PPTD:基于敏感属性泛化和轨迹局部抑制的个性化隐私保护方法

摘　　要

随着 RFID 技术和移动互联网的快速发展，越来越多的轨迹数据被收集到。这些轨迹数据通常可提供用于现实生活中的有用信息，如流量管理、地理营销和基于位置的广告。但是，轨迹数据库可能包含与用户相关的信息，并与敏感属性（如疾病，工作，和收入）相关联。因此，轨迹数据库发布不当会导致用户的隐私处于危险之中。论文的作者提出了 PPTD 算法来解决轨迹数据库中的用户隐私保护问题，同时针对不同用户的隐私保护需求实现了个性化处理。

我们组对论文中的 SAGTD、STR、SAG、MPSTD、MCST 每个算法使用 C 语言进行了代码实现，根据论文的性能分析部分对代码的性能进行了分析，最后编写完成文档。

完成算法需要对论文的内容有深刻的理解。我们首先浏览了论文概要和介绍，简单地了解了论文的背景和研究方向以及论文提出的范例模型；然后深读论文，对论文提出的定理定义进行理解，大致了解伪代码的步骤以及每步都做了什么；之后确认编程语言和开发环境、定义数据结构、实现基础数据类型、编写模块函数、实现算法代码，实现过程中对每个模块都进行单元测试，各个单元组合后进行集成测试；最后对代码的可行性和隐私性进行了分析和总结。

PPTD 通过模拟攻击者可能使用的所有攻击序列，对所有泄露概率大于阈值的行先进行属性泛化，再对概率仍大于阈值的行进行轨迹压缩。PPTD 算法通过严格的数学推理保证了经过处理后的轨迹数据库的每一行的泄露概率都不会大于阈值。我们根据伪代码实现的 C 语言代码运行结果也表明 PPTD 正确而有效。

关键词：轨迹数据；敏感属性；隐私保护；

目　　录

1 绪论

1.1 研究背景与意义

随着 RFID 技术和 GPS 技术的大量应用，轨迹数据越来越多得被收集到，这些轨迹数据通常可提供用于现实生活中的有用信息，如流量管理，地理营销和基于位置的广告。但是，轨迹数据库可能包含关于移动物体的详细信息，并将其与敏感属性（如疾病，工作，和收入）相关联[1]。因此，轨迹数据库的不当发布会导致用户的隐私处于危险之中，特别是当攻击者拥有部分轨迹信息作为背景知识时。现有的隐私保护方法提供了适用于所有用户的同等级的隐私保护，而没有考虑到不同用户的不同隐私保护需求，结果是一些需要高度隐私保护的用户没有被提供足够的隐私保护服务，而另一些则不需要高度的隐私保护的用户得到了过度的保护。在这篇论文中，论文的作者提出 PPTD 算法解决了这个问题，这是一种基于个性化隐私保护的新方法和新概念。它旨在满足用户个人隐私保护需求的前提下，在数据有效性和隐私保护之间寻得平衡。这是将敏感属性泛化方法和部分轨迹去除方法相结合的第一篇实现轨迹数据发布的定制个性化隐私模型的论文。论文的作者对 City80K 和 Metro100K 这两个轨迹数据库进行了算法实验，结果表明，论文的算法是相当富有成效的。和前人研究出的算法相比，论文所提出的 PPTD 算法的效率更高而且效果更好。

举例来说，现有一家医院使用 RFID 标签技术记录了病人的轨迹数据，存储在医院的中央数据库内，现在想要发布这个数据库进行数据挖掘工作，在这个模型下，数据表一般包含以下几部分内容：

（1）唯一标识符（ID）：唯一标识符通常可以标识出唯一的病人身份，在数据表内一般通过整型变量区分。

（2）敏感属性（SA）：敏感属性通常是攻击者不能通过背景知识获取到的数据所有者的信息，攻击者通常希望能够根据背景知识推断出攻击目标的敏感属性，本例的敏感属性是病人所患的病。

（3）轨迹数据（Trajectory）：轨迹数据一系列地点和时间对的集合，模型一般使用小写字母代替地点，整型数字代表时间，比如某病人的轨迹数据是<a2,b3>，就表明此病人时间为 2 时出现在 a 地点，时间为 3 时出现在 b 地点。

（4）显式标识符（EID）：显示标识符一般是能标识出病人个体的个人身份信息，如身份证号，姓名。

一般来说，在数据库发布之前，医院都会将显式标识符即病人的身份证号，姓名，性别去除，在攻击者完全没有背景知识的情况下，这样做是安全的。但是如果攻击者拥

有被攻击者的背景知识，被攻击者的个人敏感属性就很有可能被泄露。比如说，攻击者现在知道 Bob 在时间为 3 时去过地点 a，时间为 4 时去过地点 b，攻击者就可以根据<a3,b4>这个轨迹序列去匹配轨迹数据库中的行，进而就有可能得知 Bob 所患的病。论文提出的 PPTD 算法就是为了在攻击者拥有背景知识的前提下，将数据表内的每行数据的隐私泄露概率都降到阈值以下，从而达到保护用户隐私的目的。

1.2 国内外研究现状

在基于位置的个人隐私保护研究领域，大多数现有的工作都是在基于位置的服务（LBS）的背景下进行的[2]。可信服务器负责处理传入请求并将其传递给可用的服务提供商。因此，这些工作的主要目标是尽快提供服务，而对移动目标的匿名保护工作不足。一般来说，LBS 相关隐私有两种主要类型：位置隐私和查询隐私。位置隐私旨在保护移动目标可能从其位置泄露的个人信息；查询隐私旨在保护移动对象的私有的可能会从其查询中泄露的信息。一种最流行的用来进行位置隐私和查询隐私保护的方法是 k 匿名。k 匿名第一次在关系型数据库社区被介绍，是一种对关系型数据进行匿名处理的隐私保护方法，它随后在社区成员中流行了起来。需要提到的是，一个关系型数据库满足 k 匿名当且仅当在一个准标识符集合前提下每一行数据无法从 k-1 行数据行中被识别出来。在 LBS 的背景下，一个移动对象被认为是 k 匿名的当且仅当它的位置信息被送到一个服务提供商时，它无法从同时在同一个区域内的至少 k-1 个其他移动对象中被识别出来。

就如上面所提到的，隐私已经成为了一个在 LBS 领域的主要的研究课题。但是，轨迹数据并不是 LBS 隐私研究的重点。原因是在轨迹数据概念里，匿名保护是离线的而且是以数据为中心的，而在 LBS 的概念里，匿名保护是在线的并且是以服务为中心的。在这篇论文里，我们聚焦于轨迹数据库的概念，我们有一个静态的存储了移动节点的轨迹数据以及与之关联的敏感信息，如患病情况，工作，收入。我们的目标是在整个轨迹数据库匿名化的过程中，在保护用户隐私的同时尽可能地保证数据有效性。

现在，有三类主要的针对轨迹数据发布的隐私保护方法：

（1）基于群的在关系型数据库中实现 k 匿名的方法。

（2）假设攻击者使用部分轨迹信息识别剩余的移动节点或敏感信息的准标识符方法。

（3）保证移动对象在差分隐私定义下被保护的差分隐私方法。

差分隐私被定义成一种请求响应机制的性质，数据库被可信中心拥有，可信中心通过差分隐私保护方式进行请求响应。它确保了请求的回答在大体上不会被别的在数据库中的个人数据的缺失或出现而影响到。最近，一些方法提出了对轨迹数据进行差分隐私

保护的方法。这些方法的目标是发布能够有效支持特定的数据分析的噪声数据，比如说统计请求响应此书和常见连续形式的挖掘。但在另一方面，发布带有差分隐私保护的轨迹数据可能无法提供有意义的数据效用。这是因为差分隐私中噪声的不可确定性。总的来说，差分隐私方法经常给发布轨迹数据的机制强加了一个保证，这超出了本文讨论的范围。

1.3 研究内容及主要工作

Nergiz 等人[3]对轨迹数据采用了 k 匿名的概念并且提出了一种基于群的轨迹数据匿名方法。他们显示了发布隐去姓名资料的轨迹数据可能会导致隐私泄露，因此提出了用于发布匿名轨迹数据的一个随机分布的重构算法。 Monreale 等人[4]展示了一种结合了空间泛化概念和 k 匿名的轨迹数据匿名方法。主要的思想是使用模糊化的位置信息数据来代替更精确的位置信息。

Mahdavifar 等人[5]提出了一个贪婪的基于群的方法，在这种方法中，轨迹的匿名性在某种程度上与移动对象的隐私要求成比例。尽管这种方法目标是为了在轨迹数据发布时保护个人隐私，但是它并不能对抗属性关联攻击和相似攻击。

Domingo-Ferrer 和 Trujillo-Rasua[6]展示了两个轨迹匿名方法，叫做 SwapLocations 和 ReachLocations，这两种方法就不包含任何干扰或广义的位置方面来说都保护了原始位置。SwapLocations 基于轨迹的微聚集和位置的排列，它保证了轨迹数据的 k 匿名，但是没有考虑到可达性约束。这有可能导致在匿名发布的轨迹中一些连续不断的位置在现实世界里是不可达的，这导致攻击者很容易通过公开可获得的地图信息辨识出哪些是虚假的位置信息。而且，ReachLocations 仅仅基于位置的排列并且致力于将可达性约束考虑在内。它仅仅保证了位置信息的 k 多样化而不是轨迹数据的 k 匿名。原因是在强制可达性约束的同时提供 k 匿名会导致许多原始位置被删除掉。

Terrovitis 等人[7]假设攻击者使用了一些局部轨迹信息作为其背景知识去推断他们不知道的移动点。因此，他们反复地从原始轨迹中删去被选择的移动点直到隐私约束被满足。Yarovoy 等人[8]通过定义一种和原始轨迹数据关联在一起的攻击图，以及它扭曲的一个，引入了一种 k 匿名的概念，他们将时间戳视为一种准标识符，并且展示了两种不同的算法，即 extreme-union 和 symmetric-anonymization，来创建可证的满足 k 匿名要求得匿名群组。

Mohammed 等人[9]采用了一种叫做 LKC 隐私的隐私模型并且开发了一种利用全球抑制来实现 LKC 隐私的匿名化框架。大致的思想是确保每一个在轨迹数据库中的最大长度为 L 的子轨迹被至少 K 个轨迹数据记录所分享并且推断出任何敏感属性的几率都

不大于 C.chen 等人提出的一个提供局部和全局轨迹压缩的相似框架。他们的目标是在轨迹数据库中保存移动点的实例和在轨迹数据库中的频繁的子轨迹。

Ghasemzadeh 等人[10]提出了一个在轨迹数据库中达到匿名化的简单的方法，这种方法在保护信息以支持有效的乘客流量分析的同时，只能抵御标识关联攻击。他们首先生成了轨迹数据库的概率流程图，然后在某种程度上匿名化了数据库让 LK 隐私被满足，并且所有的对流图的影响降到了最低。

前面提到的大部分的方法都没有考虑移动对象的不同的隐私保护需求，而且，他们中的大多数不能同时抵御所有的三种身份链接攻击、属性链接攻击和相似攻击。结果是导致对某些对象而言隐私保护没有达到需求，对另外一些对象而言则实施了过度的隐私保护，导致既增加了不必要的信息丢失，有提高了隐私泄露的风险。在这篇论文里，作者着重强调了个性化隐私保护的概念，因此一些移动对象可以指定对敏感属性的隐私保护的不同程度。更具体的说，我们承认了新兴的轨迹数据发布场景，其中移动对象有不同级别的隐私保护，并且需要发布带有敏感属性的轨迹数据库。这自然需要同时防御身份链接攻击、属性链接攻击和相似攻击这三种攻击。

1.4 本文组织结构

本篇文章主要分为六个部分：

第一部分为绪论部分，对论文的研究背景和研究意义做出介绍，探讨了论文涉及问题的国内外研究现状，并且根据目前已有技术的问题，结合前人已有技术提出了 PPTD 算法的构思，然后对论文组织结构进行说明。

第二部分为论文算法的相关知识，给出与算法相关的定理定义。

第三部分为攻击模型介绍，该部分对存在的隐私泄露情况与对应的攻击模型做出介绍。

第四部分为论文所提出的五种算法的详细介绍，首先介绍算法的整体思想，然后用图和文字的方式介绍算法的具体步骤。

第五部分为用代码关键部分的截图和文字阐释实现算法代码的具体步骤和思想。

第六部分为使用代码对算法的仿真实验结果，以及根据实验结果对算法有效性和安全性的的分析。

2 相关概念介绍

论文从基础的轨迹概念开始介绍，介绍了在算法中可能涉及到的所有概念，论文在介绍这些概念时，提出了一个例子。图 2.1 展示了例子中用到的敏感属性树的结构和形状，这颗树有 19 个叶子节点，每一个节点都对应表 2.1-2.3 中出现的敏感属性一列中的值。表 2.1-2.3 展示了一张数据表以及使用了多种匿名算法后的结果数据表。其中，表 2.1 是用到的数据表的原始状态，其中的每一列的含义为{ID，隐私需求水平，轨迹数据，敏感信息}，表 2.2 是 SAGTD 算法泛化处理后的结果数据表，其中敏感信息一列的值发生了改变，其值在敏感属性树的高度上升了。表 2.3 是 MPSTD 算法对 SA 泛化结果的数据表进行轨迹压制处理后的数据表，其轨迹对应的列发生改变，其中某些点被删去了。

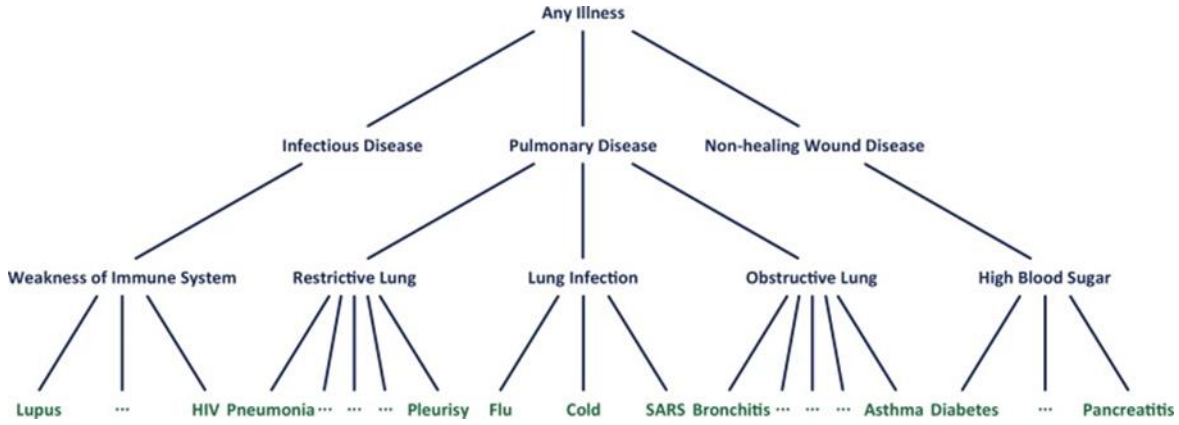

图 2.1 敏感属性树

表 2.1 原始轨迹数据表

ID	Privacy level	Trajectory	Disease
1	Low	⟨b2, d3, c4, f6, a7, e8⟩	HIV
2	Medium	⟨c4, f6, a7, e9⟩	SARS
3	Low	⟨d3, c4, a7⟩	Pancreatitis
4	High	⟨b2, f6, a7, e8⟩	HIV
5	Medium	⟨d5, f6, e9⟩	Flu
6	Low	⟨c4, d5, f6⟩	Diabetes
7	No Privacy	⟨b2, f6, e9⟩	Cold

