[15]提出了一种高效的并行 KNN-join 处理方法，该方法采用了双层的数据结构，第一层采用 Gorder[16]将数据分成多个 cell, 第二层采用 B^+树对距离建立索引。在建立好两层索引以后，算法采用并行的方式搜索 cell,并且通过某些剪枝规则去掉不必搜索的 cell 来提高查找效率。Zhang Chi 等人[17]利用 mapreduce 来解决大数据的 KNN-join 问题，mapreduce 是一个简单却非常强大的计算框架，它能够很好地应对并行和分布式计算。Zhang Chi 等人正是利用 mapreduce 的这一特点设计了一种并行的 KNN-join 算法，实验结果表明该方法较基本算法有 1-2 个数量级的提升。

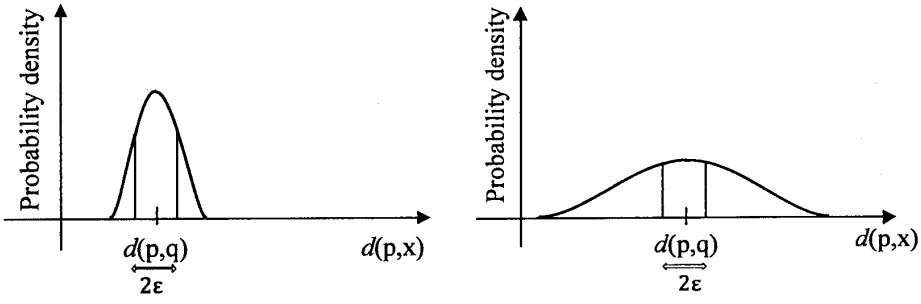
随着数据维度的增加，采用基于距离的索引和基于维度的索引算法性能都会越来越差[18]。基于 Graphics Processing Unit (GPU)来计算 k 近邻的方法[19-21]得到了研究者们的关注。GPU 即图形处理器，GPU 是显卡的核心，在功能上与 CPU 类似，但是 GPU 是专为执行复杂的数学和几何计算而设计的，在某些 GPU 中集成的晶体管数量超过了 CPU，这样对于提升计算能力很有帮助。Vincent Garcia 等人[19]利用 CUDA 和 CUBLAS API 实现了穷举 k 近邻查询算法（计算查询对象和所有数据的距离）。算法的主要思想是利用 GPU 的并行计算能力来增加计算的速度。对于有 n 个查询对象 m 条数据并且数据维度为 d 的查询系统和一个拥有两个核心的 GPU，算法首先利用一个核计算一个 $n \times m$的距离矩阵，由于矩阵中每个元素都是独立的，所以可以并行计算。然后用第二个核心来对矩阵进行排序，由于 n 个查询对象也是彼此独立的，所以 n 个处理进程也可以并行执行。算法通过 GPU 加速后，最终的结果要比经过优化的 ANN C++库[20]快 189

倍。

穷举法主要的优势在于查找数据时是顺序扫描数据，而基于索引的办法往往会随机访问数据，在不能有效剪枝的情况下，基于索引方法的 IO 代价要比穷举法高，所以会慢于穷举法。不能进行有效剪枝的主要原因在于高维情况下遇到了维灾难。在相似性搜索中维灾难是指随着数据空间维度的增加，需要进行检查的数据的数目是成指数级增长的。Beyer 等人[22]指出，随着数据维度 d 趋向于无穷时，数据中任意两条数据 q1 和 q2 间的距离期望和方差比值趋向于 0，见公式（2-1）

$$lim_{d\to\infty}\frac{Variance[d(q1,q2)]}{Expected[d(q1,q2)]}=0 \qquad (2\text{-}1)$$

即任意两条数据的距离值的分布更加集中。图 2.1 描述了数据之间距离值分布的两种情况。从图 2.1(a)中可以看出数据之间的距离分布更集中，而图 2.1(b)的数据距离分布比较均匀。数据间距离分布的越均匀越能有效地利用剪枝规则，大部分剪枝规则是基于距离设计的，在距离分布较集中时，剪枝规则基本不会起作用，最后效率要低于穷举法。所以在高维情况下基于距离的索引不能很好的排除掉非 k 近邻点，这时的搜索是非常耗时的。准确查找 k 近邻算法的主要限制是要求返回的 k 近邻是精确的，精确的结果对于返回的数据质量虽然有很好的保证，但是花费的时间代价也是非常明显的，尤其在高维数据情况下。这种情况下一些研究者提出了近似 k 近邻算法，近似 k 近邻算法主要是在牺牲一定数据质量的情况下显著减少算法在查找过程中花费的时间。



(a) 距离值方差较小的密度函数　　(b) 距离值分布更加均匀

图 2.1 距离 d(p,x)概率密度函数，其中两条竖线中间对应|d(p,q)-d(p,x)|≤2ε;

2.1.2 近似 k 近邻查找算法

Vrbsky 和 Liu[23]设计的APPROXIMATE系统是最早提出返回近似查询结果的工作之一。精确查找 k 近邻不仅耗时，而且对于某些应用也是不必要的，例如决策支持和数

据分析应用常常会处理大量数据，如果要求返回精确的结果将会造成显著的系统延时，在这样的情况下一些用户希望牺牲返回结果的精度来换取效率，这时近似 k 近邻查找算法的效率优势就得到了关注。对于精确 k 近邻算法耗时的主要原因之一是维灾难的出现，Beyer 等人[22]的研究成果表明在数据维度达到 10-15 维的时候维灾难就已经开始显现出来了。不过也有一些研究成果[24]表明出现维灾难是基于非常严格假设的，在现实情况中这些假设不是常常都能成立的。在维灾难出现的时候，基于距离索引的精确 k 近邻查找算法就会慢于穷举法，但是穷举法也不能满足查找算法的时间要求，随着数据的增多，查找时间是成线性增长的。对于一个理想的算法应该满足在数据增多的时候，算法的时间复杂度是次线性增长的。Jagadish 等人[25]提出了 iDistance 算法，Fagin 等人[26]提出了 MedRank 算法，这两个算法虽然能够满足时间要求，但是效果不是很稳定，对于某些特定数据集和查询请求算法表现良好，对于其他数据集可能会有非常大的时间开销。

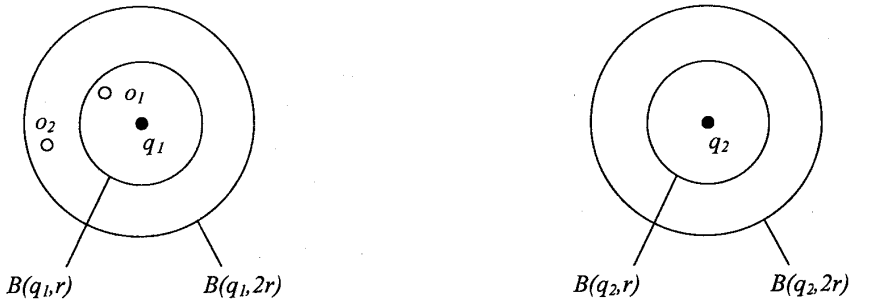
有非常多的近似 k 近邻查询算法[27-30]被提出来，这些算法分别从不同的角度进行设计，但是在已知的算法中，局部敏感哈希(Locality sensitive hashing (LSH))是唯一一个能够满足查询时间要求的算法。哈希函数的作用是尽量将不同的数据映射到不同的桶内，在查找时只需要 O(1)的时间复杂度就可以获得我们想要的数据。如果将哈希的思想应用到 k 近邻查找中，那么时间复杂度将会变为常数，极大提高查找的效率。但是某些哈希函数在映射数据时没有考虑两条数据在空间的关系，也就是说两条数据所在桶的 ID 号没有太多在数据空间层面的联系。这样就会造成我们不能够查找到 k 近邻。所以局部敏感哈希算法选取了特定的哈希函数使空间中距离相近的数据，尽可能映射到同一个桶内。LSH 对于 Jaccard 距离、余弦距离、欧式距离、海明距离以及其他度量函数，都需要设计不同的哈希函数。一个*局部敏感*的哈希函数要满足以下两个条件：(1) 如果$d(x,y)\le d_1$，那么 h(x)=h(y)的概率至少是 p_1；(2) 如果$d(x,y)\ge d_2$，那么 h(x)=h(y)的概率最大是 p_2。这里 x,y 代表空间中两条数据，d_1,d_2 代表某个度量函数 d 下的距离值，h(x),h(y)代表数据的哈希值。对于较大的 p_1 和较小的 p_2 可以返回高质量的结果。在查找过程中首先将 Query 进行哈希，然后根据哈希所得的 ID 号取出相应桶内的数据，再与 Query 计算距离，按着排序的结果返回近似的 k 近邻。整个过程时间花费主要集中在哈希函数计算和桶内数据与 Query 距离计算上，由于桶内数据与所有数据相比只占小部分，所以可以明显提高查找效率。LSH 支持 c-approximate 近邻查找，假设数据 o 是 Query 的近似最近邻，o^*是 Query 的准确最近邻，那么 c-approximate 表示$\|o,q\| \le c\|o^*,q\|$,这里$c \ge 1$,代表近似比，q 代表 Query，这个公式说明 q 到近似最近邻的距离最多是 q 到准确最近邻距离的 c 倍。

最早提出的LSH算法主要分为两种，一种是rigorous-LSH[29]，一种是Adhoc-LSH[30]。这两种算法最先并不是为了解决c-approximate问题，而是为了解决所谓的c-approximate ball cover(BC)问题。假设B(q,r)表示以q为中心，r为半径的超球体。c-approximate ball cover(BC)问题是指：对于一个数据集D和一个给定的查询请求q,算法返回如下结果：

（1）如果B(q,r)包含D中任意一条数据，那么返回B(q,cr)中任意一条数据。

（2）如果B(q,cr)不包含D中的数据，那么不返回任何结果。

如图2.2：



(a) 在B(q,cr)中包含数据　　(b) 在B(q,cr)中不包含任何数据

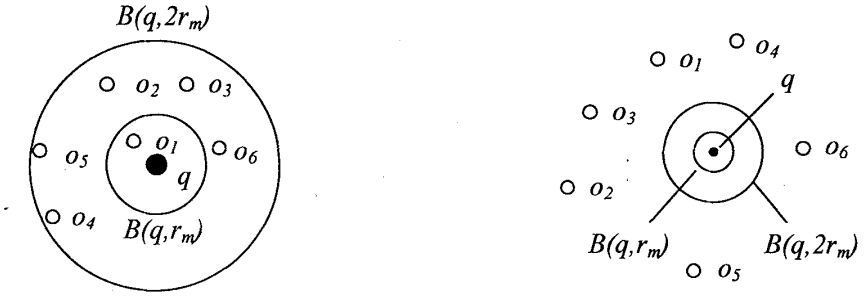
图 2.2 ball cover 查询返回结果的两种情况

在图2.2(a)中B(q1,r)包含o1，所以最终会返回o1或者o2中任意一条数据。在图2.2(b)中B(q2,2r)中没有任何数据，所以不返回任何结果。c-approximate ball cover(BC)问题很容易转化成c-approximate问题，如果B(q,r)是空的，但是B(q,cr)不是空的，那么在B(q,cr)中的数据就是q的c-approximate近邻。基于这样的想法，Indyk等人设计了rigorous-LSH算法，该算法能够针对不同的半径r返回相应的结果，例如r=1,c,c2,c3… 正是因为rigorous-LSH能够调整半径返回高质量的结果，所以导致其时间和空间开销很大。针对rigorous-LSH的弊端，Gionis等人设计了Adhoc-LSH算法。 Adhoc-LSH算法并没有在执行过程中调整半径，而是采用了启发式的方法在开始时固定了搜索的半径，例如对于2- approximate BC问题，算法只需要考虑B(q,r)和B(q,2r)，而rigorous-LSH还需要考虑r=r1, r2…等其他情况。Adhoc-LSH虽然节省了空间和时间，但是返回数据的质量较差。图2.3描绘了Adhoc-LSH算法的缺点。

在图2.3(a)中半径r定义的太大，根据前面c-approximate BC返回结果的说明可知算法会返回B(q,2r)中任意一条数据，但是从图中可以看出只有o_1是离q非常近的。图2.3(b)中半径r定义的太小，导致不能返回任何结果。所以Adhoc-LSH算法对数据的分布比较敏感。

