移动环境下的在线轨迹匹配和压缩方法及实现

重庆大学硕士学位论文

（学术学位）

学生姓名：丁琰

指导教师：陈超　教授

专　　业：计算机科学与技术

学科门类：工　学

重庆大学计算机学院

二〇一九年四月

Mobile Trajectory Data: Map-matching, Compression and Their Implementation

A Thesis Submitted to Chongqing University

in Partial Fulfillment of the Requirement for the

Degree of Master of Engineering

By

Yan Ding

Supervised by Prof. Chao Chen

Specialty：Computer Science and Technology

College of Computer Science of Chongqing University,

Chongqing, China

April, 2019

摘　　要

交通大数据是实现城市智慧交通服务的数据基石，时空轨迹是其中一种重要类型。近年来，得益于移动互联网和定位等技术的广泛普及和成熟，车辆的时空轨迹通过车载 GPS 设备很容易地被实时采集、传输并保存至云端数据中心。然而“海量”时空轨迹数据也一定程度为城市智慧交通化带来了挑战，如严重耗费带宽资源、极大占用存储空间、阻碍数据可视化与挖掘等。在线轨迹压缩技术是一种解决以上问题的有效方法。为减少移动时空轨迹，现有系统或方法只是简单地降低 GPS 设备的采样频率，但这种做法无疑增加了轨迹的“稀疏性”和“不确定性”。更糟糕的是，城市路网的复杂特性以及 GPS 设备的定位误差使推断车辆行驶轨迹变得更加困难。为此，本文设计了一套轨迹匹配与压缩的系统框架。具体而言：首先，基于 SD-Matching (Spatial-Directional Matching)算法，本文将轨迹映射在广泛易得的路网以克服定位误差及实现轨迹语义丰富化；然后，基于 HCC (Heading-Change-Compression) 算法和 DVAT (Distance-Velocity-Acceleration-Timestamp)算法，本文在空间和时间两个维度分别实现移动轨迹的压缩，以减少传输和存储移动空间轨迹数据带来的代价。移动环境对在线轨迹匹配和压缩系统的计算能力、时间延迟和量级大小提出了严格要求，这意味着计算能力有限的 GPS 设备无法负担计算任务（也即轨迹匹配与压缩）。受移动边缘计算的启发，本文创新地将繁重的计算任务（也即轨迹匹配与压缩）迁移至 GPS 设备附近的智能设备，如司机的智能手机。

概括而言，本文主要成果包括以下几个方面：

① 在轨迹匹配上，本文探索车辆航向的可用性，并在地图匹配的各个阶段巧妙地利用该信息，以提高轨迹匹配的效率与质量。

② 在空间轨迹压缩上，本文探索交叉路口车辆航向变化在压缩轨迹的可用性，并基于此设计高质量的压缩算法，以更好地平衡压缩率和效率。

③ 在时间轨迹压缩上，本文提出一种全新的轨迹表示，它包含三个部分，即距离序列（D）、加速度和瞬时速度序列（AV）和时间序列（T）。针对它们分别设计了三种不同的压缩器。另外，我们是保存并压缩时变速度信息的先驱，该信息对理解驾驶风格和掌握城市交通状态至关重要。

④ 本文将算法部署在真实移动环境。受边缘计算的启发，本文创新地将繁重计算任务从计算能力弱的 GPS 设备迁移到附近手机端。这一举措不仅能解决算力不足，还能释放对 GPS 设备采样频率的限制。

⑤ 无论是轨迹匹配还是压缩，或真实移动环境下的系统，我们都利用真实数据集对它们进行了广泛评估，实验结果证明它们在性能（如准确度、压缩率和效率等）上均优于同类算法或系统。