表 2.2　SAGTD 算法泛化后数据表

ID	Privacy level	Trajectory	Disease
1	Low	⟨b2, d3, c4, f6, a7, e8⟩	Weakness of Immune
2	Medium	⟨c4, f6, a7, e9⟩	Pulmonary Disease
3	Low	⟨d3, c4, a7⟩	Pancreatitis
4	High	⟨b2, f6, a7, e8⟩	Any Illness
5	Medium	⟨d5, f6, e9⟩	Pulmonary Disease
6	Low	⟨c4, d5, f6⟩	High Blood Sugar
7	No Privacy	⟨b2, f6, e9⟩	Cold

表 2.3　MPSTD 算法轨迹压制后的数据表

ID	Privacy level	Trajectory	Disease
1	Low	⟨b2, d3, c4, f6, a7, e8⟩	Weakness of Immune
2	Medium	⟨c4, f6, a7, e9⟩	Pulmonary Disease
3	Low	⟨d3, c4, a7⟩	Pancreatitis
4	High	⟨f6, a7⟩	Any Illness
5	Medium	⟨d5, f6, e9⟩	Pulmonary Disease
6	Low	⟨c4, d5, f6⟩	High Blood Sugar
7	No Privacy	⟨b2, f6, e9⟩	Cold

图 2.1 中敏感属性树的叶子节点有 19 个，其中 Weakness of Immune System 的叶子节点有三个，Restrictive Lung 的叶子节点有五个，Obstructive Lung 的叶子节点有五个，High Blood Sugar 的叶子节点有三个，这在构建敏感属性树的时候非常重要，这会影响到隐私泄露概率的计算。

表 2.2 中使用 SAGTD 算法的最大泛化深度为 2，隐私泄露概率阈值为 0.5，攻击序列最大长度为 2。

表 2.3 中使用 MPSTD 算法时的攻击序列最大长度为 2，隐私泄露概率阈值为 0.5。

2.1 轨迹

若O为所有用户的集合，那么T_i就表示$o_i \in O$这个用户的轨迹，轨迹是一系列时空对的集合，表示如公式（2.1）。

$$\tau_i = \langle (l_i^1, t_i^1), (l_i^2, t_i^2), \dots, (l_i^m, t_i^m) \rangle \quad (2.1)$$

其中每一个$(l_i^k, t_i^k) \in \tau_i$被叫做移动点并被标记为$p_i^k$。

T_i的长度通过$|T_i|$标记。

τ_i^k表示T_i中所有小于 k 的轨迹的集合。

两个移动点的顺序关系通过<表示，对于两个移动点$p_i^k = (l_i^k, t_i^k)$，$p_i^{k'} = (l_i^{k'}, t_i^{k'})$，$p_i^k \prec p_i^{k'} \text{iff} t_i^k < t_i^{k'}$.即先发生的排在前面，后发生的排在后面。

2.2 轨迹的联结

两轨迹$\tau_i = \langle (l_i^1, t_i^1), (l_i^2, t_i^2), \dots, (l_i^m, t_i^m) \rangle$ 和$\tau_j = \langle (l_j^1, t_j^1), (l_j^2, t_j^2), \dots, (l_j^m, t_j^m) \rangle$ 被称为可联结的当且仅当 $\tau_i^{m-1} = \tau_j^{m-1}$并且 $t_i^m < t_j^m$。

两轨迹之间的联结表示如公式（2.2）。

$$\tau_i \bowtie \tau_j = \langle (l_i^1, t_i^1), (l_i^2, t_i^2), \dots, (l_i^m, t_i^m), (l_j^m, t_j^m) \rangle \quad (2.2)$$

2.3 子轨迹

现有$\tau_i = \langle p_i^1, p_i^2, \dots, p_i^m \rangle$ 和$\tau_j = \langle p_j^1, p_j^2, \dots, p_j^s \rangle$ 两个轨迹，如果两轨迹满足公式（2.3）和公式（2.4）。

$$p_j^1 = p_i^{k_1}, p_j^2 = p_i^{k_2}, \dots, p_j^s = p_i^{k_s}. \quad (2.3)$$

$$1 < K_1 < .. < K_s \leq m \quad (2.4)$$

那么T_j就被叫做T_i的子轨迹，并被标记为$T_j \in T_i$。

轨迹数据库中r_i行表示为公式（2.5）。

$$r_i = \langle p_i^1, p_i^2, \dots, p_i^m \rangle : s_i^1, s_i^2, \dots, s_i^n : a_i^1, a_i^2, \dots, a_i^q \quad (2.5)$$

r_i行的轨迹集表示为公式（2.6）。

$$\tau(r_i) = \langle p_i^1, p_i^2, \dots, p_i^m \rangle . \quad (2.6)$$

2.4 敏感属性树

论文将待处理的数据表中的所有敏感属性的可能取值以及其各种泛化后的可能的属性值聚集在一起构建成一棵属性树，范例所使用的敏感属性树如图 2.1 所示，共有 19 个叶子结点，值得注意的是 Weakness of Immune System 下有三个叶子节点，Restrictive Lung 下有五个叶子节点，Obstructive Lung 下有五个叶子节点，High Blood Sugar 下有三个叶子节点，读者自行实现代码时构建出来的树必须和上图中一致，否则会因为泄露概率的计算结果不同导致算法结果不同。

$\ell(V_i)$表示某节点的叶子结点集合，$|\ell(V_i)|$被称为其长度，举例来说。

$$\ell(LungInfection) = \{Flu,Cold,SARS\} \tag{2.7}$$

$$|\ell(LungInfection)| = 3 \tag{2.8}$$

公式（2.7）表明 Lung Infection 在树中对应的节点的所有的叶子节点结合所包含的元素有 Flu，Cold，SARS。公式（2.8）中表明此集合的长度为 3。注意每个叶子节点的叶子结点集合的长度都为 1，且包含元素有且仅有叶子节点本身。

2.5 祖先节点

$c(v_i)$表示在属性树内某节点的所有祖先结点的集合，这里和二叉树的祖先结点的概念基本相同。

2.6 守护节点

守护节点是根据数据表中用户的不同的隐私保护需求计算出来的，表 2.1 中是原始数据表，其中隐私保护需求 Privacy Level 一列有四种不同的取值，分别是 Low，Medium，High 以及 No Privacy，其中 Low 的取值用整数 0 代替，Medium 的取值使用整数 1 代替，High 的取值使用整数 2 代替，No Privacy 的行不做处理，这样就可以计算出每一行的守护节点，某行的守护节点v_j需要满足公式（2.9）。

$$\iota(v_i) = \theta(\rho(r_i)) \tag{2.9}$$

公式（2.9）中表示r_i对应的用户。输入某用户对象r_i，θ函数返回用户的隐私保护对应的整数，如表 2.1 中$\theta(r_1)$返回的值为 0，因为第一行对应的隐私需求水平是 0。$\iota(v_i)$表示某结点的高度，所有叶子结点的高度都是 0，树中根节点 Any Illness 的高度是 3。

例如，表 2.1 中 ID 为 1 的行对应的守护节点是 HIV，因为此行的隐私保护需求水平整数为 0，守护节点的高度也是 0，因此守护节点为其敏感属性对应的节点本身。ID

为 2 的行对应的守护节点是 Lung Infection，因为此行的隐私保护需求水平整数为 1，因此此行的守护节点的高度也为 1，SARS 节点向上数 1，对应节点为 Lung Infection。

某行的守护结点通过$g(r_i)$表示。一行数据的守护节点表明在最大长度和阈值的限制下，无论攻击者拥有何种背景知识，攻击者能够得知此行对应的敏感信息至少不能是守护节点对应的值，比如表 2.1 中 ID 为 2 的行的守护节点为 Lung Infection，这表明在经过算法处理后，攻击者最多只能推断出此病人所患病为 Pulmonary Disease，而不能推断出是 Lung Infection 或者更为细致的病情。

需要注意的是，无论是在泛化过程中还是泛化后，各行的守护节点是不变的，一行的守护节点仅根据隐私保护需求而确定，即在数据表处理前就已经确定了。

2.7 孪生节点

无论是在泛化前还是泛化后，孪生节点函数$\eta(r_k)$返回的节点都是此时行r_k中敏感属性对应的树中的节点。

2.8 攻击者背景知识

攻击者在针对某对象进行攻击时，如果预先知道此对象在某些时刻去过的某些地点，即预先得知用户的某些轨迹时空对，攻击者就能通过标识符链接攻击、属性链接攻击、相似攻击三种攻击方式推断匹配得到被攻击者的敏感信息，正因为这三种攻击的存在，即使在发布的数据库中去除了用户的姓名和身份证号等标识出个人信息的标识符，用户的个人隐私依然有这样的风险。论文将攻击者预先得知的背景知识用公式（2.10）表示。

$$\xi_i = \langle p_i^1, p_i^2, \dots, p_i^l \rangle, l \le \delta \quad (2.10)$$

其中δ是预先设定好的此攻击序列所含移动节点元素的最大数目，攻击者根据这个攻击序列所匹配出的数据表用公式（2.11）表示。

$$T(\xi_i) = \{r_k \in T \mid \xi_i \sqsubseteq \tau(r_k)\}. \quad (2.11)$$

例如如果当前攻击者的序列为<b2>，在表 2.1 中，攻击者能匹配出来的行就是r_1，r_4，r_7，即$T(\xi_i)$包含r_1，r_4，r_7。

2.9 泛化深度

表 2.2 是经过了 SA 泛化后数据表结果，和表 2.1 中的数据表相比，Disease 敏感属性一列的值发生了变化，各行的值发生了不同程度的泛化，第一行 HIV 泛化为 Weakness of Immune，第二行 SARS 泛化为 Pulmonary Disease，第四行 HIV 泛化为 Any Illness，第五行 Flu 泛化为 Pulmonary Disease，第六行 Diabetes 泛化为 High Blood Sugar。

将这些行泛化前的敏感属性以及泛化后敏感属性与图 2.1 中的敏感属性树对应，可以看到这些行对应节点高度向上增加的最大值为 2，高度增加最大的是第四行，HIV 对应高度为 0，Any Illness 对应高度为 2，差值为 2。

论文定义泛化深度为公式（2.12）。

$$\zeta(r_i) = \begin{cases} \iota(\eta(r_i)) - \iota(g(r_i)) & \iota(\eta(r_i)) > \iota(g(r_i)), \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

最大泛化深度表示为ζ_{max}，它表明某行泛化结果的孪生节点和此行的守护节点的高度差不能超过ζ_{max}。

2.10 泄露概率

给定一个轨迹数据表，攻击者的背景知识ξ_i，攻击者通过背景知识匹配到$T(\xi_i)$，此时就可以计算$T(\xi_i)$中某行的泄露概率，论文定义待求值得r_i相对于匹配到的数据库中的某行r_k的泄露概率为公式（2.13）。

$$P(\ell(g(r_i))|\ell(\eta(r_k))) = \frac{|\ell(g(r_i)) \cap \ell(\eta(r_k))|}{|\ell(\eta(r_k))|}. \tag{2.13}$$

公式（2.13）中的值有三种可能的结果，表示为公式（2.14）。

$$P(\ell(g(r_i))|\ell(\eta(r_k))) = \begin{cases} 1 & g(r_i) \in c(\eta(r_k)) \vee g(r_i) = \eta(r_k), \\ \alpha & \eta(r_k) \in c(g(r_i)), \\ 0 & \text{otherwise,} \end{cases} \tag{2.14}$$

第一种情况是当待求值行r_i的守护节点被r_k所覆盖或者r_i的守护节点等于r_k的孪生节点时，这表明公式（2.13）其分子与分母相等，故结果为 1。

第二种情况是公式（2.13）中的分子小于分母，结果在 0 到 1 之间。

第三种情况是r_i和r_k没有交集，分子为 0。

计算$T(\xi_i)$中某行r_i的隐私泄露概率为公式（2.15）。

$$P_b(\rho(r_i)|\xi_i) = \frac{1}{|T(\xi_i)|} \sum_{r_k \in T(\xi_i)} P(\ell(g(r_i))|\ell(\eta(r_k))) \tag{2.15}$$

即计算r_i的隐私泄露概率，将r_i对应于r_k的每个隐私泄露概率都计算出来，加在一起后除于$T(\xi_i)$的长度。

例如，给定表 2.2 中的数据表，攻击序列为$\xi_i =$〈$a7$〉，因此匹配出的$T(\xi_i) = \{r_1, r_2, r_3, r_4\}$，我们知道$g(r_3) =$ Pancreatitis，$\eta(r_1) =$ WeaknessofImmune，$\eta(r_2) =$ PulmonaryDisease，$\eta(r_3) =$ Pancreatitis，$\eta(r_4) =$ AnyIllness，这样根据公式（2.13）和公式（2.14）就可以计算出如下结果：

$$\begin{aligned} P(\ell(g(r_3))|\ell(\eta(r_1))) &= P(\ell(g(r_3))|\ell(\eta(r_2))) = 0, \\ P(\ell(g(r_3))|\ell(\eta(r_3))) &= 1, \\ P(\ell(g(r_3))|\ell(\eta(r_4))) &= 0.05. \end{aligned}$$

进而，有在攻击背景为$\xi_i =$〈$a7$〉的情况下，r_3的隐私泄露水平为$P_b(\rho(r_i)|\xi_i) = \frac{1}{4}(0+0+1+0.05) = 0.26$，注意在本文所举的范例里图 2.1 中的敏感属性树的叶子节点是 19 个。