Rigorous-LSH算法能够保证查询结果的质量，但是其时间和空间消耗是不能接受的，

Adhoc-LSH 算法查询效率较高，但是查询结果的质量对数据的分布比较敏感。如果将这两个算法进行结合，将会弥补各自的缺点。基于这个目的，Yufei Tao 等人[31]提出了 locality sensitive B-tree(LSB-tree)，该算法能够在高维数据的查询中在保证查询效率的前提下返回高质量的查询结果。通过将多棵 LSB-tree 合并可以组成 LSB-forest, LSB-forest 所占用的空间与 Adhoc-LSH 同一个数量级，但是其却保留了 rigorous-LSH 高质量查询结果的特性。在数据均匀分布的情况下，基于 LSH 的方法可以将数据均匀映射到各个桶内，但是在其他情况下会出现各个桶内数据差异变大的情况，导致 LSH 算法表现不稳定，图 2.4(a)描绘了 LSH 算法的缺点。基于这个原因，Jinyang Gao 等人[32]提出了 Data Sensitive Hashing(DSH)算法，DSH 会针对数据的分布学习不同的哈希函数，使得各个桶内数据分布更加均匀，如图 2.4(b)。DSH 的主要思想类似于机器学习中的 Adboost 算法，都是从已标记（每个 Query 对应着相应的 k 近邻）的数据中通过 boosting 的方式进行学习，DSH 从标注数据中抽样获得权重矩阵，权重矩阵中 w_{ij} 初始值为 1,-1,0,分别代表 o_j 是 q_i 的 k 近邻、非 *ck* 近邻和其他情况。在学习获得一个哈希函数后，更新权重矩阵中的值，在进行多次迭代以后会获得哈希函数族，用这些哈希函数映射 Query，然后取出相应的数据与 Query 计算距离，返回 k 近邻。DSH 方法会受到 Query 的分布影响，如果多数 Query 并没有在训练数据中出现，最终哈希的结果将会不理想。



(a) 半径 r 定义的太大　　(b) 半径 r 定义的太小

图 2.3 Adhoc-LSH 算法缺点

(a) LSH 算法各桶内数据数量情况　　(b) DSH 算法各桶内数据数量情况

图 2.4 LSH 算法与 DSH 算法映射数据情况

2.2 密文数据查找算法

加密数据查询的主要难点在于查询过程中数据拥有者的数据保护，用户 Query 的保护以及防止数据被攻击者破解。数据拥有者的数据关系到其自身利益和其用户的利益，然而在密文下的 k 近邻计算又有诸多的困难，例如对于欧式距离，在计算的过程中会涉及到数据的平方、距离大小的比较和 Query 与查询结果隐藏等问题。

这些困难都会带来数据泄露的风险，数据的平方和距离大小的比较在处理过程中很可能会涉及到数据的解密与计算，在这个过程中如何处理解密后的数据非常重要，如果直接解密将会泄露数据，如果解密后的数据并非原始数据，那么如何保证结果的正确性需要仔细研究。对于 Query 与查询结果隐藏问题也会涉及到数据库的安全，如果攻击者发现同一 Query 或者相近的 Query 返回的所有结果（密文的结果），将会增加攻击者破解数据库的概率。对于这样的问题可以采用隐藏 Query 和结果的方法，用户提交 Query 后，云服务商根据指定的算法在保证计算结果正确性的前提下向 Query 中加入随机值[3]。由于随机值的加入将会使得同一 Query 或者相近 Query 的发现变得困难，所以在返回结果的过程中也可以采取同样的方法。

除了加入随机值的方法外，还有一种 distance-preserving transformation(DPT)[33]加密算法，DPT 的主要思想是将明文数据加密得到密文，在密文上计算两条数据的距离等于在明文上计算两条相同数据的距离。很明显在计算的过程中没有涉及到数据的解密，但是这种加密方式在实际应用中并不安全。如果一个攻击者能够得到所有的加密数据，并且知道在此数据库中的少部分明文数据，就会很容易破解所有加密数据[34-36]。

Hu Haibo 等人[6]将数据保护的研究工作分为三类：第一类的研究者们关注如何使得用户的敏感信息尽可能少的泄露[37-41]。这一类的研究工作主要应用于空间数据中轨迹信息发布和查找，在发布过程中为了防止攻击者获取到用户的敏感信息（家庭住址，长时间停留位置等），通常采用 k 匿名的办法来模糊用户的准确位置。然而这类工作中常用的相似性度量方式[42]与我们采用的不大一样。轨迹数据采用的相似性度量为编辑距离[43]、最长公共子序列[44]等，而我们关注的是欧式距离、余弦距离等。第二类的工作采用的方法是将明文数据进行加密，在加密数据上进行操作[3,6,7,36,45,46]。我们的工作也采用这种方式。第三类工作将数据进行分割，然后外包给几个独立的服务提供商，服务提供商之间通过设计好的分布式安全协议处理数据[47-49]。由于是多个服务提供商协同工作，所以存储、通信将会花费较多时间，在用户量和数据量较大情况下严重影响效率。

Yao Bin 等人[7]提出的 Secure Nearest Neighbor(SNN)算法和 Haibo Hu 等人[6]提出的

算法都采用了为查询而设计的数据结构和查找策略来实现 k 近邻的查找。SNN 算法整体思想是首先构造一个 Voronoi 图。Voronoi 图中有多个 cell，每个 cell 中都包含一条数据库中的二维数据 p，称 p 是一个 cell 的“拥有者”。一个 cell 中的任何数据与 Voronoi 图中其他 cell 中的拥有者的距离永远大于这些数据与其自身 cell 的拥有者的距离，也就是说如果一个 Query 落入 cell c（其拥有者是 p）中，那么 Query 的最近邻是 p。在 Voronoi 图构造好以后，需要对 Voronoi 图进行分割，分割的主要目的是为了提高查找效率。分割后将数据加密发送到服务提供商，服务提供商会根据设计好的查找策略给用户返回查询结果。SNN 的查找效率非常高，但是其只适合二维数据，并且对于数据的更新也非常麻烦。Hu Haibo 等人将数据构建成一棵 R 树，利用 ASM-PH 加密算法对数据加密，然后将加密后的索引发送到用户端，将解密算法发送到服务器端。用户在提交查询请求时先将 Query 加密，然后在服务器端的帮助下遍历 R 树获取结果。但是 Hu 等人的方法并不安全，Yao 等人证明利用选择明文攻击方法能够获使用户端得到所有的数据。

Wong 等人[36]在数据发送到服务提供商之前将数据用加密算法 E_T 进行加密，用户在查询数据时将 Query 用另一套加密算法 E_Q 进行加密。用户数据和数据拥有者数据经过这两种加密算法加密以后，可以使得$q \cdot p = E_Q(q) \cdot E_T(p)$,但是对于数据库中的两条数据 p_1,p_2 具有$p_1 \cdot p_2 \neq E_T(p_1) \cdot E_T(p_2)$的性质，也就是说数据库中两条数据的点积不能够通过密文获得。用户提交$E_Q(q)$后，服务器端利用$q \cdot p = E_Q(q) \cdot E_T(p)$这个公式计算距离，然后将计算获得的密文结果发送给用户，用户根据数据拥有者分享的密钥进行解密。Wong 等人的方法虽然可以使得密文上的点积等于明文上的点积，但是其在计算过程中会泄露数据[7]。Yousef 等人[3]利用加密算法的加法同态性和乘法同态性来计算两条密文数据的距离。算法将加密后的数据发送到服务提供商 SP_1,将解密的密钥发送给服务提供商 SP_2，然后根据其设计的策略使得两个提供商相互协作计算结果。Yousef 等人的算法虽然可以保证数据安全，但是在整个计算过程中会涉及到非常多的数据加密解密操作，而这两种操作非常浪费时间，导致其效率不高。

2.3 本章小结

本章主要介绍了明文 k 近邻查询算法和密文 k 近邻查询算法研究进展。明文 k 近邻查询主要分为两类，一类是准确 k 近邻查找算法，一类是近似 k 近邻查找算法。准确 k 近邻查找算法包括基于 GPU 的并行查找算法和基于索引的深度优先和最佳优先查找算法。在数据高维情况下基于索引的算法效率会下降，近似 k 近邻算法可以解决数据高维

带来的困难。在近似 k 近邻查询算法中能够满足时间要求的只有 LSH 算法。但是最早的 LSH 算法存在很多缺陷，对数据分布比较敏感，返回结果的质量不稳定。针对这些问题 Tao 等人提出了 LSH-tree 算法，这一算法继承了 LSH 的优点，并且提高了结果的质量。Gao 等人的 DSH 算法主要解决了数据分布不均匀的问题，DSH 通过 Adaboost 方法从数据中学习哈希函数，在获得哈希函数族后可以对数据和 Query 进行映射获得结果。密文数据中的 k 近邻查询主要有 Yao 等人提出的 SNN 算法，Yousef 等人提出的 SkNN 算法等。SNN 算法主要解决的是二维加密数据中查找最近邻问题。SkNN 算法利用同态加密算法的加法同态性和乘法同态性计算两条数据的距离，但是由于其需要加密解密多次，在计算过程中时间消耗非常大。

第 3 章 基于层次聚类密文 k 近邻算法

3.1 概述

随着云服务的广泛应用，越来越多的数据拥有者将数据外包给云服务提供商，但是数据外包的同时也面临数据被泄露的风险，所以如何有效的操作数据受到了研究者们的关注。本章主要讲述本文提出的基于同态加密的 k 近邻算法。本文假设系统中包含三个角色，数据拥有者(Data Owner)、云服务提供商(Service Provider)，用户(User)。数据拥有者代表拥有数据，但是由于其计算资源、存储资源或者人力资源有限不能独自处理数据的组织或者个人，例如银行、相关企业、政府部门、研究机构，App 的开发者等。SP 主要指拥有大量服务器的公司，例如阿里巴巴、亚马逊等，这些公司可以利用其剩余的资源为有需要的单位或者个人提供服务，并且可以按照一定的规则收取费用。用户主要指利用数据的个人或者单位，例如个人想要查找数据库中与某幅图像最相近的前 k 幅图像，或者交友网站中用户查询与其背景、生活爱好等最相近的异性。

本文提出的方法中包含两个云服务提供商 SP_A 和 SP_B，并且假设云服务 SP 是半诚实的，即 SP 之间会根据我们制定的协议计算数据，不会为了获取明文数据而相互协商解密数据，但是其对数据比较好奇，可能会保留一些计算的中间结果。在我们提出的算法中即使 SP 保留了这些中间结果，也可以保证数据拥有者数据的安全性。

在整个流程中数据拥有者首先将数据加密上传到 SP_A，然后将解密的密钥上传到 SP_B。查询过程中用户首先提交加密的 Query 给 SP_A，然后 SP_A 和 SP_B 相互合作计算 k 近邻。最后 SP_B 将结果返回给 User。在整个过程中需要确保数据拥有者的数据不能泄露给 SP，用户的 Query 不能泄露给 SP 和数据拥有者，以及 SP 不能确认针对 Query 具体返回了哪些数据。

本章首先介绍了基于层次聚类的 k 近邻查询算法，k 近邻查询时采用的度量方式为欧式度量。提出基于层次聚类的算法主要是为了解决系统查询效率问题，该方法可以有效地进行剪枝，减少数据间距离的计算量。接下来介绍了完整的密文 k 近邻查询算法，该算法是由基于层次聚类查询算法和密文数据计算距离算法相互结合而成，这样既保证了查询效率又保证了数据的安全性。