关键词：时空轨迹，轨迹压缩，地图匹配，边缘计算，移动环境

ABSTRACT

Big traffic data is an essential ingredient to smart urban transportation, among which the GPS trajectory data is a typical representative. A huge volume of trajectory data of moving vehicles is being accumulated and collected at data centers with unprecedented speed and scale, as a result of the proliferation of GPS-enabled devices and the maturity of Mobile Internet during recent years. However, such massive trajectory data also causes a series of issues to build and realize the smart cities, including heavy communication overhead, storing, and computing and so on. It is well-recognized that online trajectory compression is one of the promising methods to alleviate the above-mentioned issues. At the current stage, existing systems usually achieve data compression by lowering the sampling rate of moving vehicles. Such a naive solution no doubt aggravates the issues of "sparseness" and "uncertainty" in the collected trajectory. To make matter worse, the dense road network and the positioning error of GPS devices will make the inference of true driving routes more challenging. In this thesis, we take a more sophisticated approach to tackle with, i.e., we develop a novel trajectory mapping and compression system. More specifically, we propose a map matcher, namely SD-Matching (Spatial-Directional Matching), which is proved to 1) have the potential to reduce the side effects caused by the location error significantly; 2) generate a more natural and semantical trajectory representation. Next, to save the communication and storing overhead, the trajectory in the spatial and temporal dimensions is compressed based on HCC (Heading-Changes-Compression) and DVAT (Distance-Velocity-Acceleration-Timestamp) algorithms, respectively. Such computation in the system is usually resource-hungry and GPS devices themselves just cannot afford these burdensome tasks (i.e., trajectory mapping and compression). In addition, there are some requirements that the system implemented in the mobile environment should satisfy, i.e., computing capability, low latency, and light weight. Motivated by mobile edge computing, the burdensome computation is offloaded to nearby smart devices, such as drivers' mobile phones.

The main contributions are summarized as follow.

① In trajectory mapping, we first explore the usability of vehicles' heading direction, and then fully leverage this information in a smart way to improve the efficiency and accuracy of SD-Matching algorithm.

② In spatial trajectory compression, we first observe the heading changes at intersections. Based on the obtained observation, we then develop a high-quality compression algorithm, i.e., HCC algorithm, which can achieve a good balance between efficiency and compression ratio.

③ In temporal trajectory compression, we propose a new trajectory representation consisting of three parts, i.e., distance sequence (D), acceleration and instantaneous velocity sequence (AV) and timestamp sequence (T), and three compressors are devised wisely to compress each part separately. To the best of our knowledge, we are among pioneers to compress the time-varying velocity information, which is essential for driving style classification and urban traffic condition detection.

④ In system implementation, inspired by the principles of mobile edge computing, burdensome tasks are offloaded from GPS devices with limited computing capability to nearby drivers' mobile phones. Compared to the traditional method which simply collects vehicles' locations less frequently at the side of GPS devices without compression, it is expected the proposed system is more informative and robust to GPS noises.

⑤ The extensive evaluation of trajectory matching, trajectory compression, and the system are carried out with real-world datasets. According to the experimental results, we can make a safe conclusion that they outperform similar algorithms and systems in terms of accuracy, compression ratio, and efficiency.

Keywords: Spatial-temporal Trajectory, Trajectory Compression, Map Matching, Mobile Edge Computing, Mobile Environment.

目　录

1 绪论

1.1 引言

城市化进程赋予人们现代化的生活[1-4]，但同时也带来了许多问题与挑战，如交通拥挤、尾气污染、能耗增加和出行体验差等城市病[5,6]。近年来，各国政府积极倡导建设智慧城市来解决这些问题[3]。城市建设，交通先行[7-10]。交通是城市发展的命脉，智慧交通是智慧城市在交通领域的具体体现，是智慧城市极其重要的组成部分[1,2,4]。智慧交通使城市交通系统具备泛在感知、互联、分析、预测、控制等能力，能够有效缓解交通拥堵、减少尾气排放、降低出行能耗和改善出行体验[5,6,11-14]。但这一切都离不开对城市交通大数据的获取和计算，车辆的时空轨迹数据是其中一种重要类型。近年来，得益于移动互联网、GPS 定位等技术的发展与成熟，移动物体的位置、时间、速度和航向等时空轨迹数据被大规模且实时地采集并传输至云端数据中心。车辆就是最典型的例子，但与其他移动物体不同，车辆的行驶轨迹受到路网的空间限制。车辆的时空轨迹数据是众多基于地理位置应用的数据基石，如路况实时检测、空气质量预测、个性化（省油/风景优美）驾驶路线推荐等服务。通常情况下，车载 GPS 设备以固定频率采集车辆的移动信息，然后通过第三方带宽（如 GPRS，3G，4G）将数据实时地传输并存储至云端数据中心[15,16]。但海量时空轨迹数据却引发了在通信、存储和计算等方面的一系列灾难，如带宽资源被耗费、存储空间被占用、数据可视化及挖掘任务被阻碍等[17-20]。更糟糕的是，未来不断增长的公共交通和私家车辆会部署 GPS 设备，并且持续上传数据，问题会更加严峻，因此需要迫切解决传输和存储海量时空轨迹数据引发的一系列问题。