2.11 局部轨迹压缩

数据表在经过 SA 泛化后，因为某些移动节点用户的个人隐私仍然有可能泄露。因此，我们应该从轨迹中删去一定数目的移动点， 这样任何移动对象在任何可能的攻击下隐私泄露的风险都不会大于阈值。为了这个目的，论文对数据表可以对轨迹数据采用一种局部的或者全局的轨迹压缩的方法。

让T为轨迹数据表，全局压缩会从所有的轨迹数据记录内删去那些让数据表内的移动对象的隐私泄露概率非常高的移动点，而局部轨迹压缩只会从那些泄露风险较高的行删去移动点而对于其他行则原封不动，因此，局部轨迹压缩比全局轨迹压缩而言保留了更多的数据有效性，于是，论文将局部轨迹压缩其应用于轨迹数据处理的算法。

2.12 危险子轨迹

给定一个轨迹数据库T，一个非空的子轨迹τ_i被称为是危险的，当且仅当有一行$r_i \in T$有$P_b(\rho(r_i)|\tau_i) > \sigma$，其中$\sigma$是隐私泄露概率阈值。这里$r_i$被称为危险记录。

2.13 个性化压缩分数

给定一个轨迹数据库T，$\mathcal{T}_c$是危险轨迹数据的集合，$\tau_j \in \mathcal{T}_c$被称为危险子轨迹。对于一个移动点$p_j^k \in \tau_j$来说在$\mathcal{T}_c$内的个性化压缩分数使用$\varphi(p_j^k, \tau_j, \mathcal{T}_c)$标记并通过公式（2.16）计算

$$\varphi(p_j^k, \tau_j, \mathcal{T}_c) = \frac{|\mathcal{T}_c(p_j^k)|}{|T(\tau_j)|} \sum_{r_i \in T(\tau_j)} \theta(\rho(r_i)) \qquad (2.16)$$

其中$\mathcal{T}_c(p_j^k)$是$\mathcal{T}_c$中根据p_j^k匹配到的所有的轨迹。同样的，一个轨迹τ_j的最大个性化压缩分数$\psi(\tau_j, \mathcal{T}_c)$,通过公式（2.17）计算。

$$\psi(\tau_j, \mathcal{T}_c) = \max_{p_j^k \in \tau_j} \varphi(p_j^k, \tau_j, \mathcal{T}_c). \qquad (2.17)$$

即这个轨迹内的各个点中最大的个性化压缩分数。

3 攻击模型

在本章节中，将会介绍轨迹数据发布的研究中存在的三种攻击模型：身份链接攻击，属性链接攻击，相似攻击。

在隐私保护数据发布的研究中，一般假设攻击者掌握着一定的背景知识和能力。比如攻击者知道 Bob 在表 3.1 中，并且知道她在时间为 2 时分别访问了地点 b，这些背景知识都是有可能存在的，也是隐私保护算法的设计者需要考虑到的。

表 3.1 原始数据库

ID	Trajectory	Disease
1	<*b*2, *d*3, *c*4, *f*6, *a*7, *e*8>	HIV
2	<*c*4, *f*6, *a*7, *e*9>	SARS
3	<*d*3, *c*4, *a*7>	Pancreatitis
4	<*b*2, *f*6, *a*7, *e*8>	HIV
5	<*d*5, *f*6, *e*9>	Flu
6	<*c*4, *d*5, *f*6>	Diabetes
7	<*b*2, *f*6, *e*9>	Cold

3.1 身份链接攻击

如果一个轨迹在轨迹数据库内非常特殊的话，那么就没有那么多移动对象可以匹配到，攻击者使用一些背景知识就可能会精确地辨识出被攻击者，以及被攻击者的敏感信息。比如说，假设攻击者知道在表 3.1 中有一条 Alice 的数据并且她在时间为 4 和时间为 5 的时候分别访问了地点 c 和地点 d。攻击者就可以将r_6的数据关联到 Alice，并且辨识出她的病是 Diabetes，因为r_6是唯一包含了子轨迹<c4,d5>的行。这就是身份链接攻击。

3.2 属性链接攻击

如果在一些攻击序列下一个敏感属性频繁的出现，那么即使攻击者不能完全确认被攻击者属于哪一行，攻击者依然能够唯一地辨识出被攻击者的敏感信息。比如说，假设攻击者知道 Bob 在时间为 6 和时间为 7 时分别访问了地点 f 和地点 a，那么攻击者就可

以推断出 Bob 患有 HIV 的概率是$2/3 = 0.67$，因为在包含轨迹<f6,a6>的r_1，r_2和r_4三行数据里，有两行数据的敏感信息都是 HIV。这就是属性链接攻击

3.3 相似攻击

如果在某些攻击序列下匹配到的敏感属性虽然各自不同但是在语义上是非常相近的话，攻击者可能会推断出来一些敏感信息尽管不能确认唯一确认被攻击者所对应的行。比如说，假设攻击者知道 Carol 在时间为 6 和 9 时分别访问了地点 f 和地点 e，那么，攻击者就能知道 Carol 百分之百患了 Lung Infection，因为匹配到包含<f6,e9>的r_2，r_5，r_7对应的敏感信息分别是 SARS，Flu，Cold，这些都属与 Lung Infection 的范畴。这就是针对轨迹数据的相似攻击。

4 PPTD

为了弥补现有算法存在的缺陷，本算法主要解决了以下几个问题：（1）处理后的数据能够抵抗身份链接攻击、属性链接攻击、相似攻击三种隐私攻击方式；（2）与处理前的数据相比，尽可能的减少数据可用性的损失；（3）与现有方法相比，算法需要有更高的效率。

4.1 算法总体设计

算法包含两个步骤：属性泛化和局部轨迹压缩，分别对应 SAGTD 算法和 MPSTD 算法。其中 SAGTD 算法包含 STR 算法和 SAG 算法，在最大泛化深度和最大攻击序列长度的限制下，SAGTD 对超出了泄露概率的行的敏感属性尽可能的进行泛化，其中 STR 模拟攻击者所有可能的攻击序列，SAG 算法对超出泄露概率的行进行属性泛化。MPSTD 处理那些即使进行了属性泛化，其隐私泄露概率仍大于阈值的行，MPSTD 算法找出危险轨迹中移动点的压缩分数最大的移动点，然后将其从危险行中删除，由于删除某行的移动节点可能会导致其他行的隐私泄露概率上升，所以需要使用 MCST 算法重新辨识出危险轨迹，MCST 算法负责将在数据表改变后重新计算危险轨迹，以防止表内某行隐私泄露概率大于阈值。

4.2 SAGTD 算法

SAGTD 算法包含 STR 算法和 SAG 算法，输入一个轨迹数据表，返回一个属性经过泛化的轨迹数据表。SAGTD 首先使用 STR 算法穷举得到攻击者所有的攻击序列，对每一种不同的攻击序列，每次计算泄露概率最大的行的隐私泄露概率，因为只要这样的行的隐私泄露水平降到了阈值以下，那么那些守护节点在其下面的行的隐私泄露水平自然也就降到了阈值以下（定理 1）。对于筛选出来的行，使用 SAG 算法进行泛化，泛化完成后替换掉原来的行。这样保证在每一种可能的攻击下，在不大于最大泛化深度的情况下，泛化后的行的泄露概率比泄露概率阈值要小。

4.2.1 算法主体

$input$:
T: Trajectory database
$output$:
τ^s: SA‐generalized trajectory database
1: $\mathcal{A}^\delta := STR(T)$

2: for each $\tau_j \in \mathcal{A}^\delta$ do

3: $B_f := \{r_k \in T(\tau_j) | \ell(g(r_k)) \not\subset \ell(g(r_j))$ for all $r_j \in T(\tau_j)|$

4: $C_f := |r_k \in B_j | P_b(\rho(r_k)|\tau_j) > \sigma|$

5: if $c_j \neq \emptyset$ then

6: $T := (T - C_j) \cup SAG(\tau_j, C_j)$

7: end if

8: end for

9: $T^S := T$

10: return T^S

上面是 SAGTD 算法的伪代码，其具体每行的意义如下：

1：运行 STR 算法根据原轨迹数据库穷举出攻击者可能预先知道的攻击序列，记为$\mathcal{A}^\delta$。

2-3：对于每一种可能的攻击背景，计算出在此攻击背景下泄露概率最大的行（根据定理 1），这些行的集合记为B_j。

4：计算出B_j中泄露概率大于泄露概率阈值的行，这些行的集合记为C_j。

5-7：如果C_j不为空，那么先从 T 中去除C_j中的行，再将C_j中的每一行进行 SAG 泛化后加入到结果集 T 中。

9-10：此时结果数据表 T 就是 SAGTD 泛化后的数据表。

4.2.2 STR 算法

STR 算法负责在最大攻击序列长度的限制下，模拟攻击者可能使用的每一种攻击序列，并返回结果集。长度为 1 的攻击序列是原始数据表内的每一个轨迹移动点的集合，长度为 2 的攻击序列则是长度为 1 的序列集合内元素的联结，只要是满足轨迹联结条件的，都并入到结果集合内，依此类推，直到攻击序列的长度大于限制或者没有满足轨迹联结条件的轨迹了，那么就返回结果集合供 SAGTD 算法主体处理。

1: $A^\delta := \emptyset$

2: $\mathcal{A}_1 := \{\tau_j | \tau_j \in \tau(r_i)$ for some $r_i \in T \wedge |\tau_J| = 1\}$

3: for each $\tau_j \in \mathcal{A}_1$ do

4: Compute $T(\tau_j)$ using

5: $\mathcal{A}^\delta := \mathcal{A}^\delta \cup \{\tau_j\}$

6: end for

7: $i := 1$

8: while $i \le \delta$ and $\mathcal{A}_i \neq \emptyset$ do
9: $\mathcal{A}_{i+1} := \emptyset$
10: for $j := 1$ to $|\mathcal{A}_1|$ do
11: for k $:= j+1$ to $|A_i|$ do
12: if $\tau_j^{1..-1} = \tau_k^{i-1}$ and $T(\tau_j) \cap T(\tau_k) \neq \emptyset$ then
13: $T(\tau_j \bowtie \tau_k) := T(\tau_j) \cap T(\tau_k)$
14: $\mathcal{A}_{i+1} := \mathcal{A}_{1+1} \cup \{\tau_j \bowtie \tau_k\}$
15: $\mathcal{A}^{\delta} := \mathcal{A}^{\delta} \cup \{\tau_j \bowtie \tau_k\}$
16: end if
17: end for
18: end for
19: $i := i+1$
20: end while
21: return $\mathcal{A}^{\delta}$

STR 算法在最大长度小于 δ 的前提下计算出攻击者可能知道的所有攻击序列组合。上面是 STR 算法的伪代码，其具体每行的意义如下：

1-6：将输入的轨迹数据库中的每一行轨迹的每个移动点看作一条轨迹，$\mathcal{A}^{\delta}$和A_1不断取并集。

7-19：当 i 小于等于背景攻击序列的最大长度并且得到的新的攻击序列不为空时，对于A_i中的任意两条轨迹进行组合，如果这两条轨迹可联结(第 12 行)，那么就加入到$\mathcal{A}^{\delta}$和A_{i+1}中。

21：返回所有可能的攻击序列组合结果$\mathcal{A}^{\delta}$。

4.2.3 SAG 算法

SAG 算法负责对隐私泄露概率大于阈值的行的敏感属性进行泛化处理，算法接收一条背景攻击轨迹序列τ_j和危险轨迹数据行集合C_j作为输入，在τ_j这条攻击序列下，对危险轨迹数据库内的行的属性进行泛化，其泛化的深度不大于用户设定的最大泛化深度，如果在达到最大泛化深度前隐私泄露概率降到阈值以下，那么此行的泛化处理结果是正确的，可以保证泄露概率，如果达到了最大泛化深度，那么此行的泛化处理结果的泄露概率是有可能没有达到阈值以下的，这时就需要 MPSTD 算法进行局部轨迹压缩的后续处理。

1: $S_j := \emptyset$
2: for each $r_i \in C_j$ do

3: if $(\eta(r_i)) \subset \ell(g(r_i))$ or $(\eta(r_1)) = \ell(g(r_i))$ then
4: $\Omega(r_i) := l(p(g(r_i)))$
5: $if\ P_b(\rho(r_i)|\tau_j) \leq \sigma$ then
6: $D_j := \{r_{/\langle} \in C_j | \ell(g(r_{/\langle})) = \ell(g(r_1 \cdot))\}$
7: $C_j := C_j - D_j$
8: $S_j := S_j \cup D_j$
9: end if
10: end if
11: end for
12: while $C_j \neq \emptyset$ do
13: for each $\in C$, do
14: if $\iota(n(r_i)) - \iota(g(r_i)) \geq \zeta_{max}$ or $\iota(n(r_i)) = h$ then
15: $C_j := C_j - \{r_i\}$
16: $S_j := S_j - \{r_i\}$
17: else if $P_b(\rho\ ()|\tau \leq \sigma$ then
18: $D := \{r_{\mathrm{k}} \in C_j\ | l(g(r_k)) = l(g(r_i))]$
19: $C_j := C_j - D_j$
20: $S_j := S_j \cup D_j$
21 else
21: $\Omega(r_i) := l(p(n(r_1)))$
21: end if
24: end for
21: end while
26: return S_j

上面是 SAG 算法的伪代码，其具体每行的意义如下：

1-11：对于$\mathrm{C_j}$中的每一行，如果此行的泄露概率一定等于 1，那么就将这一行的属性向上泛化一层，如果泛化后泄露概率小于阈值，那么就将此行从$\mathrm{C_j}$中删除，并将此行加入到结果集$\mathrm{S_j}$内。

12-25：当$\mathrm{C_j}$不为空时，一直对$\mathrm{C_j}$中的每一行进行判断，先判断当前泛化深度是否大于等于最大泛化深度或者泛化属性已经是根节点，如果是，那么就从$\mathrm{C_j}$中删除此行并加入到结果集$\mathrm{S_j}$中，再判断泄露概率是否已经小于阈值，如果是，那么从$\mathrm{C_j}$中删除此行并加入到结果集$\mathrm{S_j}$中，否则就泛化一次属性值。

26：将输入的C_j的泛化结果S_j返回。

4.3 MPSTD 算法

由于属性泛化方法不能保证泛化后每一行的泄露概率都在阈值以下，所以需要进行局部轨迹压缩来达到这一目的，这也就是 MPSTD 算法的目的，通过计算压缩分数来找到最优压缩点和对应的最优压缩行，然后将最优压缩行中的最优压缩点除去，由于删除移动节点会静默改变其它行的泄露概率，因此需要对危险轨迹集合进行二次计算。