3.2 基于层次聚类 k 近邻查询方法

k 近邻查找主要涉及到距离计算，如果利用距离建立索引将会提高查找的效率。

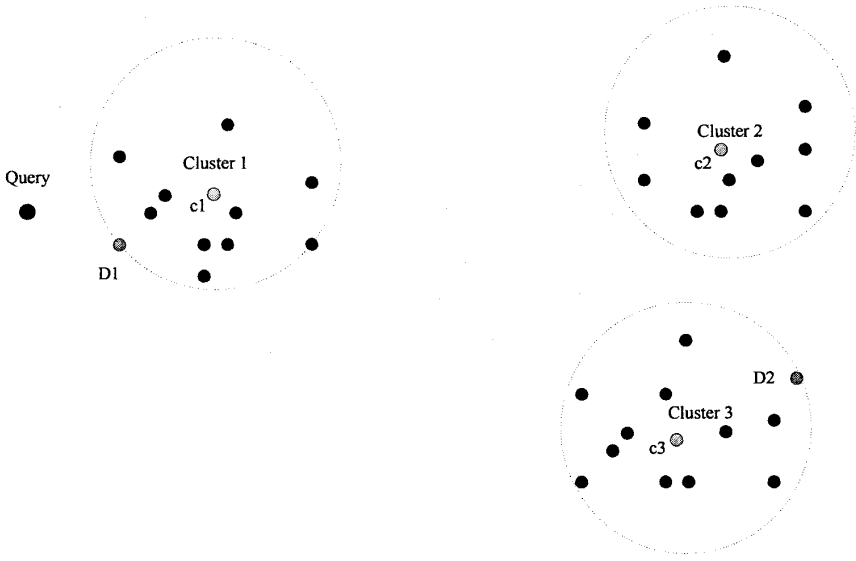


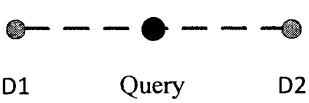
图 3.1 数据和 Query 的分布

图 3.1 中所有点代表二维数据，Cluster1, Cluster2, Cluster3 代表通过聚类算法形成的三个簇，c1、c2、c3 代表簇中心。假设 D1 是 Query 的最近邻，并且知道 Query 和 D1 的距离 dis_{q1}，D1 与 D2 的距离 dis_{12}。那么即使不计算 dis_{q2} 也可根据 dis_{q1} 和 dis_{12} 推断出 D2 绝不可能是 Query 的最近邻。

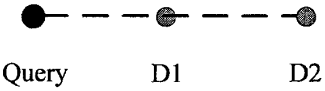
从图 3.1 中的 dis_{q1}、dis_{12} 和 Query 与 D2 的距离 dis_{q2} 可以推导出式（3-1）成立：

$$dis_{12} - dis_{q1} \leq dis_{q2} \tag{3-1}$$

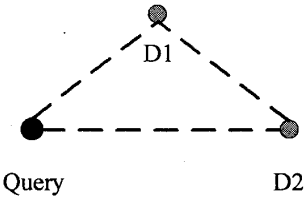
式（3-1）中小于成立的条件是三角形两边之差小于第三边，其中 Query、D1 和 D2 分别为三角形的三个顶点(图 3.2(c))。等号成立的条件是三个顶点在同一条线上且 Query 在中间（图 3.2(a)）。Query 与 D2 的最小距离为$|dis_{12} - dis_{q1}|$（图 3.3(a)），最大距离为 $dis_{12} + dis_{q1}$（图 3.2(b)）。并且在图 3.2 中用 D1 与 D2 的距离 dis_{12} 减去 D1 与 Query 的距离 dis_{q1} 要大于 dis_{q1}，即$|dis_{12} - dis_{q1}| > dis_{q1}$，又因为$|dis_{12} - dis_{q1}|$是 D2 与 Query 的最小距离，所以 D2 不可能是 Query 的最近邻。在整个过程中并没有计算 D2 与 Query 的距离，这种方法可以避免不必要的距离计算。



(a) Query 在 D1、D2 中间，三点共线



(b) D1 在 Query 与 D2 中间，三点共线



(c) 三点构成三角形

图 3.2 Query、D1 和 D2 关系图

上述的方法中需要知道所有数据之间的距离，并且每条数据都需要逐一排除。仔细观察图 3.1 可以发现 Query 与 Cluster 2 和 Cluster 3 的距离都很远，如果把每条数据逐一排除的方法变成每个簇逐一排除，将会进一步减少数据比较的次数。假设图 3.1 中每个簇的半径分别为 r1、r2、r3，c1、c2、c3 为每个簇的中心，半径 ri 表示簇内所有数据到簇中心的最远距离即：

$$ri = max\{ dis \mid dis = \|ci, Dj\|, Dj \in Cluster\ i\} \tag{3-2}$$

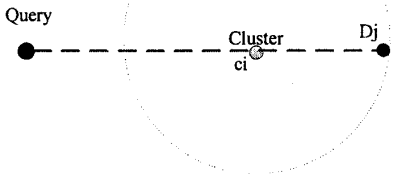
簇中心 ci 表示簇内所有数据的平均，即：

$$ci = \frac{\sum_{j=1}^{n} Dj}{n} \tag{3-3}$$

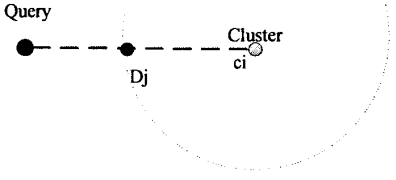
n 代表 Cluster i 内数据的数量。假设图 3.1 中每个 Cluster 中数据的数量大于 k，且 Query 到 c1 的距离为 3，c1 的半径为 2，到 c2 的距离为 10，c2 的半径为 2，到 c3 的距离为 9，c3 的半径为 3。那么可知 Query 到 c1 中所有数据最远的距离为 5，到 c2 所有数据的最近距离为 8，到 c3 所有数据的最近距离为 6。图 3.3 展示了如何计算 Query 到簇内所有数据最远距离和最近距离。Query 到 Cluster 1 的最远距离要小于到 Cluster 2 和 Cluster 3 的最近距离，所以 Query 的 k 近邻一定包含在 Cluster 1 中。Cluster 2 和 Cluster 3 里面的其他数据不需要与 Query 进行计算就可以完全排除。

基于单个簇排除的方法的效率要高于基于单个点的效率，基于簇的方法需要确定每个簇内数据的数量以及簇的个数。每个簇内的数据过多将不能有效的排除，例如所有数据只聚类成两个簇，这样每次即使能够排除也只能排除一个簇，需要与剩下那个簇内所

有数据计算距离。每个簇内数据如果太少，那么需要与簇中心计算距离的次数将会变多，例如每个簇内只有两条数据，那么需要与簇中心计算距离的次数等于N/2，N 表示所有数据的数量。基于簇的方法最好的平均计算距离的次数为$2\sqrt{N}$。



(a) Query 到簇内数据最远距离，$dis_{max} = dis_{qc} + dis_{cDj}$，$Dj \in Cluster$



(b) Query 到簇内数据最近距离，$dis_{min} = dis_{qc} - dis_{cDj}$，$Dj \in Cluster$

图 3.3 Query 与簇内数据距离关系

基于簇的方法在遇到图 3.4 的情况效率会变低，假设 Cluster i 不能够排除，那么需要计算 Query 和 Cluster i 内所有数据的距离，但是仔细观察可以发现 Cluster i 内数据还可以再一次聚类，并且再次聚类后离 Query 较远的另外三个簇内的数据也能够排除，这样又进一步减少计算量。我们称这种再次聚类的方法为层次聚类方法。层次聚类算法首先将所有数据视为一个类，然后将这个类按照一定规则分裂，分裂后的类再进行分裂，直到最小的类不满足分裂要求时停止。在经过多次分裂后所有数据将会变成树状结构，最后返回树的根节点。图 3.5 展示了如何将数据进行层次聚类以及数据的树状结构。

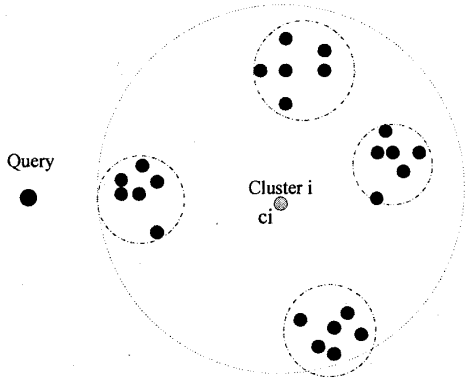
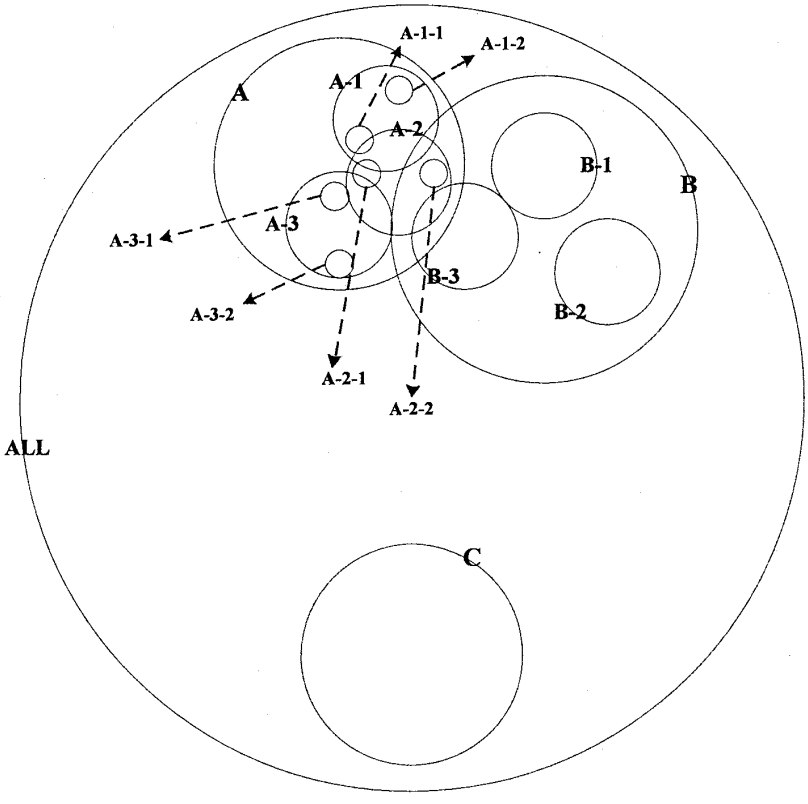
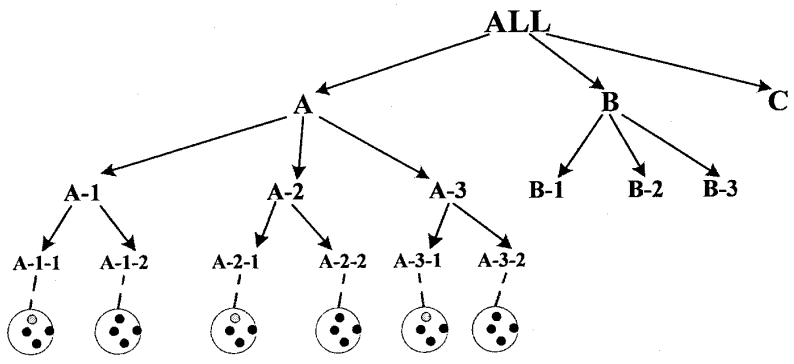


图 3.4 簇内数据分布



(a) 层次聚类后的数据



(b) 数据的树状结构

图 3.5 二维数据层次聚类示意图和数据的树状结构

图 3.5(a)中最大的圆代表所有数据，A,B,C 代表经过最大的类分裂后形成的新类，A-1,A-2,A-3 代表经过 A 分裂后形成的新类，其他情况以此类推。在查找时首先从图 3.5(b)的 ALL 节点开始遍历，然后判断 A,B,C 三个簇中是否有能够通过剪枝规则进行剪枝的簇，假若 B 符合剪枝条件，则 B 的所有子节点（B-1,B-2,B-3）不需要搜索即可排除，假若没有符合剪枝条件的簇，则进行深度优先搜索。表 3.1 展示了层次聚类算法流程。

表 3.1 层次聚类算法

输入：待聚类的数据
输出：树的根节点
1. if(待分裂数据满足分裂条件)
2. 计算所有数据的中心 center
3. 执行 k-means 算法，将数据分裂成 k 份数据
4. 对每一份数据执行层次聚类算法
5. 返回 center
6. else 算法停止

在查找过程中需要利用剪枝规则进行剪枝，下面介绍几种常用的剪枝规则[14]：

假设目前可以得到 Query 的 k 个候选近似近邻。Query 与 k 个候选近邻中最远的距离为 D（表明 Query 距其准确 k 近邻最远的距离要小于等于 D），现有一个类，假设类的中心为 M，类的最大半径为 R（类中所有点距类中心 M 最远的距离），类的最小半径为 r(类中所有点距类中心最近的距离)，d(q,M)为 Query 到 M 的距离。