为了解决以上问题，在时空轨迹数据发送到云端数据中心之前就将其在线压缩是一种有效降低通信和存储开销的办法[20-22]。现有系统通常采用一种简单原始的方法，即调低车载 GPS 设备的采样频率来减少需要传输和保存的数据量，但该做法无疑加剧了轨迹数据的“稀疏性”和“不确定性”[23]。因为移动车辆常被用作城市交通“探针”来实时监测交通状况，所以管理者必须实时掌握车辆的真实位置[5,19,24,25]，然而过于稀疏的轨迹点将导致无法推断出车辆真实的行驶路径。更糟糕的是，车载 GPS 设备的噪声（即设备的定位误差）和复杂的城市路网环境（如高架立交桥等）会使车辆行驶路径的推断变得更加困难。城市路网数据广泛且易得，车辆轨迹通常被映射匹配到道路上，即地图匹配[23,26-28]，也称为轨迹匹配。这种做法不仅能克服 GPS 设备的定位误差，而且嵌入路网的轨迹数据也会蕴含更丰富的语义信息，这已经被证明利于轨迹压缩，因此基于地图匹配的轨迹压缩通常优

于其他同类算法。但是，轨迹匹配任务往往要消耗大量的计算资源[29]，而车载GPS设备计算能力非常有限，它们的RAM通常只有4KB，很明显车载GPS设备并不能承担繁重的计算任务[30]。因此，本文开发了一种运行在移动环境下轻量级且高效的地图匹配算法，算法细节请参阅第2章。

在线压缩是降低轨迹数据传输与存储开销一种常见方法，其本质是追求更简洁的轨迹表达[31]。根据压缩后轨迹中元素物理含义是否变化，轨迹压缩工作可大致分为两类，第一类是通过丢弃原始轨迹中冗余数据来获得数据约简，另一类是使用编码原有数据元素来压缩数据，具体细节介绍在相关工作中。最近工作中，车辆的时空轨迹数据首先被分离为空间轨迹和时间轨迹，然后应用不同的压缩算法对它们分别压缩。文献[16]已经通过理论推导证明了这种做法能够获得更高的压缩率。背后原理是通过时空分离操作增加了轨迹的信息熵值，众所周知信息熵越高，意味着重复数据越多，越利于轨迹的压缩。空间轨迹通常被表示为车辆在路网中的行驶路径，即一系列首尾相连的边e_i。这种表达是冗余的，可以对其进行进一步压缩。更详细来说，空间轨迹压缩算法移走原始轨迹中一些冗余边，丢弃掉的这些边可以被剩余边进行推断和复原。其中最有代表性的工作是PRESS算法[19]，它利用最短路径来完成空间轨迹压缩。对于一条给定的从边e_1到边e_2的路径，若它是从边e_1到边e_2的最短路径，那么此路径就被表示为二元组 e_1,e_2 ，这是因为驾驶员倾向于选择起始点和目的地之间的最短路径，因此应用该方法可以移除空间轨迹中大量的边。但PRESS算法需要计算并保存路网中所有最短路径，在移动环境下这种操作是不现实的。举个例子，北京市路网中所有最短路径占了约10GB的存储空间。因此，本文设计了一种低成本的空间轨迹压缩算法，算法细节请参阅第3章。