4.3.1 算法主体

1: $\mathcal{T}_c := \emptyset$
2: for each $\tau_j \in A^{\partial}$ do
3: $B_j := \{r_k \in T(\tau_j) | l(g(r_k)) \notin l(g(r_i))$ for all $r_i \in$
4: $C_j := \{r_k \in B_j | P_b(\rho)|_{T_j)} > \sigma\}$
5: if $C_j = \emptyset$ then
6: $\mathcal{T}_c := \mathcal{T}_c \cup \{\tau_j\}$
7: end if
8: end for
9: while $\mathcal{T}_c \neq \emptyset$ do
10: $\tau_z := arg\max_{\tau_{j_\in} \mathcal{T}_C} \psi(\tau_j, \mathcal{T}_c)$
11: $C_z := \{r_k \in T^S(\tau_Z) | P_b(\rho(r_k)|\tau_Z) > \sigma]$
12: $p_z^q := arg\max_p \ \tau\varphi(p_\angle \tau - -\mathcal{T}_c)$
13: $D_z := \emptyset$
14: while $C_z \neq \emptyset$ do
15: $\gamma_i := arg\max_{rk \in c_z} \theta(\rho(r_k))$
16: $D_Z := D_z \cup \{r_k\}$
17: $T^S := T^S - \{r_i\}$
18: $\tau(r_i) := \tau(r_i) - \{p_z^q\}$
19: $T^S := T^S \cup \{r_i\}$
20: $C_z := \{r_k \in T\grave{s}(\tau_z) | P_b(\rho\ (\ r_k)|\tau_z) > \sigma\}$
21: end while
22: for each $r_i \in D_z$ do
23: $\mathcal{T}_c := \mathcal{T}_c \cup$ MCST $(T^S, \tau(r_i), p_z^q)$
24: end for
25: $\mathcal{T}_c := \mathcal{T}_c - |\tau_z|$

26: end while
27: $T^G := \mathcal{T}^S$
2S: return T^G

MPSTD 接收经过 SA 泛化的数据库和所有可能的攻击序列作为输入，返回轨迹经过处理的数据库$\mathrm{T^G}$作为结果，此时的$\mathrm{T^G}$面对任何攻击序列，每一行的泄露概率都不会大于阈值。

上面是 MPSTD 算法的伪代码，其具体每行的意义如下：

1-8：对于 STR 算法结果中的每一种攻击序列，找出泄露概率最大的数据行集合B_j，并在B_j中找到泄露概率大于阈值的数据库C_j，此时如果C_j不为空，那么说明对于τ_{j}，数据库中有危险的行，τ_{j}是一个危险序列，被加入到危险轨迹集合T_c中。

10-13：对于不为空的$\mathrm{T_c}$，找出压缩分数最大的轨迹τ_{z}，计算出对于τ_{z}匹配出的行的泄露概率大于阈值的数据库的行的集合C_z，找出τ_{z}中压缩分数最大的移动点p_z^q。

15-20：处理泄露概率大于阈值的行的集合C_z，处理每一行直到C_z为空，找出隐私需求水平最大的行先处理，将此行加入D_z，对T^s中的r_i删除其轨迹内的p_z^q，然后对此行计算泄露概率。

22-25：处理完C_z，删除某些点以后，虽然当时此行的泄露概率小于了阈值，但是之后对其他行的改变可能导致此行泄露概率再次超过阈值，此时若对处理后数据库的再次使用 MPSTD 算法，显然处理代价非常大，因此论文提出了 MCST 算法，以最小的代价重新生成危险轨迹集合，对删除了p_z^q后数据库再次计算危险轨迹集合T_c，完成后从T_c中删去τ_{z}，进入下一轮循环。

27：此时T^s即处理后的数据库，其中导致某些泄露概率处于阈值以上的行的移动节点p_z^q都已被删去。

4.3.2 MCST 算法

Input:
T^S: SA - generalized trajectory database
τ : Trajectory
p_z^q : Moving point
Output:
T_c: Set of sub-trajectories
1: $\mathcal{T}_c := \emptyset$
2: $A_1 := \{\langle p_z^q \rangle\}$

3: $i:=1$
4: while $i \leq \delta$ and $A_i \neq \emptyset$ do
5: for each $\tau_j \in A_i$ do
6: if $(P(\rho(r_k)|\tau_j) > \sigma)$ for some $r_k \in T^S(\tau_j)$ then
7: $T_c:=\mathcal{T}_c \cup \{\tau_j\}$
8: end if
9: end for
10: $A_{i+1}:=\{\tau_k \in \tau_z \ |p_z^q \in \tau_k \wedge |\tau_k| = i+1\}$
11: $i = i+1$
12: end while
13: return $\mathcal{T}_c$.

MCST 算法接收删除过移动节点p_z^q的新数据库T^s作为输入，同时根据当前的攻击序列τ_z和危险轨迹点p_z^q，重新计算危险轨迹数据库的危险攻击序列并返回。

上面是 MCST 算法的伪代码，其具体每行的意义如下：

1-3：设置待返回的T_c为空集，$\mathcal{A}_1$初始化为只有点p_z^q作为唯一攻击序列的集合；

4-9：当循环次数小于等于攻击序列的最大长度并且$\mathcal{A}_i$不为空时，对于$\mathcal{A}_i$中的每一个攻击序列，如果在T^s的某行因为此序列泄露概率大于了阈值，说明此序列仍然危险，需要再次被加入到危险轨迹集合T_c内，保证 MPSTD 算法可靠性；

10：此时$\mathcal{A}_{i+1}$是在τ_z中所有拥有移动节点p_z^q并且长度等于$i+1$的所有轨迹的集合，并进入下一轮计算。

13：返回危险轨迹集合T_c供 MPSTD 再次计算。

5 安全分析

在介绍完 PPTD 内的 SAGTD 算法和 MPSTD 算法的详细步骤之后，我们根据定理及其证明来对算法的安全性进行数学上的分析和确认。

5.1 属性泛化与泄露概率

首先给出定理 1，给定一个轨迹数据库 T 和两个任意的轨迹数据记录$r_{i_1}, r_{i_2} \in T$，让T_j为$\tau(r_{i_1})$和$\tau(r_{i_2})$的子轨迹，如果r_{i_1}的守护节点的是r_{i_2}的守护节点的祖先节点，用公式表达就是$\ell(g(r_{i_2})) \subset \ell(g(r_{i_1}))$，那么$P_b(\rho(r_{i_2})|\tau_j)$的值就和$P_b(\rho(r_{i_1})|\tau_j)$相等或者比其小，不管是否实施了属性泛化。

证明如下：$\eta(r_k)$为轨迹数据记录$r_{\mathrm{k}} \in \mathrm{T}(T_j)$的孪生节点，因为$g(r_{i_2})$是$g(r_{i_1})$的祖先节点，因此$|\ell(g(r_{i_2})) \cap \ell(\eta(r_k))|$比$|\ell(g(r_{i_1})) \cap \ell(\eta(r_k))|$小或相等，根据公式（2.15），$P_b(\rho(r_{i_2})|\tau_j)$一定比$P_b(\rho(r_{i_1})|\tau_j)$小或者和其相等。

当搜索哪些行需要进行属性泛化时，我们可以避免去泛化如r_{i_2}这样的结点，因为只要r_{i_1}的隐私泄露概率降到了阈值一下，由于r_{i_2}的泄露概率始终比r_{i_1}要小，所以r_{i_2}的隐私泄露概率也会降到阈值以下而不需要再进行泛化。

这就是 SAGTD 算法第 3 行先求出B_j的原因。

然后给出定理 2，如果在背景攻击序列ξ_i下，r_{i}的泄露概率大于泄露概率阈值，那么在实施了属性泛化之后，那么r_{i}的属性一定会被设置成r_{i}敏感属性对应的守护节点的祖先节点中的一个，其中r_{i}的敏感属性会变成 SA 泛化后的r_{i}的属性值。

证明如下：很明显，在实施了属性泛化后，对应行的泄露概率一定会减少，假设一个相反的情况，r_{i}的敏感属性的值没有被设置成其守护节点的祖先节点中的一个，那么$\eta(r_i)$被$g(r_i)$所覆盖，而公式（2.13）中的结果泛化前后相同，即泛化前后泄露概率不变，显然矛盾。

定理 2 证明了属性泛化的结果是行对应的守护节点的祖先节点中的一个，进而证明了算法的正确性。

5.2 轨迹压缩与泄露概率

定理 3，一个匿名隐私保护数据表满足个性化隐私当且仅当它不包含危险子轨迹。

证明：假设数据表在不包含任何危险子轨迹时也不是隐私匿名的。因此，对于不是隐私匿名的数据表，攻击者可以实施至少身份链接攻击，属性链接攻击，相似攻击中的

一种时，结果是，根据三种攻击模型的定义，一定会有子轨迹ξ_i让某行的泄露概率大于阈值，这时ξ_i就是危险子轨迹，这就和假设矛盾，所以定理成立。

定理表明当一个数据表没有任何危险子轨迹时，它就是一个隐私匿名的数据表，并且能够抵抗身份链接攻击，属性链接攻击和相似攻击。

定理 3 为局部轨迹压制提供了理论基础，证明去除某些节点让数据表不再有危险子轨迹后，数据表就能变成保护个性化隐私的匿名数据表。

6　代码实现和仿真实验

我们在细致地读完论文的基本定义并通读了一遍伪代码，对算法的思想有了基础的掌握之后，就开始构思使用什么语言实现，如何实现论文中 SAGTD、MPSTD 等五个算法。想来想去，由于需要今年笔者要考研究生，以后在复习数据结构课程时一定会用到 C 语言的相关知识，不如趁此写作业实现算法的机会复习熟悉一遍 C 语言，于是敲定了使用 C 语言实现论文算法。

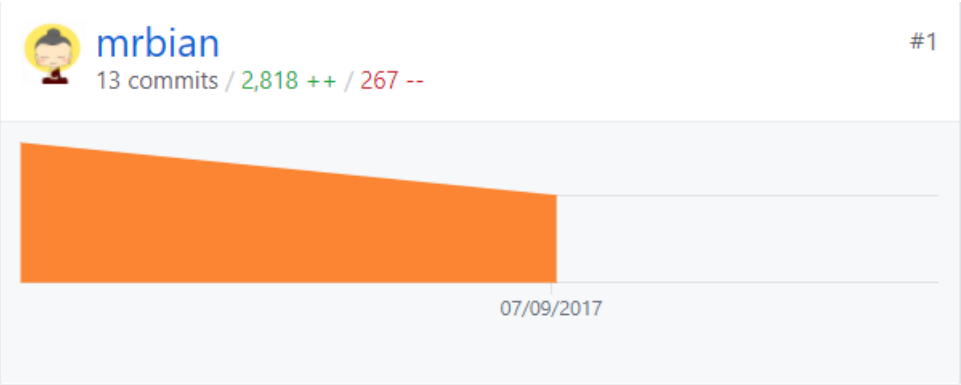

图 6.1　Github 上 PPTD 算法代码库代码统计

笔者编程过程中熟悉了 C 的编码风格，同时复习了链表、队列、树等数据结构，反复的 debug 让笔者对 C 语言的内存管理有了一个新的认识。

6.1　代码设计

论文中主要涉及到的一些对象有数据表、敏感属性树、移动轨迹、攻击序列集合等。

无论任何系统，在最开始的时候一定是从简单的架构开始做起，在实践中不断的迭代变复杂的，因此，笔者在实现代码的过程中也是尽可能从最简单的版本做起。

首先论文的最基础结构有轨迹数据库和敏感属性树，因此，可以先实现这两个结构作为代码的第一个版本，数据库的数据和敏感属性树的数据都通过简单的 txt 文件来读取，这样将代码与数据解耦，能够极大的提高系统的可重用性。

实现基础数据结构层后，对数据结构层进行单元测试；然后实现算法逻辑层，包含 SAGTD、STR、SAG、MPSTD、MSCT 五种算法，进行单元测试；然后实现 UI 界面层，实现完成后就可以对整个系统进行系统测试。

6.1.1 总架构图

系统的总架构图如图 6.2 所示。程序运行起来后主函数调用数据源和 UI 界面层让用户来进行操作，在主函数中，控制数据源，将其填入数据结构层，产生数据结构供算法逻辑层调用，算法逻辑层处理完成后，将返回结果传给 UI 界面层，UI 界面层再展示给用户。

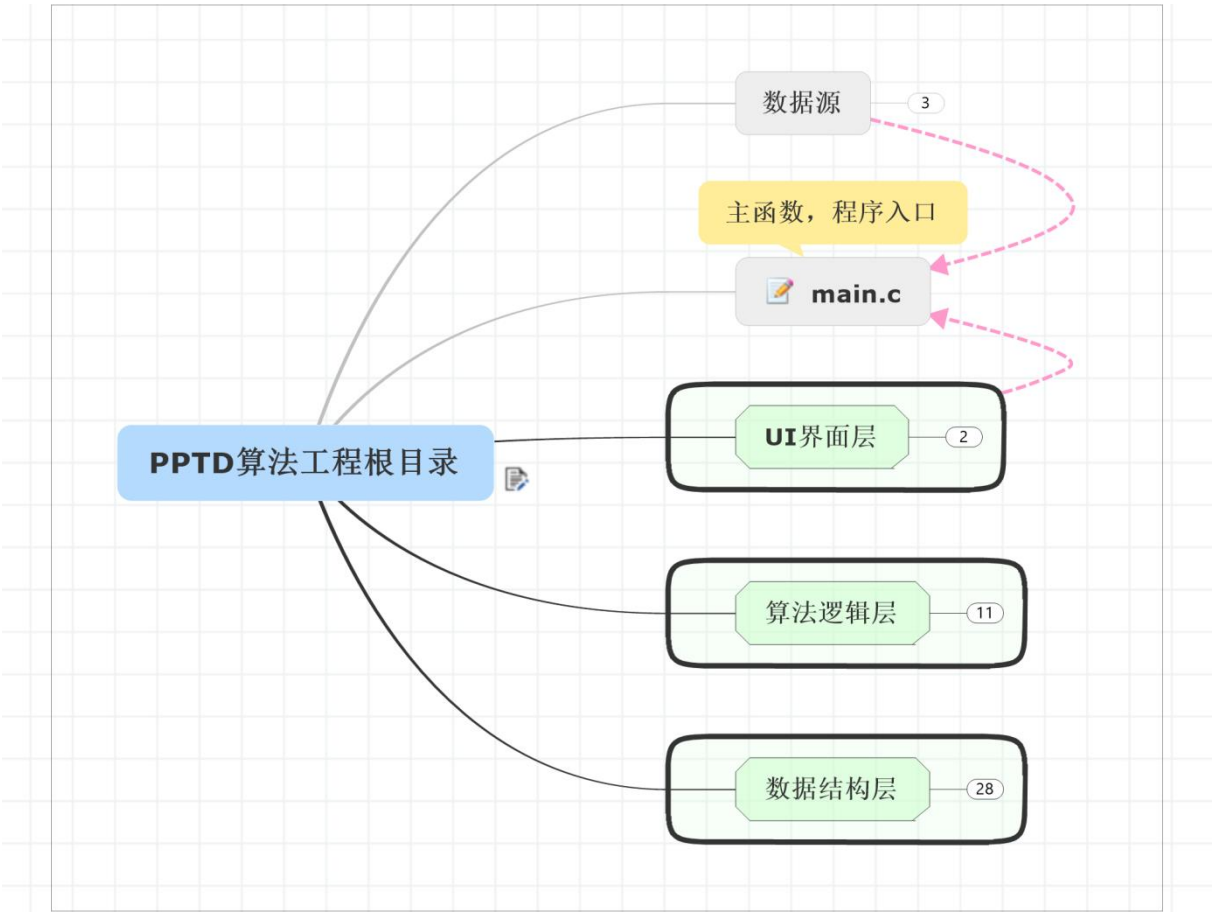

图 6.2 工程基础结构图

6.1.2 数据结构层

图 6.2 中的数据结构层的结构图如图 6.3 所示，根目录下 Queue 文件夹存储队列结构的程序文件，Database 文件夹存储轨迹数据库相关的程序文件，Tree 文件夹存储敏感属性树的相关程序文件。TrackSet 存储与轨迹序列及其集合相关数据结构定义以及其操作。