剪枝规则第一条：

当满足 D+R<d(q,M)时，此时类中不可能包含 Query 的 k 近邻。如下图：

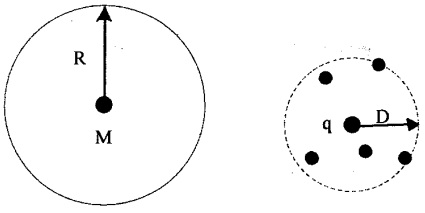


图 3.6 剪枝规则第一条

图 3.6 表明以 M 为类中心 R 为半径的类中所有数据距 Query 的距离要大于 D，所以在该类中不存在 Query 的 k 近邻。

剪枝规则第二条：

当满足 D+d(o,M)<d(q,M)时，o 不可能是 Query 的 k 近邻。如下图：

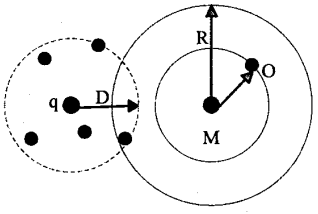


图 3.7 剪枝规则第二条

图 3.7 中 o 代表以 M 为类中心 R 为半径类中的某条数据，其不可能成为 Query 的 k 近邻的原因与剪枝规则第一条相同，剪枝规则第一条可以排除整个类中的数据，而第二条可以排除单条数据。

剪枝规则第三条：

当满足 D+d(q,M)<r 时，类中不可能包含 Query 的 k 近邻。如下图：

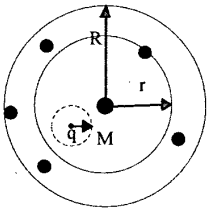


图 3.8 剪枝规则第三条

图 3.8 表明 Query 距类中心 M 极近且 M 到 Query 的距离加上 D 要小于类中所有数据到 M 的距离（即类的最小半径 r）。这样在以 Query 为中心 D 为半径的范围内不可能包含该类中任何数据。

剪枝规则第四条：

当满足 D+d(q,M)<d(o,M)时，o 不可能是 Query 的 k 近邻。如下图：

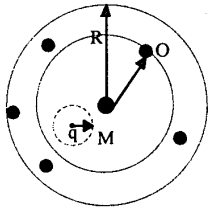


图 3.9 剪枝规则第四条

图 3.9 中 o 为类中任意一条数据，其剪枝的原理与第三条同理，只是第四条剪枝规则从单条数据的角度考虑，第三条从整个类的角度进行考虑。

在数据间距离分布较均匀的情况下，利用剪枝规则可以显著提高查找效率。下面将介绍在上述层次聚类数据结构上采用的深度优先查找策略。在查找 k 近邻之前，我们需要将数据按照表 3.1 所述的层次聚类算法流程进行聚类，然后获取树的根节点。在处理每个 Query 时，首先从根节点进行查找，然后按照需要逐层遍历直到叶子节点。表 3.2 简述了深度优先查找算法流程。

表 3.2 深度优先查找算法

输入：用户 Query，树的根节点
输出：Query 的 k 近邻
1. 计算根节点的孩子与 Query 的距离
2. if(节点不能利用剪枝规则排除)
3. 将节点加入剪枝栈中，更新 Query k 近邻半径 D
4. while(剪枝栈不空)
5. 节点出栈，执行遍历算法（表 3.3）
6. 计算所有不能剪掉的叶子节点与 Query 的距离，返回 k 近邻

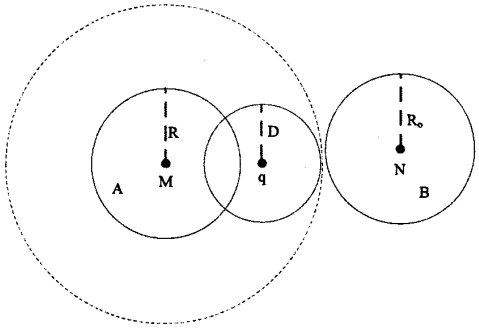
表 3.3 遍历算法

输入：用户 Query，树的节点
输出：剪枝栈
1. if(树的节点为叶子节点)
2. 将节点加入待计算队列中
3. else
4. 计算 Query 与该节点孩子节点的距离
5. if(孩子节点不能利用剪枝规则排除)
6. 则留下一个孩子节点继续执行遍历算法且更新 Query k 近邻半径 D，将不能剪掉的孩子节点加入剪枝栈中

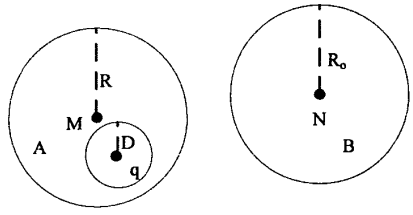
在深度优先查找算法中 Query 的 k 近邻半径 D 可以初始为∞，每当 Query 与节点计算距离时，可以更新半径 D，更新方法为：如果 d(q,M)+R<D，则更新 D 为 d(q,M)+R。M 为某个类的中心，R 为该类的最大半径。剪枝栈中存放的是不能用剪枝规则剪掉的节

点，待计算队列中存放的是待最后计算的叶子节点。

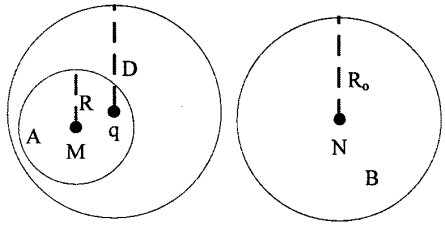
在算法中除了用到的上述四条剪枝规则外，还采用了下述的几种基于相对位置而设计的剪枝规则，利用这几条剪枝规则需要保存同一父节点的孩子之间的距离。基于相对位置的剪枝规则有如下几种情况：



(a) 相交，$d(q,M)+D+R_o<d(M,N)$



(b) Query 在类 A 内部，$R+R_o<d(M,N)$



(c) 类 A 在 Query 内部，$D+R_o<d(q,N)$

图 3.10 Query 与类 A、类 B 的相对关系

上述的三条剪枝规则中，首先确定 Query 与类 A 的位置关系，然后根据 D、d(q,M)和已保存的 d(M,N)来判断类 B 能否被剪掉。在这个过程中不需要计算 Query 与类 B 的距离 d(q,N)就可完成判断。

基于层次聚类的 k 近邻查找算法可以充分利用数据间的距离来建立索引。在能够充分利用剪枝规则的情况下 Query 与数据计算距离的次数为$\log N/p + cp$，N 为数据的数量，p 表示每个叶子节点中数据的数量，c 为不固定的常数，取值为$1 \le c \le N/p$，其取值与数据的分布和 Query 的分布有关。在数据分布不均匀情况下，数据间的距离方差很大，这样剪枝规则可以充分利用数据间距离的差距来减少不必要的计算，此时 c 的取值越接近 1。在数据的维度非常高且数据非常稠密的情况下，数据间的距离差距会变小，这样剪枝规则的作用会减弱，能够排除的数据也变得有限，c 的取值也相对变大，降低了算法的查询效率。在这种情况下返回准确 k 近邻的代价将会变高，近似 k 近邻即可满足用户对返回结果的质量和效率要求。近似 k 近邻查找算法的时间复杂度是次线性的，并且在数据分布均匀情况下其结果的质量可以得到保证。

3.3 密文 k 近邻查询算法

在本节中首先介绍了密文k近邻查询算法流程并对各个参与者需要执行的任务做了详细介绍，然后针对本文的查询流程进行了效率与安全性分析。

3.3.1 密文 k 近邻查询算法流程

基于层次聚类的 k 近邻查询算法主要是为了解决查询效率问题，如果将其直接应用到密文查询中必定会在计算 Query 与数据间距离时泄露数据拥有者的数据，所以必须将数据拥有者的数据和用户提交的 Query 进行加密处理。加密处理首先将数据拥有者的数据按照层次聚类算法流程进行聚类，然后把聚类后的数据进行加密，其中加密的数据包括在聚类过程中产生的新数据（树的非叶子节点，每个类的类中心）和数据拥有者的数据（树的叶子节点中包含的数据）。数据加密后可以将整个加密的索引结构发送到 SP_A，将解密的密钥发送到 SP_B，SP_A 与 SP_B 相互协作处理密文的 Query 和数据，最终返回查询结果。具体流程参见表 3.4。

表 3.4 密文 k 近邻查询算法

输入：用户的加密 Query
输出：Query 的 k 近邻

1. **SP_A**：将加密的 Query 和需要计算距离的数据经过适当处理发送到 SP_B
2. **SP_B**：解密数据，进行距离计算，将计算后的距离结果返回给 SP_A
3. **SP_A**：根据 SP_B 返回的距离结果，计算出真正的距离密文，然后将 Query 与数据的密文距离、Query 的 k 近邻半径 D 和节点的半径发送给 SP_B
4. **SP_B**：根据 SP_A 发送过来的数据进行剪枝计算，将计算结果返回给 SP_A
5. **SP_A**：根据 SP_B 返回的数据确定如何向下遍历，以及哪些节点可以进行剪枝
6. 在剪枝栈为空时，SP_A 与 SP_B 计算待计算队列内的数据与 Query 的距离，计算完成后 SP_A 将随机生成的数据、待计算队列内的数据以及距离发送到 SP_B，将对数据的处理方式发送给用户
7. **SP_B**：对数据进行解密操作，并且按照距离从小到大的顺序分多批将数据返回给用户

在整个查询过程中数据拥有者只参与一次操作，即将数据聚类、加密以及将加密用的公钥发送给用户。

SP_A 接触到的数据全部是加密的数据，这样整个过程中 SP_A 对于数据的内容完全不

能解析，即使在知道 Query 与数据间相对距离的情况下，依然不能破解加密算法获得密钥，从而获取到数据的明文。SP_B 主要的作用是解密数据，虽然是直接解密数据，但是解密后的数据并非原始数据，而是经过 SP_A 处理过的数据，这些数据在解密后在其原有数据的基础上都加入了随机值，这样 SP_B 即使记录下这些加入随机值的数据也不能准确地判断出数据的原始值，从而使原始数据得到保护。SP_A 需要给用户发送其对数据的处理方式，例如告诉用户什么形式的数据是随机生成的数据，真实数据是如何加入随机值的等。用户在收到这些信息以后就可以确定哪些数据是符合要求的以及如何还原出真实数据。

SP_B 在最后解密距离时，获得的密文距离是经过 SP_A 加入扰动值以后的距离，这样可以减少 SP_B 了解到更多的关于数据间距离的信息，但是加入扰动值以后必须确保不能改变 Query 与原始数据距离的相对顺序。例如 Query 与 D1，D2，D3 的距离分别为 1,3,6，加入扰动值以后必须保证 Query 与数据 D1，D2，D3 的距离依然是 d(Query,D1)<d(Query,D2)<d(Query,D3)，若加入扰动值后距离变为 2,5,7 则是符合要求的，而 3,2,6 则是不符合条件的，因为 d(Query,D1)>d(Query,D2)。对于 k 近邻查找而言，距离的真实值并不重要，距离间的相对顺序更为重要。

用户在提交 Query 的时候需要将 Query 进行加密，这样可以防止 SP 了解到用户的查询意图，而这样的查询意图非常容易泄露用户的隐私，并且加密后的 Query 在处理时可以增加 SP 破解数据的难度，例如 SP_B 在计算数据与 Query 距离时，若 Query 没有经过处理，SP_B 则会根据 Query 以及计算的距离推测原始数据的值，增加了数据泄露的风险。用户在获取结果时需要分批次获取，这样处理的目的第一是为了防止用户获得过多的数据，例如用户要求 k=10，但是在获取结果时获取了 100 条结果，这样使得数据拥有者的数据更多的呈现给了用户，对于一些不法分子可以用这些多余的数据设计不同的 Query，从而获得更多非重复数据。这样会损害数据拥有者的利益。第二个原因是为了防止 SP_B 了解到原始数据的分布，对于某些特殊的数据库其数据分布是很容易根据 Query 猜测到的，而分批次返回过程中，每批返回的数据中可能都夹杂着一些 SP_A 随机生成的数据，这些随机生成的数据干扰了 SP_B 对数据分布的判断，从而使得数据拥有者的数据和用户的隐私得到更好的保护。为了防止用户在客户端出现作弊行为，即伪造数据数量没有满足 k 近邻要求，SP_A 在生成随机数据时需要保证最多分几批一定可以满足用户的要求并且将这个值发送给 SP_B，SP_B 根据这个值发送小于等于该值的批次。

下面就表 3.4 中每一步处理的细节做详细介绍。

本文用到的同态加密算法为 Paillier 加密算法。该加密算法属于非对称加密算法，算

法在加密时，首先要创建一个公钥，公钥的创建方法是选择两个比较大的素数 p 和 q，计算这两个素数的乘积 p.q=n，然后选择一个非零的整数 g， $g \in \mathbb{Z}_{n^2}^*$，(n,g) 构成了加密用的公钥。数据拥有者构造好（n,g）后将其发送给 SP_A，SP_B 和用户。而密钥就是构造 n 所用的两个大的素数 p,q 和 g。数据拥有者将这个密钥发送给 SP_B。在加密的过程中还需要所取一个随机数 r，r 只有在加密时使用，解密不需要知道且不会影响解密后的数据。加密公式为 $c \equiv g^D \cdot r^n \bmod n^2$，D 代表数据拥有者的数据，c 代表数据加密后的密文。下面是一个具体加密的例子，更多的信息请参考文献[50,4]。