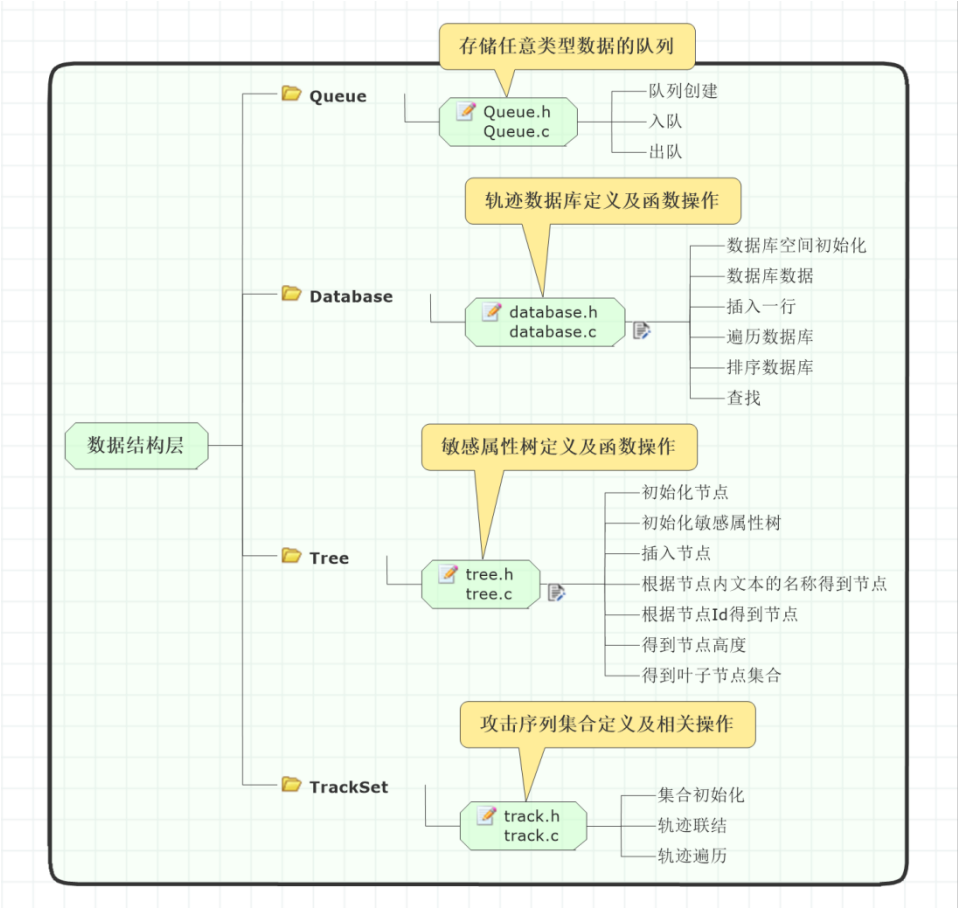

图 6.3　数据结构层结构图

数据结构层会从数据源中读取数据。图 6.4 是数据源结构图，数据源文件存放在 config 文件里，只需要对被算法处理的数据表进行更改时，只需要对 database.txt 和 sensitive_tree.txt 进行修改即可。

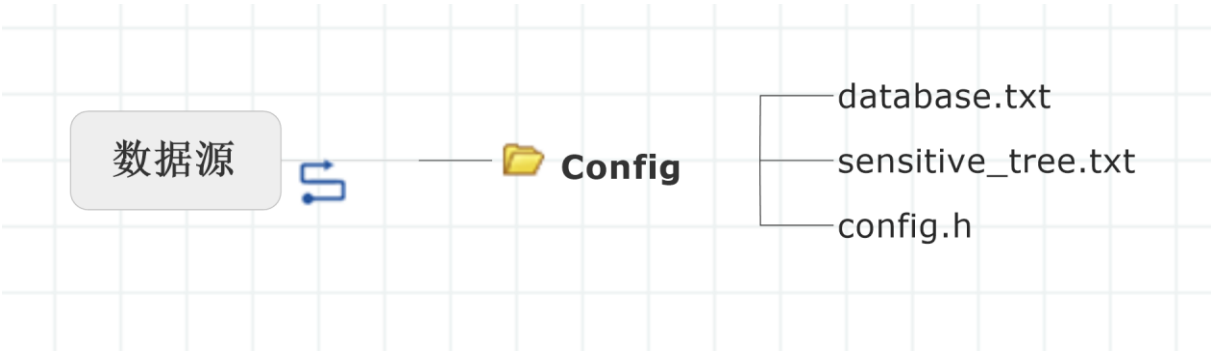

图 6.4　数据源结构图

6.1.3 算法逻辑层

图 6.2 中的算法逻辑层的具体结构图如图 6.5 所示，SAGTD 文件夹存储了和属性泛化相关的所有算法程序文件，下有 STR、SAG 两个文件夹，SAGTD 会在主函数中调用 STR 算法和 SAG 算法，这三种算法都会用到对数据库，集合，敏感属性树这些数据结构的基础操作，如计算泄露概率，根据攻击序列匹配行，数据库取并集交集操作，以及极其常用的得到某行的守护节点和孪生节点的操作。而 MPSTD 算法包含 MCST 算法，其中比较重要的函数就是通过计算压缩分数得到危险轨迹点和危险轨迹行。

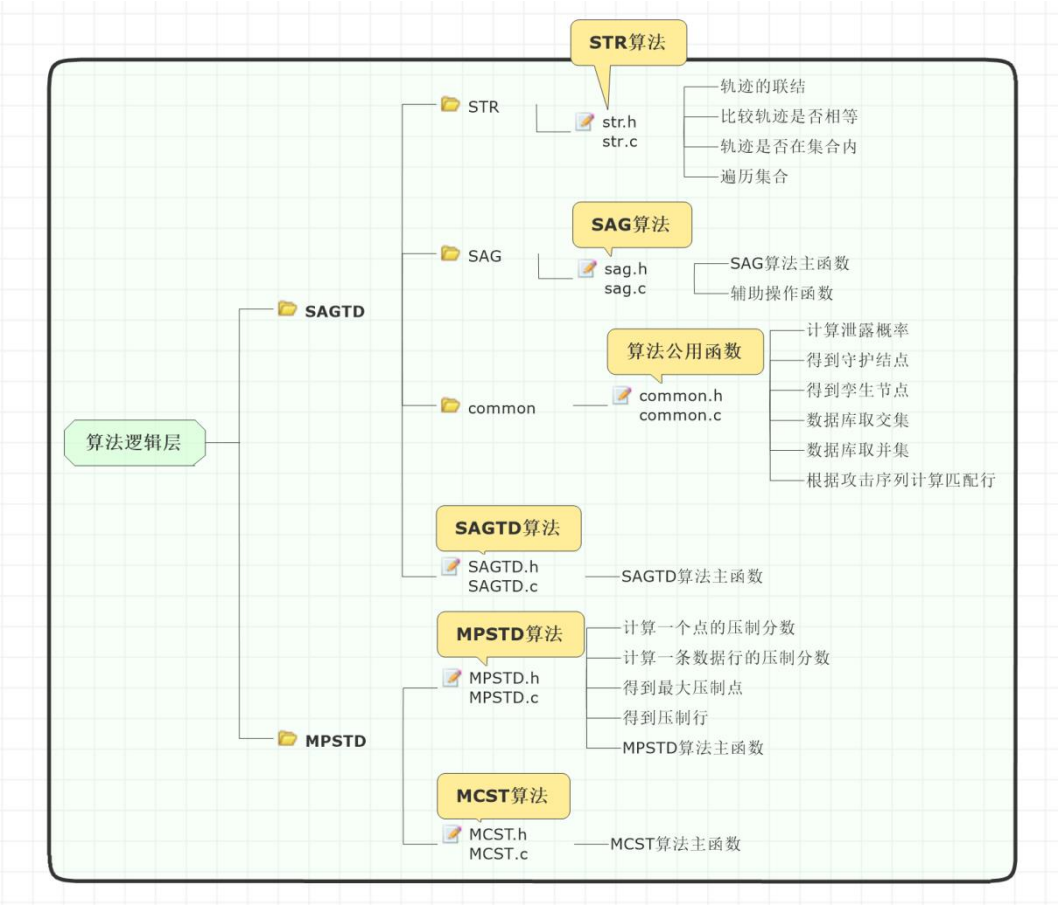

图 6.5 算法逻辑层结构图

6.1.4 UI 界面层

UI 界面层较为简单，其结构如图 6.6 所示，对用户输入处理后打印输出。

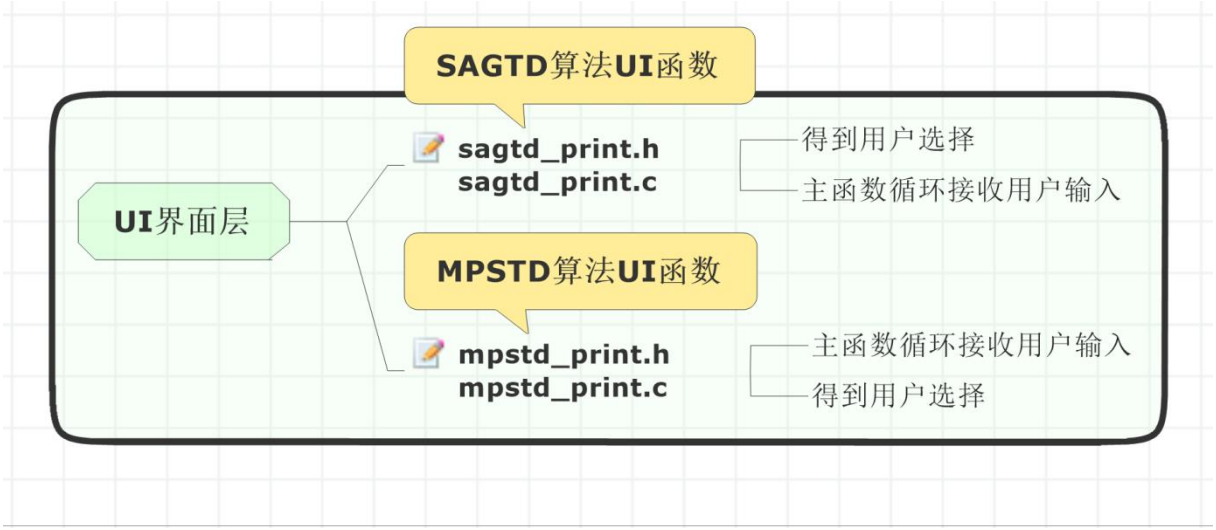

图 6.6　UI 界面层结构图

6.2　主要数据结构代码解析

如图 6.3 数据结构层所示，代码中主要用到的数据结构有轨迹数据库的定义，敏感属性树的定义，攻击序列的及其集合结构定义，以及辅助数据结构队列的结构定义。

6.2.1　轨迹数据库结构定义

轨迹数据库的结构体定义如图 6.7 所示。图 6.7 中 16-22 行是数据库中行的结构体定义，整型变量 id 代表行的 id，整型变量 p_level 代表行的隐私等级，在范例里 p_level 的取值范围为-1 到 2，二维字符数组 trajectory 为存储轨迹序列的数组，整型变量 trajectoryCount 记录着序列的长度，如存储<b2,d3>，那么 trajectoryCount 的值就是 2。

图 6.7 中第 24 行是数据库的指针，数据库其实就是行的结构体指针数组，存储一定数目的指向行的结构体的指针。

```
16: typedef struct row{
17:     int id;
18:     int p_level;     // 隐私等级
19:     char** trajectory;
20:     int trajectoryCount;
21:     char disease[disease_len];
22: }row;
23:
24: typedef row* database; // 存储指向结构体的指针
```

图 6.7　数据库行结构体和数据库指针

6.2.2 敏感属性树结构定义

定义完了数据库，就需要定义和构建敏感属性树。敏感属性树的结构体定义如图 6.8 所示的结构体。图 6.8 中 17-23 行是敏感属性树的节点的定义，其中 parent 指针指向一个节点的双亲节点，childLists 为该节点子节点的数组，最大长度已在宏内定义好，text 变量代表该节点所包含的病的信息，以字符串的形式存储，整型变量 parentId 代表当前节点的父节点的 Id，整型变量 selfId 为当前节点的 Id，使用 Id 的方式让树的创建、查找等操作变的非常简单。

```
17: typedef struct treeNode{
18:     struct treeNode * parent;
19:     struct treeNode * childLists[childListsLength];     // 最大五个子节点
20:     char text[MAX_TEXT_LEN];                 // 事先分配好空间
21:     int parentId;
22:     int selfId;
23: } treeNode;
```