假设 p=7，q=11，n=77，g=5652，r=23，D=42。那么 $c \equiv (5652)^{42} \cdot (23)^{77} \bmod 5929 \equiv (4019)(606) \equiv 4624 \bmod 5929$。42 在加密后为 4624。

在选择两个素数时不应该选取例子里这么小的素数，过小的素数非常容易被破解。本文用到的加密算法有几个非常重要的性质，主要包括加法同态性（Homomorphic Addition），乘法同态性（Homomorphic Multiplication）和语义安全性（Semantic Security）。假设 E 代表加密函数，E(x)代表 x 加密后的密文，那么

加法同态性：

$$E(a+b) = E(a) * E(b) \ mod \ n^2 \tag{3-4}$$

即 a 的密文乘以 b 的密文模 n^2 等于 a+b 的密文。

乘法同态性：

$$E(a*b) = E(a)^b \ mod \ n^2 \tag{3-5}$$

即密文 a 的 b 次幂模 n^2 等于 a 乘 b 的密文。

语义安全性：攻击者在获得一组密文时，不能根据密文推测出明文。

下面将介绍利用 SM(Secure Multiplication)算法来安全地计算加密的 Query 与数据的距离。

假设 SP_A 拥有两个加密过的数字 E(a)，E(b)，SM 算法的作用是经过 SP_A 与 SP_B 的合作，来计算 E(a*b)，计算过程中 SP_A 和 SP_B 不能获得明文数据的内容。SM 之所以能够完成安全计算，主要是应用下面的等式：

$$a*b = (a+r_a)*(b+r_b) - a*r_b - b*r_a - r_a*r_b \tag{3-6}$$

SM 算法流程见表 3.5

表 3.5 SM 算法

输入：*E(a), E(b)*
输出：*E(a*b)*
1. SP_A:
a) 随机生成两个随机数 $r_a, r_b \in \mathbb{Z}_N$
b) $a' = E(a) * E(r_a)$ （式 3-4）

c) $b' = E(b) * E(r_b)$

2. **SP_B**：根据从 SP_A 获得的$a'b'$做如下运算
   a) $h_a = D(a'),\ h_b = D(b')$
   b) $h = h_a * h_b \bmod n$
   c) $h' = E(h)$
   d) 将h'发送到 SP_A
3. **SP_A**：根据从 SP_B 返回的数据h'做如下计算
   a) $s = h' * E(a)^{n-r_b}$ （式 3-5）
   b) $s' = s * E(b)^{n-r_a}$
   c) $E(a*b) = s' * E(r_a * r_b)^{n-1}$

在 SM 算法中 SP_A 首先生成两个随机数r_a，r_b，然后将加密的 $E(r_a)$与 $E(r_b)$分别与 E(a)，E(b)相乘获得a′与b′。$a' = E(a + r_a), b' = E(b + r_b)$，即$a + r_a$的密文与$b + r_b$的密文。$SP_A$ 将计算后的结果发送给 SP_B。SP_B 用其私钥将a', b'进行解密，解密以后计算 $h = (a + r_a) * (b + r_b)$，然后将h的密文h′发送给 SP_A。SP_A 在获得数据后分别计算 $s = E\big((a + r_a) * (b + r_b) - a * r_b\big), s' = E\big((a + r_a) * (b + r_b) - a * r_b - b * r_a\big)$，最后根据式(3-6)计算出$E(a * b)$。

整个计算过程中 SP_A 接触到的数据全部是密文形式，而 SP_B 虽然能够解密数据，但是其获得的数据已经加入了随机值，不能知悉具体的明文。在能够安全地计算两个加密数字乘积的前提下，就可以安全地计算E(Query)与E(Di)的距离了。

$$dis(Query, Di) = (q_1 - d_1)^2 + (q_2 - d_2)^2 + \cdots + (q_m - d_m)^2 \tag{3-7}$$

式(3-7)中的距离公式并没有开根号，主要原因是查询过程中只需要比较距离的大小而不需要知道距离的具体值，并且对于任意两个正数x, y，若$x > y$，那么$\sqrt[2]{x} > \sqrt[2]{y}$一定成立。$SP_A$ 拥有$E(q_1)$与$E(d_1)$，可以根据加密算法的同态性来计算$E(q_1 - d_1)$，然后再根据 SM 算法通过与 SP_B 合作计算出$E((q_1 - d_1)^2)$，这样就可以计算出 Query 与 Di 的加密距离。具体算法流程见表 3.6。

表 3.6 安全计算两条数据的距离算法

输入：*E(Query)，E(Di)*
输出：*E(dis(Query,Di))*
1. **SP_A**：for *i=1:m* do
   $E(q_i - d_i) = E(q_i) * E(d_i)^{n-1}$
2. **SP_A** 与 **SP_B** 利用 SM 算法计算$E((q_i - d_i)^2)$
3. **SP_A**：计算$E\big(dis(Query, Di)\big) = \prod_{i=1}^{m} E((q_i - d_i)^2)$

安全计算 Query 与 Di 加密距离算法的大体流程是 SP_A 首先计算出$E(q_i - d_i)$，然后 SP_A 与 SP_B 相互合作利用 SM 算法计算$E((q_i - d_i)^2)$，最后 SP_A 再根据加密算法的同态性计算出 Query 与 Di 的加密距离。

SM 算法与安全计算两条数据距离算法解决的主要问题是计算两条加密数据的加密距离，对应着表 3.4 的第 1,2,3,步。SM 算法在计算过程中需要执行多次加密解密计算，而加密解密计算相对其他计算更耗时，所以在实现时可以采用 Liu an 等人[42]提出的数据 packing 技术来对数据进行加解密。数据 packing 技术的提出主要基于以下两点原因，第一，SM 算法加密解密次数较多，耗时严重。第二，加密算法加密的数据大小一般是 1024bits，但是在正常加密时有效利用的往往小于 1024bits，所以可以采用数据 packing 的技术对空间进行充分利用，这样可以显著减少计算耗时。

在这个过程中 SP_A 最终获得了加密距离，但是对于利用剪枝规则还需要判断数值之间的大小，由于 SP_A 没有解密的密钥，所以 SP_A 需要将 Query 与 Di 的加密距离、遍历过程中 Query 的 k 近邻半径 D 以及树的非叶子节点的半径 R 发送到 SP_B。SP_B 收到数据以后首先解密获得明文的 Query 与 Di 的距离，因为 Di 是非叶子节点的中心并且 SP_B 也不了解 Di 与 Query 的明文内容，所以 Query 与其的距离不能够被 SP_B 利用。SP_B 解密距离后需要判断哪条剪枝规则成立，并且将判断的结果发送给 SP_A，SP_A 再确定哪些节点需要继续向下遍历，哪些节点需要加入剪枝栈中。该过程对应表 3.4 中第 4,5 步。

SP_A 在多次遍历后，需要计算待计算队列内数据与 Query 的距离，并且将最终结果发送给用户。SP_A 与 SP_B 按照安全计算两条数据算法流程进行距离计算。SP_A 在最终获得加密距离后，将 Query 与数据的密文距离中加入一个随机值，例如 $E(new\_dis) = E(dis) * E(r) \bmod n^2 = E(dis + r)$，加入随机值可以防止 SP_B 在解密时获得真实距离，使得数据拥有者的数据得到更好的保护。除了处理加密距离，还要处理待计算队列中的数据，可以将数据的每一维度都加入随机值，这样 SP_B 在解密后获得的并非原始数据的真实值，在数据加入随机值后 SP_A 需要将相应的随机值发送到用户端，这样在用户获得解密数据后可以还原数据的真实值。为了防止 SP_B 统计 Query 与数据的对应关系，SP_A 可以向数据中加入一些随机生成的数据，并且为每个随机生成的数据生成一个假的 Query 与数据的加密距离，这样可以干扰 SP_B 的判断。SP_A 需要对随机生成的数据做标记，然后将标记的方法发送到用户端，用户根据相应的标记过滤掉随机生成的数据。该过程对应表 3.4 中第 6,7 步。

3.3.2 安全与效率分析

查询算法的安全性可以从两方面进行考虑，第一是考虑加密算法的安全性，第二是考虑查询策略的安全性。由于本文采用的加密算法具有语义安全性，所以这里主要考虑

查询策略的安全性。

Yao 等人[7]利用选择明文攻击证明了 Hu 等人[6]的工作不能保证数据安全，虽然采用的是选择明文的攻击方式，但是攻击的并不是加密算法，而是 Hu 等人采用的查询策略。该方式有效的主要原因是在 Hu 的方法中客户端需要借助服务端比较两条密文对应明文的大小，我们知道利用二分查找很容易确定一条数据的位置，所以为了避免此情况发生，在我们的查询策略中不会出现两个服务端交互比较大小的过程，也就保证了 SP_A 不能通过 SP_B 以合理的方式还原数据。

在整个查询过程中 SP_A 接触的均是加密数据，SP_B 虽然具有解密的密钥，但是其接触的数据均是 SP_A 加入了随机值的数据，所以 SP 不能直接获得明文的数据。在计算过程中 Query 的 k 近邻半径以明文方式传递，每个类的半径以及 Query 与树中每个节点的距离最终会被 SP_B 知道。虽然知道这些中间结果，但是并不能根据这些来推断数据本身的内容。如果为了防止 SP_B 获悉 Query 与真实数据的距离，SP_A 可以在将 Query 与待计算队列内数据的距离发送到 SP_B 前，每次向所有距离中加入一个相等的随机值，这样既保护了真实距离，又不能打乱距离的顺序。在返回结果的过程中 SP_A 向原始的密文数据中每个维度都加入随机值，这样做的目的有两个：

第一是为了防止 SP_B 获得真实数据，因为每个维度都加入了随机值。

第二是为了防止 SP_B 了解 Query 对应的结果。相同的 Query 虽然返回相同的结果，但是在经过加入随机值处理后，即使是相同的 Query，SP_B 解密后看到的结果也是不同的结果，因为每次查询都会向每个维度加入不同的随机值，这样使得用户的信息、数据库中数据分布的信息得到了保护。

SP_A 也同样不能知道 Query 对应的结果，因为待计算队列中数据的数量要大于 k，并且 SP_A 只知道 Query 与真实数据的加密的距离，所以其不能根据距离进行排序，也就不能知道具体返回了哪几条结果。所以本文的方法不仅可以保证数据拥有者数据的安全，还可以使得用户的隐私得到保护。

数据拥有者在上传数据到 SP_A 前，需要将数据聚类加密，这个过程只需要执行一次，并不会影响 k 近邻查询的时间，所以在计算查询时间时可以忽略。SP_A 与 SP_B 在计算过程中涉及到的数据加密解密的次数依赖于 Query 与数据的分布，每个 Query 需要计算距离的次数为$\log N/p + cp$，那么加密解密的次数为$O(m * ((\log N/p) + cp))$，N 为数据的数量，p 表示每个叶子节点中数据的数量，c 为不固定的常数，取值为$1 \le c \le N/p$，其取值与数据的分布和 Query 的分布有关，m 代表数据的维度。由于维灾难的影响，在数据维度 m 很大的情况下，后面的比较次数$\log N/p + cp$也会增大，这样会导致加密解密次数较多，

时间消耗变大，这种情况可以考虑近似 k 近邻查询。若是在 m 很小的情况下，整体的加密解密次数都会降低，查询效率较高。然而这些计算都在 SP 端完成，对于用户是透明的，提高 SP 端服务器的计算能力可以使得用户获得更好的体验。用户只需要做简单计算就可以获得准确的结果，所以算法对于用户端的计算能力要求不高。

3.4 本章小结

本章主要介绍了基于层次聚类的密文 k 近邻查询算法。

基于层次聚类的 k 近邻查询算法主要是为了解决查询效率问题。算法通过利用数据间距离建立索引，利用深度优先查找策略逐层进行遍历，在遍历过程中利用剪枝规则进行剪枝，可以有效地减少 Query 与数据的距离计算，从而提高了查询效率。在数据维度较低的情况下，算法能够充分利用剪枝规则，但是在维度增高到一定值时，维数灾难就会出现，这种情况下所有基于距离的索引都会出现效率问题，这种情况下建议采用近似 k 近邻查询算法。近似 k 近邻算法查询效率高，在数据维度较高情况下返回结果的质量也可以得到保证，并且近似 k 近邻算法很容易与安全计算密文数据距离的算法相结合完成加密数据的 k 近邻查询。

在密文 k 近邻算法中主要介绍了整个查询流程，其中包括用户如何提交 Query，SP 如何相互合作安全地计算数据间的距离来进行剪枝、遍历节点以及如何向用户返回结果等流程。用户首先用公钥加密 Query 发送到 SP_A，SP_A 在收到加密 Query 以后与 SP_B 相互合作查找 Query 的 k 近邻。密文 k 近邻查询算法能够有效保护数据拥有者的数据以及用户的隐私。SP 在整个计算过程中不能获得数据的真实内容，即使在 SP 收集中间计算结果的情况下，SP 也不能还原数据的原始内容。密文 k 近邻的时间复杂度取决于 Query 与数据的分布与以及数据的维度，在维度较低的情况下，层次聚类 k 近邻算法能够充分利用剪枝规则。这样 SP 加密解密的次数就会减少，从而提高查找效率。

第 4 章 实验结果及评估

本章主要介绍实验环境、实验采用的数据集、不同参数对实验结果的影响以及与最近研究成果的对比。影响实验结果的参数有很多，例如不同的数据维度，不同密钥长度以及 k 近邻查询中不同的 k 值均会影响实验的结果。下面将从各个不同的层面进行详细分析。

4.1 实验环境及实验数据

本文所有实验程序均用 Java 程序设计语言编写，JDK 版本号为 1.8，实验中所用的 CPU 为 Intel Core i3-370，主频为 2.4GHz，双核，内存 6G，操作系统为 Windows 8.1。

加密数据 k 近邻查询研究没有统一的数据集进行结果评估，所以本文数据集由合成数据集与真实数据集组成。在加密数据 k 近邻查询中有非常多的实验参数影响实验结果的评估。可以通过控制不同的实验参数生成不同的数据集，这样可以对算法进行较为详细的评估。本文中合成数据包含了不同数据维度，不同数据分布的数据。不同的数据维度与不同的数据分布均会对剪枝规则的利用情况产生影响，如果剪枝规则能够充分利用那么就会减少数据间距离计算次数，如果利用的不充分，则会增加加密解密的次数。在计算过程中加密解密会产生较大的计算量，增加了算法的查询时间，并且加密的密钥位数越大所耗费时间越长。所以对于分析数据维度与数据分布如何影响算法查询效率十分必要。合成数据集中包含 2-20 维的数据，在数据均匀分布中维灾难会在 10-15 维中出现，并且我们的实验结果也表明在数据 10 维左右时，数据之间的距离变化会明显减少。

实验中所有数据维度均进行规范化处理，使各个维度的值在[0,3000]之间。真实数据集我们采用 Color[51]和 Mnist[52]。Color 是一个 32 维的数据集，包含 68040 条数据。每一条数据描述的是一幅图像的颜色直方图。Mnist 是一个手写数字识别数据集，每条数据由 784 维数据构成，包含一幅手写数字图像的 28×28 个像素，在 Mnist 中包含 60000 条训练数据，10000 条测试数据。在使用这两个数据集之前，本文均对其做降维处理，分别降维到不同维度以观察实验的效果。

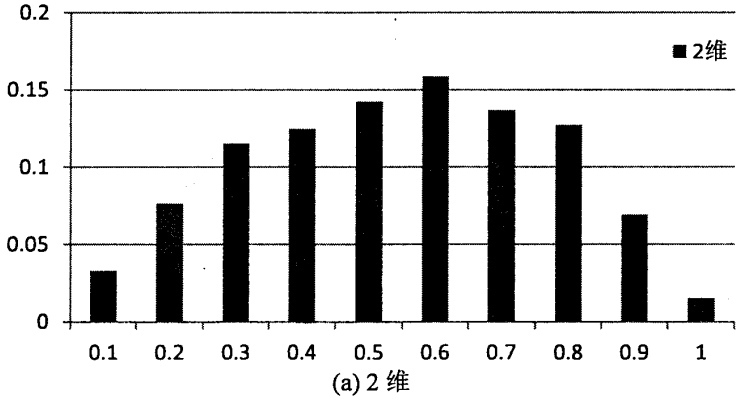
4.2 数据维度对算法性能的影响

这节内容将主要探讨数据维度对算法性能的影响。根据文献[22]在均匀数据中，随着数据维度的升高，数据间距离的变化将显著减小，在非常极端的情况下，即当数据的

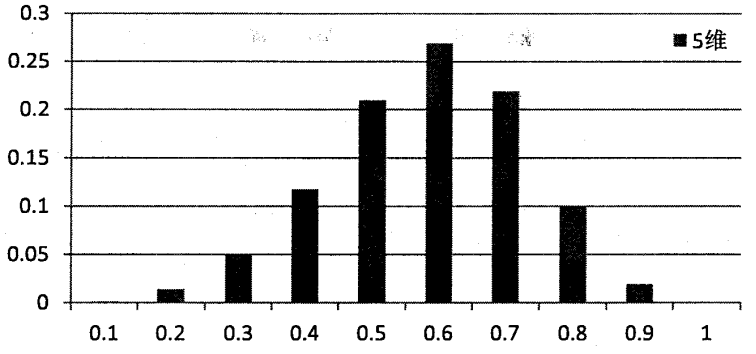
维度趋向于无穷时，数据间的距离几乎相等。在文献[22]中作者假设所有数据服从均匀分布，且数据的维度趋向于无穷。这样的假设虽然在理论上阐明了数据间距离分布的规律，但是缺少现实意义。现实世界中的数据维度通常较小，例如地理空间维度只有三维，经纬度只有二维，对于一些查询用户间相似度的网站来讲，每个用户的限制条件也是有限的，这样导致每个用户向量的维度也是有限的。并且对于现实世界中不同的数据集，数据会有不同的分布，这样就导致了数据间距离分布的差异。

假设数据的每个维度规范化到[0,1]之间，且数据均匀分布在 D 维的单位立方体中。在我们进行 k 近邻查询时，假若 k 的值占数据集中数据数量的 10%。在数据为一维的情况下，也就是一条 0 到 1 的线段，我们只需要取出 0.1 长度内的数据即可满足 k 值的要求，也就是说检索出来的 k 近邻距 Query 最远为 0.1。在二维数据中，也就是在一个面的情况下，若要检索取出所有数据的 10%作为 Query 的 k 近邻，则每个维度的长度至少为 0.32，因为 0.32*0.32≈10%。在 10 维的情况下，要求每个维度的长度将会更长，达到 0.8。即使在降低 k 值的情况下，每个边的长度也会很大。例如不用所有数据的 10%作为 Query 的 k 近邻，只取所有数据的 1%，即使在这种 k 值是原来十分之一的情况下，在 10 维中每个维度需要的长度也将达到 0.63。但是数据中每个维度的长度只为 1，似乎违背了近邻的要求，在这种情况下选取出来的 k 近邻将会缺乏代表性，失去了 k 近邻原有的意义。

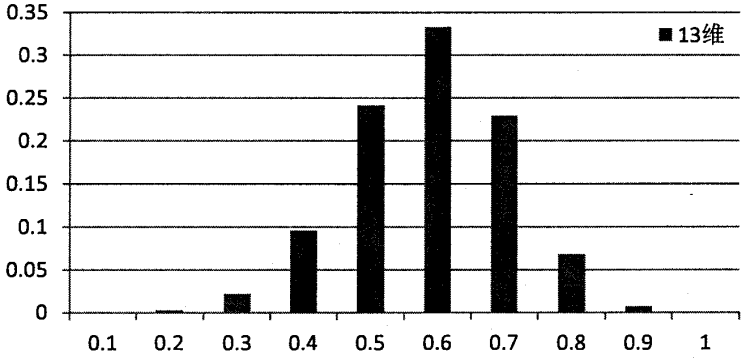
在图 4.1 中描绘了不同数据维度数据间距离的分布。图 4.2 内描绘了不同数据的维度对算法查询效率的影响。



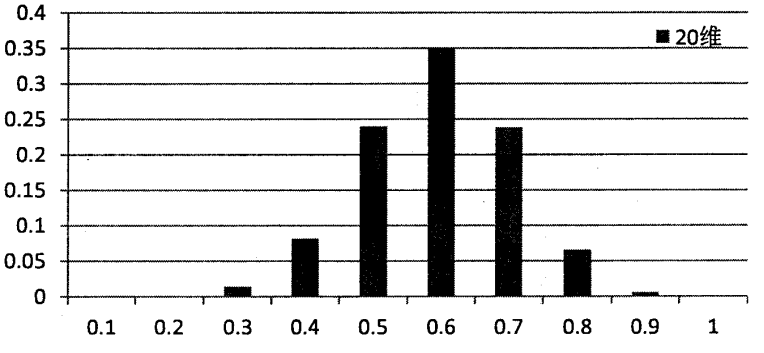
(a) 2 维



(b) 5 维



(c) 13 维



(d) 20 维

图 4.1 不同数据维度数据间距离分布

图 4.1 中横坐标代表距离归一化后的值，即随机从数据集中选取一条数据，与所有的数据进行距离计算，得到所有距离后，利用公式(4-1)对距离进行归一化处理，归一化以后再进行分箱处理，即每隔 0.1 个距离单位统计在此范围内有多少条数据。为了显示更加直观，将数据数量的范围也进行了归一化处理。在式（4-1）中 dis_{max} 代表所有距离中的最大值，dis_{min} 代表所有距离中的最小值，dis 代表某个距离值。

$$\frac{dis-dis_{min}}{dis_{max}-dis_{min}} \tag{4-1}$$

从图 4.1 中可以看出图 4.1(a)和图 4.1(b)中数据的距离分布相对均匀一些，而图 4.1(c)与图 4.1(d)中数据距离的分布多集中于 0.5-0.7 这个范围内。这一实验结果可以说明，随着数据维度的升高，数据间距离分布较集中。在 2 维与 5 维数据中，因为选取的数据与其他数据在计算距离后，这些距离值差别较大，这就使得在 2 维数据中每个区间数据数量差别较小，距离分布更加均匀。而在 13 维与 20 维数据中，选取的数据和其他数据计算出距离值后，这些距离值大部分相差较小，这就使得在图 4.1(c)与图 4.1(d)中这些距离值分布更加的集中。并且在图 4.1(c)与图 4.1(d)中可以发现大部分数据是出现在连续的范围内，例如图 4.1(c)与图 4.1(d)中大部分数据出现在 0.5-0.7 范围内。连续的范围代表距离的值也是连续的，也就是说随着数据维度的升高，数据间距离的值趋向于某个范围内，并且维度越高这个范围将会越小。在极端情况下数据间距离的差值将会越趋向于 0。此时 k 近邻查询将会失去意义，因为任何一条数据与 Query 的总体差别都很小，不能选择出较有代表性的数据。

数据的距离均匀地分布在每个区间，说明可以更加充分地利用基于距离的索引来检索数据。而在数据间距离分布较集中的情况下，基于距离索引方法的查询效率将会下降。在图 4.2 中验证了这一结论的正确性。

图 4.2 中横坐标代表数据的维度，纵坐标代表不同维度下算法查询 k 近邻花费的时间。该组实验中数据数量为 10000，密钥长度为 512bits，k=5，采用的生成的数据集。在每个维度下，随机选取 50 个 Query，然后分别计算算法查询 k 近邻花费的时间，最后再计算出所有的查询时间的平均值，以此平均值来代表每个维度下算法的查询时间。

在图 4.2 中分别展示了采用了基于层次聚类的密文 k 近邻查询算法与未采用该算法在不同维度下的查找时间情况。可以看出在未采用基于层次聚类的密文 k 近邻算法的情况下，查询时间几乎随着数据的维度升高成线性增长。产生这一结果的主要原因是在数据的整体数量、k 值与密钥长度不变的前提下，由于数据维度的增加，算法需要对增加的维度进行加密解密操作，所以会随着维度增加算法的查找时间成线性增长。而基于层次聚类的密文 k 近邻查询算法在 10 维以内时，算法的查找时间与未采用层次聚类算法的查找时间相差较大，而在 10 维以后将会随着维度的增加，两个算法的查找时间逐渐接近。产生这一结果正与前面论述的不同数据维度数据间距离分布的结果相符。在 10 维以内时，数据间距离分布较均匀，Query 与数据计算距离后，距离间的差值较大，能够充分利用剪枝规则剪掉更多的非 k 近邻数据。而在 10 维以后，数据间的距离值分布较集中，剪枝规则的作用将会减小，所以增加了算法的查找时间。在 17 维时剪枝规则能够剪掉的数据非常少，所以导致与未采用层次聚类方法的查询时间相近。不过从低维