图 6.8 敏感属性树节点结构体

6.2.3 攻击序列及其集合结构定义

在定义好了轨迹数据库和敏感属性树后，就需要对 STR 和 SAGTD 以及 MPSTD 中的出现的攻击序列及其集合进行结构定义，在这点上笔者首先想到的是三维字符数组，因为集合本身是一个数组，集合内每一个元素都是移动节点的集合，又是一层数组，移动节点本身又是一层字符数组，但是这样很明显操作起来异常复杂，想到指针的指针的指针，我的天啊，这不是作死吗？于是，想到了万能的结构体，定义了如图 6.9 所示的结构体，图 6.9 中 16-19 行是轨迹序列集合内一个元素的定义，里面有移动点集合和相应的移动点数目，22-25 行是轨迹序列集合的定义，存储轨迹集合内元素的指针和轨迹元素的数目。

```
15: // 集合内每一个元素
16: typedef struct trackRow{
17:     char ** tracks;      //路径链
18:     int count;  // 路径点长度
19: }trackRow;
20:
21: // 集合
22: typedef struct trackSet{
23:     int count;
24:     trackRow ** trackCollection;
25: }trackSet;
```

图 6.9　轨迹序列和轨迹序列集合的结构体

6.2.4　队列结构定义

因为需要对树进行遍历，并且显然对树而非二叉树的遍历使用递归在性能上是非常不合适的，因此需要使用队列或栈，通过节点出队入队操作对树实现广度优先遍历。同时队列需要能够存储任意类型的数据，这在 C 语言里只能通过 void 类型的指针来实现，同时在入队操作时，需要使用 memcpy 函数来复制函数在传参时的堆上的内存。图 6.10 是队列结构体的相关定义，其中第 16 行为存储数据的数组，18 行记录当前队列的头位置，19 行记录当前队列的尾位置，20 行存储数据域所存放数据的字节大小，这决定了在入队操作中复制的内存块的大小。

```
14: typedef void * T;
15: typedef struct Queue{
16:     T * queue;
17:     int maxSize;
18:     int front;
19:     int rear;
20:     size_t dataSize;
21: }Queue;
```

图 6.10　队列结构体定义

6.3　主要函数代码解析

这里只拣出了较为重要的函数进行了详细解析，对于 SAGTD 和 MPSTD 等算法，其函数代码实在过长，即使展示一个都会占去想当大的篇幅，而且基本都是按照伪代码逻辑逐行实现，而伪代码的各行逻辑在第 4 部分 PPTD 已经有所介绍，因此不再赘述，部分代码贴在附录内，完整代码查看请移步[10]。

这里选择了较为重要的用于计算攻击序列的轨迹联结函数、根据攻击序列计算泄露概率的函数、计算个性化压缩分数的函数。

6.3.1 轨迹的联结

轨迹的联结函数如图 6.11 所示，根据 2.2 中轨迹的联结的定义，可知如果两个轨迹序列长度不一样或者前 n-1 个移动点不同，或者没有时间差，两个轨迹都是不可联结的。因此图 6.11 中第 88 行先判断了空轨迹、轨迹个数相同、没有时间差三种情况，如果有这三种情况，就返回空结果；第 98 行判断了前 n-1 个移动点是否相同，如果相同，说明此时满足联结条件，在第 109 行判断了时间差后确认最终的结果轨迹。

```
83: trackRow * joinTracks(trackRow * track1, trackRow * track2){
84:     int count1 = track1->count;
85:     int count2 = track2->count;
86:     int timediff = (track1->tracks[count1-1][1] - '0') - (track2->tracks[count2-1][1] - '0');   // 时间差
87:
88:     if(count1 == 0 || count2 == 0 || (count1 != count2) || timediff == 0)
89:         return NULL;
90:
91:     trackRow * result = (trackRow *)malloc(sizeof(trackRow *));
92:     result->tracks = (char **)malloc(sizeof(char *) * String_Max_Len * (track1->count));
93:     result->count = track1->count + 1;
94:
95:     int i,j;
96:
97:     // 如果可联结
98:     if(cmpTracks(track1->tracks,
99:                  track1->count - 1,
100:                 track2->tracks,
101:                 track2->count - 1
102:    )){
103:        // copy i - 1
104:        for(i=0;i< count1-1;i++){
105: //           memcpy(result->tracks[i],track1->tracks[i],sizeof());
106:            result->tracks[i] = track1->tracks[i];
107:        }
108:        // copy i i+1
109:        if(timediff < 0){
110:            result->tracks[count1 - 1] = track1->tracks[count1 - 1];
111:            result->tracks[count1] = track2->tracks[count1 - 1];
112:        }else{
113:            result->tracks[count1 - 1] = track2->tracks[count1 - 1];
114:            result->tracks[count1] = track1->tracks[count1 - 1];
115:        }
116:
117:    } « end if cmpTracks(track1->tra... » else{
118:        result = NULL;      // 注意不成功一定设置成NULL
119:    }
120:
121:    return result;
122: } « end joinTracks »
```

图 6.11 轨迹联结函数实现

6.3.2 泄露概率计算函数

泄露概率的计算函数包含两部分，一部分是公式（2.13）中针对 rk 计算 ri 的泄露概率，一部分是公式(2.15)中在攻击序列背景下计算出 ri 的泄露概率。图 6.12 是公式(2.13)的实现，图 6.13 的函数代码是公式（2.15）的实现。图 6.12 针对某行的泄露概率计算函数中，程序首先根据守护节点和孪生节点的关系，判断结果属于公式（2.13）中的哪一种，判断结束后，如果需要计算，那么就计算孪生节点和守护节点的子集，对集合取交

集后，第 31 行分子除分母得到计算结果，同时保留两位小数返回。需要提到的是，隐私保护需求为 No Privacy 的行的泄露概率置为 0。

```
6:  // 根据ri的guard node和rk的twin node计算概率
7:  float _getProbabilityByNode(treeNode * root, treeNode * guardingNode, treeNode * twinNode){
8:      float result = 0;
9:      if(nodeAisCoveredByNodeB(root,twinNode,guardingNode)|| guardingNode == twinNode){
10:
11:         return 1;
12:
13:     }else if(nodeAisCoveredByNodeB(root,guardingNode,twinNode) ){
14:
15:         int i,j,flag;   // 集合A中元素是否匹配到集合B的元素
16:         int mem = 0;    // 分子
17:         int den = getSubsetsLengthOfNode(twinNode);     // 分母
18:         treeNode **guardingNodeSets = getSubsetsOfNode(guardingNode);
19:         treeNode **twinNodeSets = getSubsetsOfNode(twinNode);
20:
21:         for (i = 0; i < getSubsetsLengthOfNode(twinNode); i++) {
22:             flag = 0;       // 默认未匹配到
23:             for (j = 0; j < getSubsetsLengthOfNode(guardingNode); j++) {
24:                 if (twinNodeSets[i] == guardingNodeSets[j])
25:                     flag = 1;
26:             }
27:             if (flag)
28:                 mem++;
29:         }
30:
31:         result = (float) mem / (float) den;
32:         return ((float)((int)(result * 100))) / 100;
33:     } « end if nodeAisCoveredByNodeB... » else{
34:         return 0;
35:     }
36:
37: } « end _getProbabilityByNode »
```

图 6.12　针对某行泄露概率计算函数实现

图 6.13 中针对某攻击序列计算函数首先在第 44 行调用函数得到攻击序列所匹配到的数据表的行，然后在第 51 行到第 57 行，对攻击序列匹配到的新数据表内的每一行运行图 6.12 中的函数，结果加入结果集内，最后在第 59 行到 61 行得到结果。

```
39: float caculateBreachProbability(database * originDb, database * db,treeNode * root, int rowId, char ** background, int backgroundCount){
40:     row * row = getRowById(db,rowId);           // 待求值的行
41:     if(row->p_level == -1){     // 如果是No Privacy
42:         return 0;
43:     }
44:     database * rowsMatched = matchRowByTrajectory(db,background,backgroundCount);   // 攻击者匹配到的行
45: //    printf("matched %d\n",getLengthOfDB(rowsMatched));
46:     treeNode * guardingNode = getGuardingNodeByRow(originDb, root, rowId);   // ri的guard node
47:     treeNode * twinNode;        // rk 的twin node
48:
49:     int i,count = 0;    // count 记录Tk行数
50:     float result = 0;
51:     for(i=0;i<row_count;i++){
52:         if(rowsMatched[i] != NULL){
53:             twinNode = getTwinNodeByRow(root,rowsMatched[i]);   // rk 的twin node
54: //            printf("ri is r%d, rk is r%d,the p is %.2f\n",rowId,rowsMatched[i]->id,_getProbabilityByNode(root,guardingNode,twinNode));
55:             result += _getProbabilityByNode(root,guardingNode,twinNode);
56:             count ++;
57:         }
58:     }
59:     result = result  / (float)count;
60:     result = ((float)((int)(result * 100))) / 100;
61:     return result;
62: } « end caculateBreachProbability »
```

图 6.13　针对某攻击序列泄露概率计算函数实现

6.3.3 个性化压缩分数计算函数

个性化压缩分数的计算包含两部分，一部分是计算某个移动点的个性化压缩分数，一部分是计算某轨迹的个性化压缩分数。图 6.14 是两个函数的代码截图。计算某移动点的压缩分数时，第 25 行首先根据攻击序列得到匹配到的轨迹的集合，在第 28 行得到匹配到的轨迹集合的长度，此长度为公式（2.16）的分子，第 29 行得到匹配到的数据表的行数，为公式（2.16）的分母，在第 30 行到 38 行对匹配到的数据表中的各行的隐私需求求和，此为公式（2.16）中右侧的乘数。第 43 行到第 52 行在计算一个轨迹的个性化压缩分数时，对所求轨迹内的每一个移动点都会进行计算，对各个移动点中压缩分数比较后，返回最大值即结果。

```
23: float caculatePssOfPoint(char * track, trackRow * row, trackSet * Tc, database *db){
24:     int i;
25:     database * matched = matchRowByTrajectory(db,row->tracks,row->count);
26:     float result = 0;
27:     float p_level_sum = 0;
28:     int row_len = getLenOfMatchRowByPoint(track,Tc);    // 分子
29:     int den = getLengthOfDB(matched);
30:     for(i=0;i<row_count;i++){
31:         if(db[i] == NULL)
32:             continue;
33:
34:         if(db[i]->p_level != -1){       // 去除no privacy
35:             p_level_sum += db[i]->p_level;
36:         }
37:     }
38:     result = ((float)row_len / (float)den) * p_level_sum;
39:
40:     return result;
41: }
42:
43: float caculatePssOfRow(trackRow * row, trackSet * Tc, database * db){
44:     float result = 0;
45:     float val;
46:     int i;
47:     for(i=0;i<row->count;i++){
48:         val = caculatePssOfPoint(row->tracks[i],row,Tc,db);
49:         result = val > result ? val : result;
50:     }
51:     return result;
52: }
```

图 6.14　个性化压缩分数计算函数实现

6.4　仿真实验

论文的仿真实验是根据 Metro100K 和 City80K 这两个数据集进行的，笔者虽然上网找到了两个数据集所对应的论文，但是没有能够下载到相关数据集，只能自己通过扩展

论文范例中的例子来对尽可能的模拟大数据量下的实验结果，我们通过对论文范例的扩展，得到了我们称为 Simulation1K 的数据集。

6.4.1 仿真环境

代码的编写和运行都是在 Windows 下进行的，使用了 Jetbrains 公司的 Clion 进行编码，代码运行环境为 CPU 2.10GHz，RAM 8G。程序编译环境如图 6.15 所示。

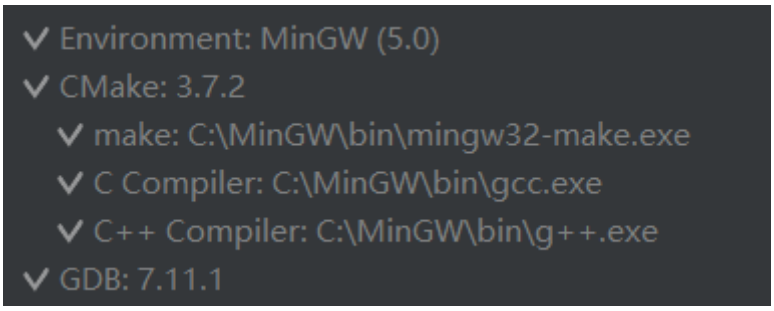

图 6.15　程序编译环境

6.4.2 仿真数据

图 6.16 是轨迹数据库的部分仿真数据的截图，其中第一列为数据库内行的 ID，第二列是行的隐私需求水平对应的整数，之后一直到空格处，都是由逗号划分的轨迹序列集合，最后一列是该行的敏感属性值，模拟数据的行树是可变的，通过函数生成，下面的性能测试里面用到的数据有 1400 行。所以称之为 Simulation1K。

```
1377,1,d3,f6,e9 Flu
1378,0,c4,d5,f6 Diabetes
1379,-1,b2,f6,e9 Cold
1380,0,b2,d3,c4,f6,a7,e8 HIV
1381,1,c4,f6,a7,e9 SARS
1382,0,d3,c4,a7 Pancreatitis
1383,2,b2,f6,a7,e8 HIV
1384,1,d5,f6,e9 Flu
1385,0,c4,d5,f6 Diabetes
1386,-1,b2,f6,e9 Cold
1387,0,b2,d3,c4,f6,a7,e8 HIV
1388,1,c4,f6,a7,e9 SARS
1389,0,d3,c4,a7 Pancreatitis
1390,2,b2,f6,a7,e8 HIV
1391,1,d5,f6,e9 Flu
1392,0,c4,d5,f6 Diabetes
1393,-1,b2,f6,e9 Cold
1394,0,b2,d3,c4,f6,a7,e8 HIV
1395,1,c4,f6,a7,e9 SARS
1396,0,d3,c4,a7 Pancreatitis
1397,2,b2,f6,a7,e8 HIV
1398,1,d5,f6,e9 Flu
1399,0,c4,d5,f6 Diabetes
1400,-1,b2,f6,e9 Cold
```

图 6.16　轨迹数据库部分仿真数据

图 6.17 是敏感属性树的仿真数据，其中第一列是节点自己的 ID，第二列是节点的父节点的 ID，第三列是该节点所带的敏感属性，模拟数据的敏感属性树的高度为 3。配置文件中的每一行都是一个节点，实现代码中根据节点自身的 ID 和父节点的 ID 在初始化了节点之后插入到树中。

```
1,0,Any Illness
2,1,Infectious Disease
3,1,Pulmonary Disease
4,1,Non-healing Wound Disease
5,2,Weakness of Immune System
6,3,Restrictive Lung
7,3,Lung Infection
8,3,Obstructive Lung
9,4,High Blood Sugar
10,5,Lupus
11,5,HIV
12,5,Padding1
14,6,Pneumonia
15,6,Pleurisy
13,6,Padding2
16,6,Padding3
17,6,Padding4
18,7,Flu
19,7,Cold
20,7,SARS
21,8,Bronchitis
22,8,Asthma
23,8,Padding5
24,8,Padding6
28,8,Padding8
25,9,Diabetes
26,9,Pancreatitis
27,9,Padding7
```

图 6.17　敏感属性树仿真数据

6.4.3　程序运行截图

程序运行后，用户需输入最长攻击序列长度，泄露概率阈值，主界面运行后截图如图 6.18 所示。

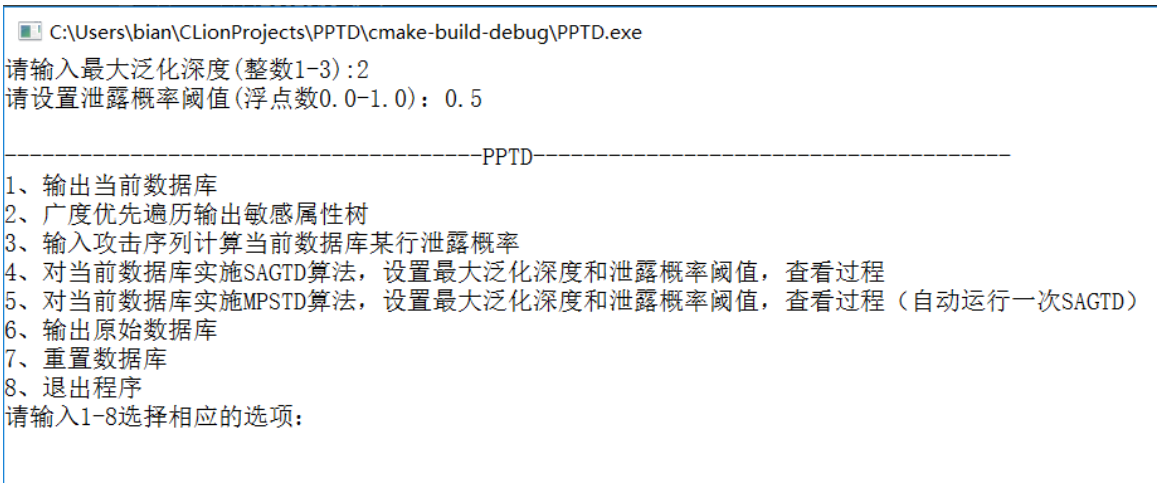

图 6.18　程序主界面截图

在主界面选择选项 4 后，进入 SAGTD 算法界面，STR 算法的运行结果如图 6.19 所示，其中设定的攻击序列的最大长度为 3。

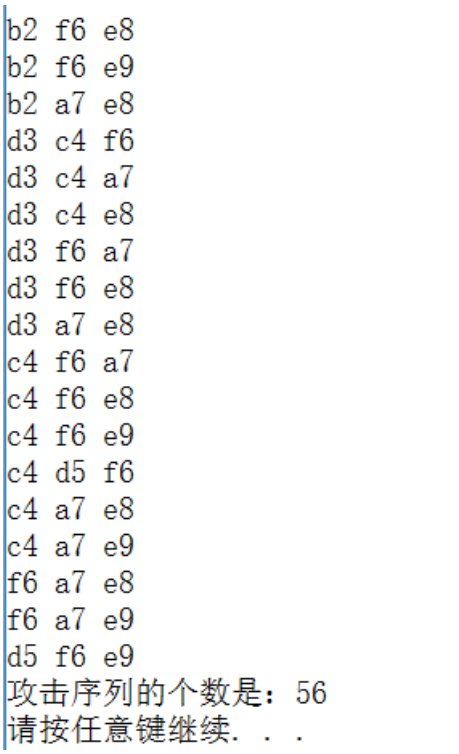

图 6.19　STR 算法部分结果截图

SAGTD 算法的运行结果如图 6.20 所示，第一列显示的是行的 ID，第二列是隐私保护需求对应整数，第三列是泛化后的敏感属性值，第四列是轨迹数据集合。可以看到 SAGTD 处理 Simulation1K 这个有 1400 行的数据表用了 152 秒，时间大概为 2 分半。

```
1373 0    HIV          <b2,d3,c4,f6,a7,e8,>
1374 1    SARS         <c4,f6,a7,e9,>
1375 0    Pancreatitis  <d3,c4,a7,>
1376 2    Any Illness  <b2,f6,a7,e8,>
1377 1    Flu          <d5,f6,e9,>
1378 0    Diabetes     <c4,d5,f6,>
1379 -1   Cold         <b2,f6,e9,>
1380 0    HIV          <b2,d3,c4,f6,a7,e8,>
1381 1    SARS         <c4,f6,a7,e9,>
1382 0    Pancreatitis  <d3,c4,a7,>
1383 2    Any Illness  <b2,f6,a7,e8,>
1384 1    Flu          <d5,f6,e9,>
1385 0    Diabetes     <c4,d5,f6,>
1386 -1   Cold         <b2,f6,e9,>
1387 0    HIV          <b2,d3,c4,f6,a7,e8,>
1388 1    SARS         <c4,f6,a7,e9,>
1389 0    Pancreatitis  <d3,c4,a7,>
1390 2    Any Illness  <b2,f6,a7,e8,>
1391 1    Flu          <d5,f6,e9,>
1392 0    Diabetes     <c4,d5,f6,>
1393 -1   Cold         <b2,f6,e9,>
1394 0    HIV          <b2,d3,c4,f6,a7,e8,>
1395 1    SARS         <c4,f6,a7,e9,>
1396 0    Pancreatitis  <d3,c4,a7,>
1397 2    Any Illness  <b2,f6,a7,e8,>
1398 1    Flu          <d5,f6,e9,>
1399 0    Diabetes     <c4,d5,f6,>
1400 -1   Cold         <b2,f6,e9,>
Process 1400 line data, sagtd use 152.793000 seconds
请按任意键继续. . .
```

图 6.20　SAGTD 算法结果截图

MPSTD 算法的结果如图 6.21 所示，其使用的源数据库是图 6.20 中的数据库，修改的是第四列的轨迹数据，将第 4 行、11 行以及其他敏感属性为 Any Illness 的行的 b2，e8 的移动点删去了。

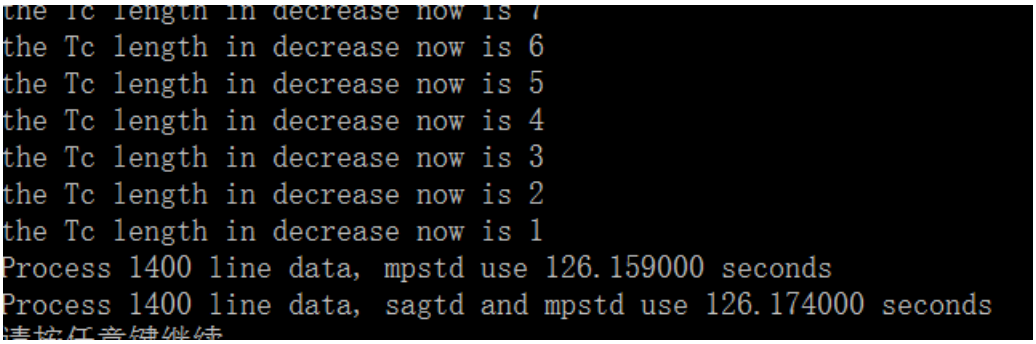

图 6.21　MPSTD 算法结果截图

6.5　结果安全性与性能分析

虽然论文所使用的 City80K 和 Metro100K 的数据集笔者没有找到，但是我们自行生成了 7K 的数据，我们称之为 Simulation7K，通过这 7000 行数据，也是可以做一些微小的统计的。

6.5.1　结果安全性分析

论文使用泄露风险作为衡量移动对象的隐私泄露风险的度量。给定一个 PPTD 匿名处理后的数据库$T^{\mathcal{G}}$和其原始数据库T，让r_i为在$T^{\mathcal{G}}$中的一行并且$\xi_i \in T(o(r_i))$是一个攻击者序列，其中$o(r_i)$为轨迹数据库中的某行r_i所对对应的移动对象，那么这个对象的敏感信息$s(o(r_i))$在给定ξ_i的情况下泄露的概率通过公式（6.1）来计算。

$$P(s(o(r_i))|\xi_i) = \begin{cases} \dfrac{1}{|T^{\mathcal{G}}(\xi_i)|} \displaystyle\sum_{r_k \in T^{\mathcal{G}}(\xi_i)} P(s(o(r_i))|\Omega(r_k)) & \xi_i \sqsubseteq \tau(r_i) \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

而$P(s(o(r_i))|\xi_i)$是$s(o(r_i))$在给定泛化属性$\Omega(r_k)$后$r_k \in T^{\mathcal{G}}(\xi_i)$的泄露概率。其计算公式为公式（6.2）。

$$P(s(o(r_i))|\Omega(r_k)) = \begin{cases} \dfrac{1}{|\Omega(r_k)|} & s(o(r_i)) \in \Omega(r_k), \\ 0 & \text{otherwise.} \end{cases} \tag{6.2}$$

攻击者可能会使用任意长度不大于最大攻击序列长度的背景知识来进行隐私攻击。公式（6.1）和（6.2）的泄露概率应当在不同攻击序列情况下计算。

想要计算一个轨迹数据库的泄露风险，给定数据库$T^{\mathcal{G}}$，要计算$r_i \in T^{\mathcal{G}}$的泄露概率就需要用到公式（6.3）。

$$\mathcal{R}(r_i) = \frac{1}{|\mathcal{K}_i|} \sum_{\xi_i \in \mathcal{K}_i} P(s(o(r_i))|\xi_i) \quad (6.3)$$

其中$\mathcal{K}_i$为公式（6.4）中所示。

$$\mathcal{K}_i = \{\xi_i \mid \xi_i \sqsubseteq \tau(o(r_i)) \wedge |\xi_i| \leq \delta\}. \quad (6.4)$$

因此，我们通过公式（6.1）到公式（6.4）来对一个轨迹数据库在 PPTD 算法实施前后的隐私安全性进行量化。我们下面得到的安全性统计结果基于 Simulation1K 的 1400 行数据集。统计每一种需求的的平均隐私泄露概率分数，统计结果如表 6.1 和表 6.2 所示。部分 PPTD 泛化和轨迹压缩后的结果如图 6.22 和图 6.23 所示，处理 1400 行数据集共用时 152 秒，时长大约是 2 分半。

表 6.1　最大泛化深度为 1 时平均隐私泄露概率分数

Privacy level	$\delta = 2$			$\delta = 3$		
	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$
Low	3.1805	19.9533	19.6438	2.6817	19.3127	19.9007
Medium	0.0540	16.4161	19.5912	0.0244	14.1736	19.3483
High	0.0000	14.1543	19.2180	0.0015	12.4787	17.7954

表 6.2　最大泛化深度为 2 时平均隐私泄露概率分数

Privacy level	$\delta = 2$			$\delta = 3$		
	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$
Low	2.0609	19.9435	19.8027	2.5609	19.9115	19.9521
Medium	0.0375	16.0863	19.6463	0.0254	12.3736	19.6794
High	0.0008	13.8918	19.1033	0.0009	12.4385	18.8567

```
1375,0,d3,c4,a7 Pancreatitis
1376,2,b2,f6,a7,e8 Any Illness
1377,1,d5,f6,e9 Flu
1378,0,c4,d5,f6 Diabetes
1379,-1,b2,f6,e9 Cold
1380,0,b2,d3,c4,f6,a7,e8 HIV
1381,1,c4,f6,a7,e9 SARS
1382,0,d3,c4,a7 Pancreatitis
1383,2,b2,f6,a7,e8 Any Illness
1384,1,d5,f6,e9 Flu
1385,0,c4,d5,f6 Diabetes
1386,-1,b2,f6,e9 Cold
1387,0,b2,d3,c4,f6,a7,e8 HIV
1388,1,c4,f6,a7,e9 SARS
1389,0,d3,c4,a7 Pancreatitis
1390,2,b2,f6,a7,e8 Any Illness
1391,1,d5,f6,e9 Flu
1392,0,c4,d5,f6 Diabetes
1393,-1,b2,f6,e9 Cold
1394,0,b2,d3,c4,f6,a7,e8 HIV
1395,1,c4,f6,a7,e9 SARS
1396,0,d3,c4,a7 Pancreatitis
1397,2,b2,f6,a7,e8 Any Illness
1398,1,d5,f6,e9 Flu
```

图 6.22　SAGTD 算法结果数据文件截图

```
1373 0    HIV          <b2, d3, c4, f6, a7, e8, >
1374 1    SARS         <c4, f6, a7, e9, >
1375 0    Pancreatitis  <d3, c4, a7, >
1376 2    Any Illness  <b2, f6, a7, e8, >
1377 1    Flu          <d5, f6, e9, >
1378 0    Diabetes     <c4, d5, f6, >
1379 -1   Cold         <b2, f6, e9, >
1380 0    HIV          <b2, d3, c4, f6, a7, e8, >
1381 1    SARS         <c4, f6, a7, e9, >
1382 0    Pancreatitis  <d3, c4, a7, >
1383 2    Any Illness  <b2, f6, a7, e8, >
1384 1    Flu          <d5, f6, e9, >
1385 0    Diabetes     <c4, d5, f6, >
1386 -1   Cold         <b2, f6, e9, >
1387 0    HIV          <b2, d3, c4, f6, a7, e8, >
1388 1    SARS         <c4, f6, a7, e9, >
1389 0    Pancreatitis  <d3, c4, a7, >
1390 2    Any Illness  <b2, f6, a7, e8, >
1391 1    Flu          <d5, f6, e9, >
1392 0    Diabetes     <c4, d5, f6, >
1393 -1   Cold         <b2, f6, e9, >
1394 0    HIV          <b2, d3, c4, f6, a7, e8, >
1395 1    SARS         <c4, f6, a7, e9, >
1396 0    Pancreatitis  <d3, c4, a7, >
1397 2    Any Illness  <b2, f6, a7, e8, >
1398 1    Flu          <d5, f6, e9, >
1399 0    Diabetes     <c4, d5, f6, >
1400 -1   Cold         <b2, f6, e9, >
Process 1400 line data, sagtd use 152.793000 seconds
请按任意键继续. . .
```