角度来看，基于层次聚类的方法较普通的方法有显著性的优势，明显提高了算法的查询效率。

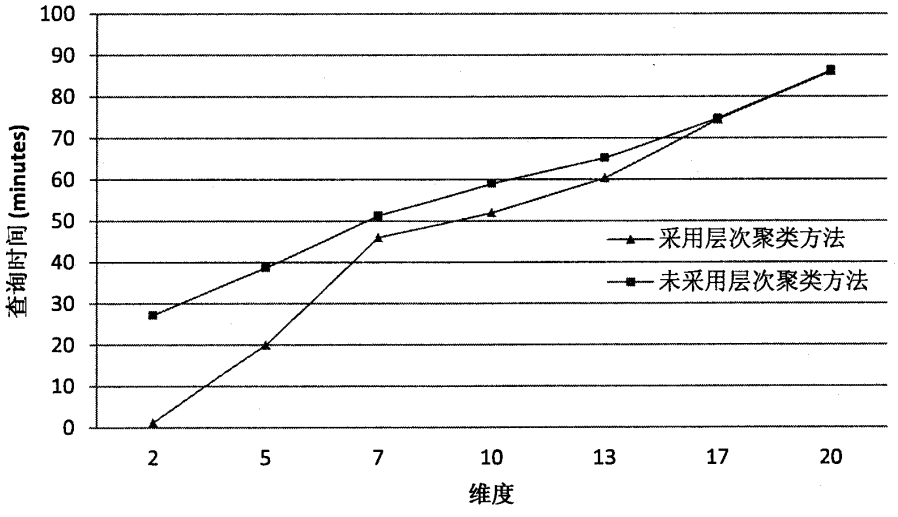


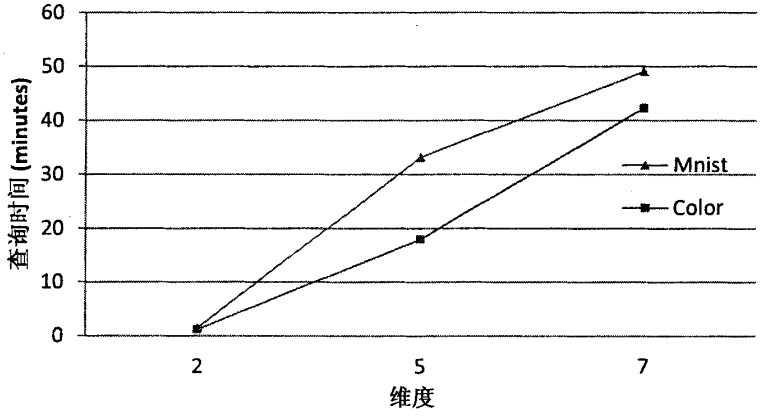
图 4.2 不同数据维度算法查询时间

4.3 基于层次聚类算法在 Color 与 Mnist 数据集上效果评估

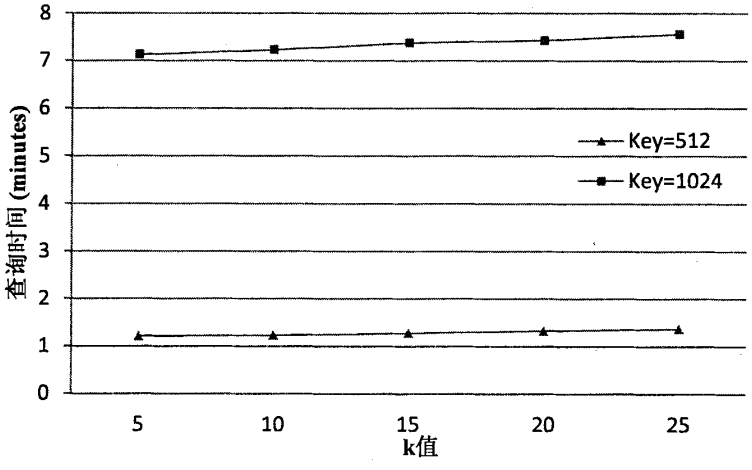
在本节的实验中，我们首先评估了算法在 Color 与 Mnist 两个数据集上的表现，然后分析了算法在不同的数据维度、不同的 k 值、不同的数据数量以及不同的密钥长度上的性能，最后与文献[3]中的 $SkNN_b$ 算法进行了对比，通过实验结果可以发现本文提出的基于层次聚类密文 k 近邻算法具有较高的查询效率。

图 4.3 中描绘了算法在不同的参数下的查询时间，其中 N 代表实验中使用数据的数量，Key 代表加密算法中密钥的长度，dimension 代表实验中数据的维度，dataset 代表使用的数据集，k 代表 k 近邻中的 k 值。

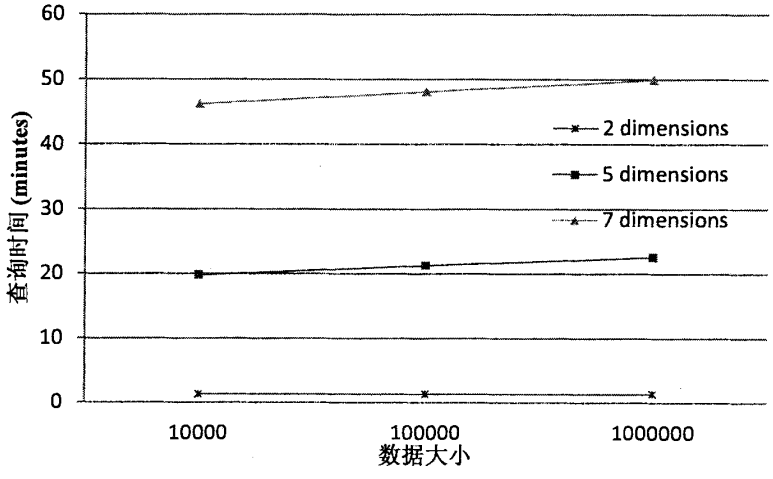
在图 4.3(a)中对比了算法在 Mnist 与 Color 数据集的查询时间情况。在该组实验中密钥长度为 512bits，k 等于 5，数据的数量为 60000。从数据集中随机选取 50 条数据当作 Query，然后将每个 Query 的查询时间相加求取平均值当作在每个维度下的查询时间。从图中可以看出即使在相同数据维度下，算法在两个数据集上的表现也有一些差异。产生这种差异的主要原因是不同的数据集中数据的分布不同，不同的数据分布将会导致数据间距离分布存在差异，从而影响算法的查询效率。



(a) Key=512bits, k=5, N=60000



(b) N=60000, dataset=Mnist, dimension=2



(c) Key=512bits, dataset=合成数据, k=5

图 4.3 不同参数下算法的查询时间

图 4.3(b)中描绘了在不同的 k 值与不同的密钥长度情况下算法的查询效率情况。其中数据数量为 60000，数据维度为 2，数据集为 Mnist。从图中可以看出在密钥长度为 1024bits 时，算法的查询时间将会显著增加，在这种情况下若想提高算法的查询效率只能牺牲一定的安全性，将密钥长度设置为 512bits。可以根据具体情况进行不同的选择，若是查询效率要求较高，则选择 512bits 的密钥长度，若是安全性较为重要，则可以选择 1024bits 的密钥长度。从图中还可以看出随着 k 值的增加，在密钥长度一定情况下，算法的查找时间几乎不变。产生这种结果的原因是基于层次聚类的密文 k 近邻查询算法的查询时间对于 k 值不敏感。对于查找 Query 的 5 近邻与 10 近邻，在这两个查找过程中，每一步内 k 近邻半径更新的值是一样的，换句话说这两个查找过程以相同的路径遍历了整棵树，所以最终花费的时间相差不多。在遍历数据的过程中，我们采用的剪枝规则每次只剪掉树形结构中的非叶子节点，可以不针对某一条数据进行剪枝，并且在聚类过程中只要确保每个叶子节点内的数据大于最大的 k 值，就可保证算法在最终返回准确的结果。

图 4.3(c)中描绘了算法的查询时间随数据数量变化的情况。该组实验中密钥长度为 512bits，k=5，采用的生成的数据集。从图中可以看出随着数据数量的增多算法的查询时间增长的较缓慢。算法的加密解密次数为$O(m*(\log N/p+cp))$，在低维情况下，m 固定且较小，这样可以充分利用剪枝规则，所以使得 c 值也较小，当只有 N 增加时，整个算法的加密解密次数是成非线性增长的。

在本文中我们只与 $SkNN_b$ 算法进行了查询效率的对比，因为 Hu[6]和 Wong[36]等人的算法存在安全性问题，Yao[7]等人提出的算法虽然安全，但是其只支持 2 维数据中最近邻查找。

图 4.4 中描绘了基于层次聚类的密文 k 近邻查询算法与文献[3]中的 $SkNN_b$ 算法对比情况。在该组实验中密钥长度为 512bits，k=5，数据数量为 60000，数据集为 Color。可以从图中看出本文提出基于层次聚类的密文 k 近邻查询算法较 $SkNN_b$ 算法有显著的优势，查询效率较高。$SkNN_b$ 算法花费较高时间获得查询结果的原因是并没有采取适当的优化策略，每次进行 k 近邻查询时，需要比较所有的数据，并且在距离比较的过程中，SMIN 算法也会花费一定的时间，整个查询的过程加密解密次数非常多，所以导致整体查询时间较长。而本文的基于层次聚类的密文 k 近邻查询算法首先将数据进行聚类，然后在查询过程中 SP_A 与 SP_B 利用剪枝规则将大部分非近邻数据剪掉，有效地减少了无用数据的加密解密计算，从而减少了查询时间。

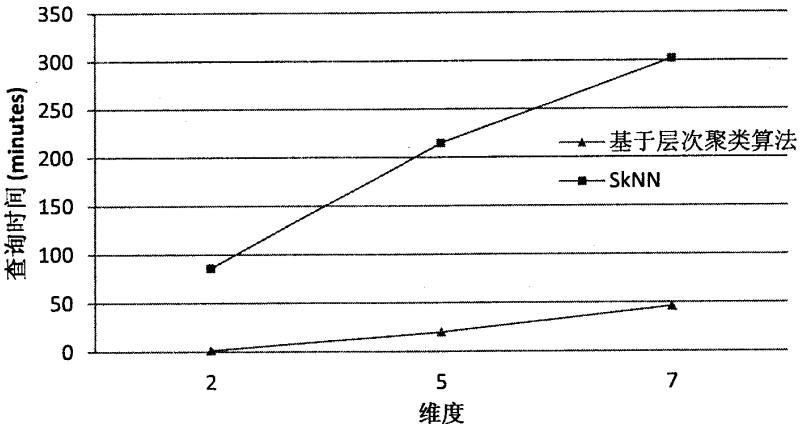


图 4.4 基于层次聚类算法与 SkNN 算法对比情况

4.4 本章小结

本章首先介绍了实验环境及实验用的数据。由于密文 k 近邻计算没有统一的数据集进行验证，为了更好的评估本文提出的方法的实验效果以及在真实数据上的表现，所以本文中分别采用了人工合成的数据集与真实数据集来验证算法的查询效率。

其次针对数据维度对算法性能的影响做了评估。实验中我们验证了数据的维度越高，数据间距离分布越集中，数据维度越低，数据间距离的分布越均匀这一规律。对于 k 近邻查询而言，在高维数据中查找 k 近邻没有意义，因为不能检索出具有代表性的数据。对于本文提出的算法而言，在数据低维时可以充分利用剪枝规则进行剪枝，减少数据间的比较次数，提高查询效率。在高维时算法效率较低，此时可以降低要求，利用在明文上查找近似 k 近邻方法与密文计算两条数据距离算法相结合来满足查询的要求。

最后本文对文献[3]中提出的 $SkNN_b$ 算法与本文的提出的算法进行了比较，得出结论是基于层次聚类的密文 k 近邻查询算法具有较高的查询效率，其中主要原因是本文算法利用了剪枝规则，排除了大部分非近邻数据，减少了对数据的加密解密操作，从而提高了查询的效率。

结　论

本文主要研究在加密数据上进行 k 近邻查询。在加密数据计算中涉及的参与者主要有三个：用户、数据拥有者、服务提供商。用户主要提交 k 近邻查询请求，数据拥有者拥有数据，但是介于自身计算资源的限制，需要将数据托管到服务提供商处，服务提供商进行数据操作来完成用户的 k 近邻请求。这个过程中为了防止服务提供商窃取数据拥有者的数据以及窥探用户的隐私，本文提出了基于层次聚类的密文 k 近邻查询算法。

本文的研究思路是从两方面考虑：首先是查询效率，因为查询效率高可以提高用户的体验，并且节省服务提供商的计算资源，避免了资源浪费。然后是基于安全方面的考虑，数据拥有者的数据对其自身来讲拥有巨大的价值，如果数据泄露或者被他人窃取，不但给用户的隐私带来威胁，还使得数据拥有者遭受巨大的损失。基于效率的考虑本文提出了基于层次聚类的 k 近邻查询算法。该算法是一种在基于层次聚类的数据结构上进行特定查询的算法，此算法是一种查询策略，不仅适用于明文查询，也同样适用于密文查询。介于在密文上查询涉及到如何安全计算两条数据的距离问题，所以本文利用了加密算法的同态性来安全计算两条密文数据的距离。最后将该方法与基于层次聚类的 k 近邻查询算法相结合，构成了基于层次聚类的密文 k 近邻查询算法。使得不仅解决了查询效率问题，而且保证了数据的安全性。

在基于层次聚类的 k 近邻查询算法中，首先是数据拥有者将数据按照层次聚类算法进行聚类。在获得层次聚类结构以后，利用深度优先算法进行遍历数据。遍历数据的过程中，服务提供商利用剪枝规则对数据进行剪枝，对于最终未能剪掉的数据需要与 Query 计算距离，最后根据距离排序返回给用户正确的结果。

最后本文从多个方面对算法的效果进行了评估。首先评估了数据维度对算法性能的影响，得出了在数据维度较低时，算法查询效率较高的结论，并且也通过实验验证了数据在不同维度时，数据间距离分布的规律。然后评估了不同的参数对算法性能的影响，例如不同密钥长度、不同的 k 值、不同的数据分布等。最后对比了本文提出的方法与文献[3]中方法的查询时间情况。从实验结果可以看出，本文提出的方法具有较高的查询效率。

参考文献

[1] 谷利泽, 郑世慧, 杨义先. 现代密码学教程[M]. 北京邮电大学出版社, 2009.

[2] Rivest R L, Adleman L, Dertouzos M L. On data banks and privacy homomorphisms[J]. Foundations of secure computation, 1978, 4(11): 169-180.

[3] Elmehdwi Y, Samanthula B K, Jiang W. Secure k-nearest neighbor query over encrypted data in outsourced environments[C]//Data Engineering (ICDE), 2014 IEEE 30th International Conference on. IEEE, 2014: 664-675.

[4] Paillier P. Public-key cryptosystems based on composite degree residuosity classes[C]//Advances in cryptology—EUROCRYPT'99. Springer Berlin Heidelberg, 1999: 223-238.

[5] Gentry C. A fully homomorphic encryption scheme[D]. Stanford University, 2009.

[6] Hu H, Xu J, Ren C, et al. Processing private queries over untrusted data cloud through privacy homomorphism[C]//Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011: 601-612.

[7] Yao B, Li F, Xiao X. Secure nearest neighbor revisited[C]//Data Engineering (ICDE), 2013 IEEE 29th International Conference on. IEEE, 2013: 733-744.

[8] Fukunaga K, Narendra P M. A branch and bound algorithm for computing k-nearest neighbors[J]. Computers, IEEE Transactions on, 1975, 100(7): 750-753.

[9] Kamgar-Parsi B, Kanal L N. An improved branch and bound algorithm for computing k-nearest neighbors[J]. Pattern recognition letters, 1985, 3(1): 7-12.

[10] Murphy O J, Selkow S M. The efficiency of using kd trees for finding nearest neighbors in discrete space[J]. Information Processing Letters, 1986, 23(4): 215-218.

[11] Broder A J. Strategies for efficient incremental nearest neighbor search[J]. Pattern Recognition, 1990, 23(1): 171-178.

[12] Samet H. K-nearest neighbor finding using MaxNearestDist[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2008, 30(2): 243-252.

[13] Hjaltason G R, Samet H. Distance browsing in spatial databases[J]. ACM Transactions on Database Systems (TODS), 1999, 24(2): 265-318.

[14] Hanan Samet 著, 周立柱, 王宏等译.多维与度量数据结构基础[M]. 清华大学出版社,

2011.

[15]何洪辉, 王丽珍, 周丽华. pgi-distance: 一种高效的并行 KNN-join 处理方法[J]. 计算机研究与发展, 2015, 44(10): 1774-1781.

[16]Xia C, Lu H, Ooi B C, et al. Gorder: an efficient method for KNN join processing[C]//Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. VLDB Endowment, 2004: 756-767.

[17]Zhang C, Li F, Jestes J. Efficient parallel kNN joins for large data in MapReduce[C]//Proceedings of the 15th International Conference on Extending Database Technology. ACM, 2012: 38-49.

[18]Aggarwal C C. On the effects of dimensionality reduction on high dimensional similarity search[C]//Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, 2001: 256-266.

[19]Garcia V, Debreuve E, Nielsen F, et al. K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching[C]//Image Processing (ICIP), 2010 17th IEEE International Conference on. IEEE, 2010: 3757-3760.

[20]Qiu D, May S, Nüchter A. GPU-accelerated nearest neighbor search for 3D registration[M]//Computer Vision Systems. Springer Berlin Heidelberg, 2009: 194-203.

[21]Zhuge Y, Cao Y, Miller R W. GPU accelerated fuzzy connected image segmentation by using CUDA[C]//Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE. IEEE, 2009: 6341-6344.

[22]Beyer K, Goldstein J, Ramakrishnan R, et al. When is "nearest neighbor" meaningful?[M]//Database Theory—ICDT'99. Springer Berlin Heidelberg, 1999: 217-235.

[23]Vrbsky S V, Liu J W S. APPROXIMATE-a query processor that produces monotonically improving approximate answers[J]. Knowledge and Data Engineering, IEEE Transactions on, 1993, 5(6): 1056-1068.

[24]Bennett K P, Fayyad U, Geiger D. Density-based indexing for approximate nearest-neighbor queries[C]//Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 1999: 233-243.

[25]Jagadish H V, Ooi B C, Tan K L, et al. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search[J]. ACM Transactions on Database Systems (TODS),

2005, 30(2): 364-397.

[26]Fagin R, Kumar R, Sivakumar D. Efficient similarity search and classification via rank aggregation[C]//Proceedings of the 2003 ACM SIGMOD international conference on Management of data. ACM, 2003: 301-312.

[27]Bern M. Approximate closest-point queries in high dimensions[J]. Information Processing Letters, 1993, 45(2): 95-99.

[28]Arya S, Mount D M, Netanyahu N S, et al. An optimal algorithm for approximate nearest neighbor searching fixed dimensions[J]. Journal of the ACM (JACM), 1998, 45(6): 891-923.

[29]Indyk P, Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality[C]//Proceedings of the thirtieth annual ACM symposium on Theory of computing. ACM, 1998: 604-613.

[30]Gionis A, Indyk P, Motwani R. Similarity search in high dimensions via hashing[C]//VLDB. 1999, 99: 518-529.

[31]Tao Y, Yi K, Sheng C, et al. Quality and efficiency in high dimensional nearest neighbor search[C]//Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. ACM, 2009: 563-576.

[32]Gao J, Jagadish H V, Lu W, et al. DSH: Data sensitive hashing for high-dimensional k-nnsearch[C]//Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 2014: 1127-1138.

[33]Oliveira S R M, Zaiane O R. Privacy preserving Clustering by data transformation[J]. Journal of Information and Data Management, 2010, 1(1): 37.

[34]Liu K, Giannella C, Kargupta H. An attacker's view of distance preserving maps for privacy preserving data mining[M]//Knowledge Discovery in Databases: PKDD 2006. Springer Berlin Heidelberg, 2006: 297-308.

[35]Turgay E O, Pedersen T B, Saygin Y, et al. Disclosure Risks of Distance Preserving Data Transformations[C]//SSDBM. 2008: 79-94.

[36]Wong W K, Cheung D W, Kao B, et al. Secure kNN computation on encrypted databases[C]//Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. ACM, 2009: 139-152.

[37]Bertino E, Ooi B C, Yang Y, et al. Privacy and ownership preserving of outsourced

medical data[C]//Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on. IEEE, 2005: 521-532.

[38]Xiao X, Tao Y. Anatomy: Simple and effective privacy preservation[C]//Proceedings of the 32nd international conference on Very large data bases. VLDB Endowment, 2006: 139-150.

[39]Gruteser M, Grunwald D. Anonymous usage of location-based services through spatial and temporal cloaking[C]//Proceedings of the 1st international conference on Mobile systems, applications and services. ACM, 2003: 31-42.

[40]Gedik B, Liu L. Location privacy in mobile systems: A personalized anonymization model[C]//Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on. IEEE, 2005: 620-629.

[41]Ghinita G, Kalnis P, Skiadopoulos S. PRIVE: anonymous location-based queries in distributed mobile systems[C]//Proceedings of the 16th international conference on World Wide Web. ACM, 2007: 371-380.

[42]Liu A, Zhengy K, Liz L, et al. Efficient secure similarity computation on encrypted trajectory data[C]//Data Engineering (ICDE), 2015 IEEE 31st International Conference on. IEEE, 2015: 66-77.

[43]Chen L, Özsu M T, Oria V. Robust and fast similarity search for moving object trajectories[C]//Proceedings of the 2005 ACM SIGMOD international conference on Management of data. ACM, 2005: 491-502.

[44]Vlachos M, Kollios G, Gunopulos D. Discovering similar multidimensional trajectories[C]//Data Engineering, 2002. Proceedings. 18th International Conference on. IEEE, 2002: 673-684.

[45]Wong W K, Kao B, Cheung D W L, et al. Secure query processing with data interoperability in a cloud database environment[C]//Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 2014: 1395-1406.

[46]Hore B, Mehrotra S, Canim M, et al. Secure multidimensional range queries over outsourced data[J]. The VLDB Journal—The International Journal on Very Large Data Bases, 2012, 21(3): 333-358.

[47]Shaneck M, Kim Y, Kumar V. Privacy preserving nearest neighbor search[M]//Machine Learning in Cyber Trust. Springer US, 2009: 247-276.

[48]Qi Y, Atallah M J. Efficient privacy-preserving k-nearest neighbor search[C]//Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on. IEEE, 2008: 311-319.

[49]Xiong L, Chitti S, Liu L. Preserving data privacy in outsourcing data aggregation services[J]. ACM Transactions on Internet Technology (TOIT), 2007, 7(3): 17.

[50]O'Keeffe M. The Paillier Cryptosystem[J]. A Look Into The Cryptosystem And Its Potential Application, college of New Jersey, 2008.

[51]http://kdd.ics.uci.edu/databases/CorelFeatures/.

[52]http://yann.lecun.com/exdb/mnist.

攻读硕士学位期间发表的论文和取得的科研成果

致　　谢

时光荏苒，岁月如梭，两年多的研究生生涯即将过去。两年多的时光内收获很多，从一名数学专业的学生到一名计算机专业的研究生，中间有许多困难和迷惑。不过幸好周边有可敬的老师与热爱助人的同学的帮助。

在这两年生涯里，我想最先感谢的人是我的导师张志强老师，张老师在平日里对待事情、解决事情的态度对我产生了影响。作为一名数学专业的本科毕业生，在编程方面缺乏锻炼，在计算机基础知识方面缺少学习。所以对待任何一个项目、一场比赛，都会觉得困难。不过幸好有张老师的指点，使我渐渐明白任何一件事情不是没有可能，只要去做慢慢都可解决，迎难而上才能慢慢成长。这里有的不仅是“鱼”，还有更多的“渔”，谢谢张老师。

我要感谢我的父母，我的奶奶。难过时的安慰，迷茫时的指引。他们给予我选择的权利，经济上的帮助，他们让我生长在一个更加自由的环境中。作为子女，现在无以为报，深感愧疚。希望他们健康、幸福。

我要感谢胡江华师兄在比赛中的指导，感谢周永师兄在机器学习学习中的帮助与探讨，感谢孙悦师姐在找工作时的帮助。

还要感谢周边的同学与室友，两年多的相处，增添了许多快乐，大家相互交流，互相学习，共同成长。

最后，感谢所有帮助过我、鼓励过我的人。