图 6.23　SAGTD 算法结果截图

6.5.2 性能分析

数据表在泛化后和轨迹压缩后的性能损失可以定量的表示出来。

给定一个匿名处理后的数据库$T^{\mathcal{G}}$，一棵敏感属性树，那么某$r_i \in T^{\mathcal{G}}$在 SA 泛化后的敏感信息损失量就被定义为公式（6.5）的表达式的值。

$$\mathcal{I}_s(r_i) = \frac{|\Omega(r_i)| - 1}{|\mathcal{L}|}, \tag{6.4}$$

其中$\Omega(r_i) = \ell(\eta(r_i))$是$r_i$的敏感属性泛化后的值，$\mathcal{L} = \ell(\upsilon_0(\Gamma))$是敏感属性树的根节点。

计算被删去的移动点的数量是一种通用的测量方法，它可以用于测量匿名轨迹数据库的数据挖掘的有效性。

性能分析还包括计算轨迹压缩的信息损失量，给定轨迹数据库$T^{\mathcal{G}}$，让$o: T^{\mathcal{G}} \to T$称为一个映射$T^{\mathcal{G}}$内的轨迹数据行和T内的轨迹数据行的函数，任何一行$r_i \in T^{\mathcal{G}}$的信息损失量都可以通过$\mathcal{I}_\tau(r_i)$来标记。并被定义成公式（6.5）中的形式。

$$\mathcal{I}_\tau(r_i) = \frac{|\tau(o(r_i))| - |\tau(r_i)|}{|\tau(o(r_i))|}, \tag{6.4}$$

平均信息的统计如表 6.3 和表 6.4 所示，结果数据表部分截图如图 6.24 和图 6.25 所示，可以看到结果集中 Any Illness 对应行的 b2，e8 点已被删去，算法共用时约两分钟。

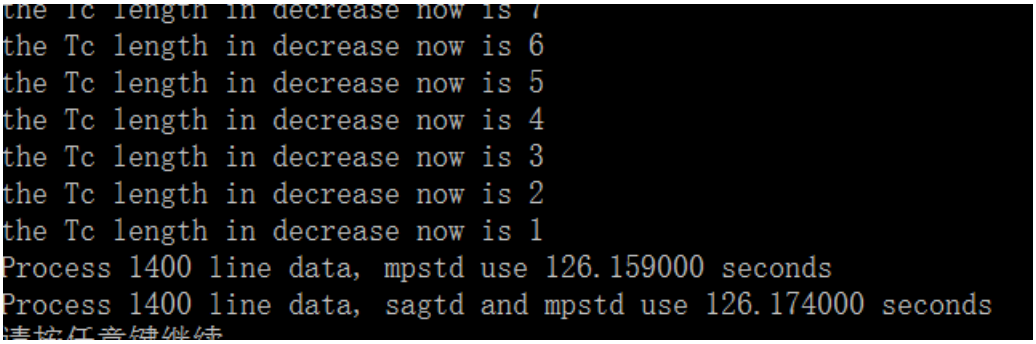

图 6.24　MPSTD 算法结果截图

```
1372,-1,b2,f6,e9 Cold
1373,0,b2,d3,c4,f6,a7,e8 HIV
1374,1,c4,f6,a7,e9 SARS
1375,0,d3,c4,a7 Pancreatitis
1376,2,f6,a7 Any Illness
1377,1,d5,f6,e9 Flu
1378,0,c4,d5,f6 Diabetes
1379,-1,b2,f6,e9 Cold
1380,0,b2,d3,c4,f6,a7,e8 HIV
1381,1,c4,f6,a7,e9 SARS
1382,0,d3,c4,a7 Pancreatitis
1383,2,f6,a7 Any Illness
1384,1,d5,f6,e9 Flu
1385,0,c4,d5,f6 Diabetes
1386,-1,b2,f6,e9 Cold
1387,0,b2,d3,c4,f6,a7,e8 HIV
1388,1,c4,f6,a7,e9 SARS
1389,0,d3,c4,a7 Pancreatitis
1390,2,f6,a7 Any Illness
1391,1,d5,f6,e9 Flu
1392,0,c4,d5,f6 Diabetes
1393,-1,b2,f6,e9 Cold
1394,0,b2,d3,c4,f6,a7,e8 HIV
1395,1,c4,f6,a7,e9 SARS
1396,0,d3,c4,a7 Pancreatitis
1397,2,f6,a7 Any Illness
1398,1,d5,f6,e9 Flu
1399,0,c4,d5,f6 Diabetes
1400,-1,b2,f6,e9 Cold
```

图 6.25　MPSTD 算法结果文件截图

表 6.3　最大泛化深度为 2 时的敏感属性信息损失分数

Privacy level	$\boldsymbol{\delta} = 2$			$\boldsymbol{\delta} = 3$		
	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$
Low	74.8666	0.0227	0.0150	81.6327	1.7729	0.0155
Medium	99.5200	20.8296	0.0561	99.6832	28.8251	1.4126
High	99.8946	40.1283	0.8023	99.9219	43.4581	5.9018

表 6.4　最大泛化深度为 2 时的轨迹信息损失分数

Privacy level	$\boldsymbol{\delta} = 2$			$\boldsymbol{\delta} = 3$		
	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$
Low	79.2527	0.2990	0.0264	87.0525	0.7323	0.0506
Medium	99.4738	21.9078	0.4983	99.1222	23.8958	0.7405
High	99.9863	39.8781	2.3720	99.9112	41.0811	3.2008

结　论

由于移动设备的快速发展和基于位置的服务的广泛使用，轨迹数据越来越受欢迎。它们可能与敏感属性相关联，如疾病，工作和收入。因此，轨迹数据的不当发布可能会使位置被易于监视和跟踪的移动对象的隐私处于危险之中。

大多数以前的对轨迹数据发布隐私保护的研究工作对所有移动对象考虑的都是默认他们具有相同的隐私保护需求。结果是，一些隐私要求较高的移动物体可能会被提供低隐私保护，反之亦然，最终导致信息丢失和披露风险增加。我们通过关注个性化隐私的概念来解决这个问题，并提出了 PPTD，一种维护轨迹数据发布中的隐私的新方法，其结合敏感属性泛化和轨迹局部抑制，以实现数据效用和数据隐私之间的平衡，满足移动物体的隐私要求。

PPTD 不仅能够在轨迹数据发布中提供个性化的隐私保护，而且能抵御所有三种身份链接攻击，属性链接攻击和相似攻击。在轨迹数据发布中防止这些攻击至关重要，因为越来越多的轨迹数据挖掘任务将采用轨迹数据和敏感属性。

我们使用两个轨迹数据集，即 City80K 和 Metro100K，并将数据集中的每个轨迹数据记录随机分配到五个隐私级别（隐私，低，中，高或非常高）之一。我们还生成了一个具有 108 个叶节点的深度为 6 的分类树，用于敏感属性疾病。接下来我们根据不同参数值对敏感属性信息丢失，轨迹信息丢失，披露风险，亲和力系数和查询误差进行评估。实验结果表明，隐私级别极高的轨迹数据记录信息损失最大，泄露风险最小。此外，具有较高隐私级别的轨迹数据记录的匿名化显着降低了隐私级别较低的人员的披露风险。我们终于考虑到与 KCL-Global 和 KCL-Local 相等的条件，并提出了一种称为 KCL-PPTD 的 PPTD 的新变体。应该注意的是，KCL-Global，KCL-Local 和 KCL-PPTD 不能抵抗相似性攻击。实验结果表明，KCL-PPTD 信息丢失率低。这是因为它仅从关键轨迹数据记录中消除了关键移动点，而 KCL-Global 和 KCL-Local 除了关键轨迹数据记录外，还可以消除非关键轨迹数据记录中的关键移动点。

在本文中，我们假设给定的轨迹数据库只包含一个敏感属性。在未来的工作中，我们有兴趣扩展具有多个敏感属性的轨迹数据库的 PPTD。这使得 PPTD 可以应用于移动对象的更逼真和复杂的场景。

参 考 文 献

[1] E. Kaplan, T.B. Pedersen, E. Sava ¸s, Y. Saygın, Discovering private trajectories using background information, Data Knowl. Eng. 69 (7) (2010) 723–736, doi:10.1016/j.datak.2010.02.008

[2] B.C.M. Fung, K. Wang, R. Chen, P.S. Yu, Privacy-preserving data publishing: A survey of recent developments, ACM Comput. Surv. 42 (4) (2010) 1–53, doi:10.1145/1749603.1749605.

[3] A.H.M. Sarowar Sattar, J. Li, X. Ding, J. Liu, M. Vincent, A general framework for privacy preserving data publishing, Knowl. Based Syst. 54 (2013) 276–287, doi:10.1016/j.knosys.2013.09.022.

[4] B.C.M. Fung, M. Cao, B.C. Desai, H. Xu, Privacy protection for RFID data, in: Proceedings of the 2009 ACM Symposium on Applied Computing, in: SAC ’09, ACM, New York, NY, USA, 2009, pp. 1528–1535, doi:10.1145/1529282.1529626.

[5] N. Mohammed, B.C.M. Fung, M. Debbabi, Preserving Privacy and Utility in RFID Data Publishing, Technical Report. 6850, Concordia University, Montreal, QC, Canada, 2010.

[6] R. Chen, B.C.M. Fung, N. Mohammed, B.C. Desai, K. Wang, Privacy-preserving trajectory data publishing by local suppression, Inf. Sci. 231 (2013) 83–97, doi:10. 1016/j.ins.2011.07.035.

[7] X. Xiao, Y. Tao, Personalized privacy preservation, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, in: SIGMOD ’06, ACM, New York, NY, USA, 2006, pp. 229–240, doi:10.1145/1142473.1142500.

[8] E. Ghasemi Komishani, M. Abadi, A generalization-based approach for personalized privacy preservation in trajectory data publishing, in: Proceedings of the 2012 6th International Symposium on Telecommunications, in: IST ’12, 2012, pp. 1129–1135, doi:10.1109/ISTEL.2012.6483156.

[9] K.G. Shin, X. Ju, Z. Chen, X. Hu, Privacy protection for users of location-based services, IEEE Wirel. Commun. 19 (1) (2012) 30–39, doi:10.1109/MWC.2012.6155874.

[10] X. Pan, J. Xu, X. Meng, Protecting location privacy against location-dependent attacks in mobile services, IEEE Trans. Knowl. Data Eng. 24 (8) (2012) 1506–1519, doi:10.1109/TKDE.2011.105.

附录 A　PPTD 算法程序

```
database * SAGTD(database * originDB, database *db, treeNode* root, int MaxSetNum,float PbThreshold,int maxDepth,
                 int breakFlag, int (*breakFun)(database *, database *, database *, float, int, trackRow *)){
    // 1
    trackSet * AResult = str_main(db,MaxSetNum);

    // 2 - 8
    int i,j,k,l;
    int flag;    // 守护结点是否在最上层
    database   * matched, * Bj, * Cj, *Ts;
    database * temp, *db_free_temp_1, *db_free_temp_2;
    database   * CjCopy = initDb();          // 用于断点逐步打印
    matched = initDb();
    Bj = initDb();
    Cj = initDb();
    trackRow * background;
    int c_len = 0;       // Cj 的长度

    for(i=0;i<AResult->count;i++){
        background = AResult->trackCollection[i];
        flag = 1;
        c_len=0;

        freeDb(Bj);
        freeDb(Cj);
        freeDb(matched);

        Bj = initDb();
        Cj = initDb();
        matched = matchRowByTrajectory(db,background->tracks,background->count);

        for(j=0;j<row_count;j++){
            flag = 1;                    // 查找前置 1
```

```
            if(matched[j] == NULL)
                continue;
            for(k=0;k<row_count;k++){
                if(matched[k] == NULL)
                    continue;
                // j 如果被 k 的所覆盖
                if(nodeAisCoveredByNodeB(
                            root,
                            getGuardingNodeByRow(originDB,root,matched[j]->id),
                            getGuardingNodeByRow(originDB,root,matched[k]->id)
                ){
                    flag = 0;
                }
            }

            if(flag){
                insertRow(
                            Bj,
                            matched[j]->id,
                            matched[j]->p_level,
                            matched[j]->trajectory,
                            matched[j]->trajectoryCount,
                            matched[j]->disease
                );
            }
        }
//          traverseDb(Bj,printRow);

        for(l=0;l<row_count;l++){
            if(Bj[l] == NULL)
                continue;
            // 如果泄露概率比 P 大
            if(caculateBreachProbability(
                        originDB,
                        db,
```

```
                        root,
                        Bj[l]->id,
                        background->tracks,
                        background->count
                ) > PbThreshold){


insertRow(Cj,Bj[l]->id,Bj[l]->p_level,Bj[l]->trajectory,Bj[l]->trajectoryCount,Bj[l]->disease)
;

                }
            }

            // 5-7
            if(getLengthOfDB(Cj) != 0){
                if(breakFlag){
                    freeDb(CjCopy);
                    CjCopy = initDb();
                    CjCopy = DBUnion(CjCopy,Cj);
                }
                // 6
 //                 traverseDb(Cj,printRow);
                temp = sag(originDB, db,root,background,Cj,maxDepth,PbThreshold);
// Cj 已变
 //                 traverseDb(temp,printRow);

                db_free_temp_1 = db;
                db = DBSub(db,Cj);

                db_free_temp_2 = db;
                db = DBUnion(db,temp);

                freeDb(db_free_temp_1);
                freeDb(db_free_temp_2);
                freeDb(temp);
                temp = NULL;
```

```
            }

            // 根据标识符判断是否打印步骤，用户可以随时退出
            if(breakFlag){
                breakFlag = breakFun(Bj,CjCopy,db,PbThreshold,maxDepth,background);
            }

        }

        Ts = db;
        return Ts;

    }
```

致　　谢

写至致谢，意味着大三生活即将结束，即使有万般不舍，但是这一天也终将到来。回首三年的学习时光，我学习到了很多专业知识，也认识了很多优秀的老师和热情的同学，他们给了我许多帮助，谨以最朴实的话语来表达我的感恩之情。

首先要衷心感谢我小学期课程的王新雨学长。碰到了一些概念上的问题，是在学长的帮助下解决的。在小学期课程的这段时间，学长和老师像一位知心朋友一样对我的工作做出鼓励和引导，从实验设计到文档撰写，仍然能够为我指点迷津，正是学长和老师的悉心指导，我的作业才能够顺利完成，谢谢老师和学长。

最后我要感谢学校，能够为我提供一个良好的学习环境和氛围，为我度过这匆匆三年起到了重要作用。谢谢大家，我会在未来的学习生活中更加努力。