现有时间维度上的轨迹压缩通常聚焦于车辆的位置和时间信息，直接忽略了车辆的时变速度，而该信息在了解驾驶风格、探测路况等方面发挥着至关重要的作用。并且，由于瞬时速度是基于多普勒效应计算出来的，即使能轻易地推断出两个连续GPS点之间的平均速度，也不能将其视为冗余信息，因此在时间轨迹压缩算法中，本文对时变速度单独进行处理。具体说来，本文提出了一种更精确更完整时间维度上的全新轨迹表示，它包括三个部分，即距离序列（D）、加速度和瞬时速度序列（AV）以及时间序列（T）。针对这三个部分，本文设计了三套全新的压缩算法。压缩后的数据可以有效地推断出车辆的实时位置，便于监控、跟踪、调度等任务。另外，还可以支持多种查询，包括地点查询(即车辆在某时刻的位置)和时间查询(即车辆在某位置的时间)。此外，由于我们设计的轨迹表示还能处理时变速度信息，因此还可以支持在给定时间间隔查询速度方差和查询感兴趣道路

的交通状况。此外，还可以支持更多潜在的时空数据挖掘应用，如隐式嵌入到时变速度信息[11,32]中驾驶员的驾驶风格或爱好。算法细节请参阅第 4 章。

移动环境下在线轨迹匹配和压缩对系统的计算能力、时间延迟和量级提出了更严格的要求。详细来说，在线地图匹配需要消耗大量的计算资源，因此需要一个强大的计算平台。通常情况下，GPS 设备因为有限的计算能力并不合适作为移动环境下的任务计算平台，因此系统必须拥有足够的计算资源；为了降低轨迹的“稀疏性”和“不确定性”，车载 GPS 设备最好以高频率实时采集轨迹信息。因为轨迹系统的最大时间延迟不应该超过相邻 GPS 点的时间间隔，所以系统必须及时处理（匹配和压缩）车辆轨迹数据；在移动环境下，用于嵌入式计算的存储空间是有限的，因此系统必须是轻量级的。受启发于移动边缘计算，本文创新性地将繁重的轨迹匹配与压缩任务迁移至 GPS 设备附近的手机端，利于手机闲置的充足计算资源来完成计算任务。除了能高效地完成任务，本文设计的系统还具有低功耗和轻量级等优点。系统实现请参阅第 5 章。

1.2 国内外研究现状

本文从轨迹匹配和轨迹压缩两个方面来讨论国内外的研究现状。

1.2.1 轨迹匹配的国内外研究现状

轨迹匹配也称为地图匹配，它一般包含三个阶段。第一个阶段是寻找 GPS 点真正的位置，第二个阶段是推断车辆可能的行驶路径，最后一个阶段是筛选出车辆真正的行驶路径。Quddus 等人写了一篇优秀的关于地图匹配的文献综述[28]，感兴趣的读者可以阅读以获得全面的地图匹配概述。早期的匹配工作通常只是为了找到 GPS 轨迹点真正的位置，此类工作只能克服 GPS 设备的测量误差[33]。为了获得真实位置，我们基于 GPS 点在路网中的观察位置，并结合 GPS 设备的测量误差范围，首先确定若干条路网中的候选边。很容易知道，这些候选边都有成为 GPS 点真正匹配边的可能性。地理信息（如“点到曲线”的距离或者“点到形点”的距离[28]）被利用起计算概率，概率的大小衡量可能性的高低[34-38]。通常，GPS 点真正的匹配边可以通过将该点投射在概率最高的候选边上来获得。在这类工作中，最关键的就是概率计算。以前的研究者就曾利用车辆方向信息来提高概率计算的准确度，车辆方向信息被定义为前一个 GPS 点指向后一个的连线方向，其物理意义是车辆的移动方向。但如果 GPS 采样点过于稀疏，车辆的方向信息可能会太粗糙而无法使用。更糟糕的是，在线移动环境中，车辆下一个 GPS 点的位置无法获知，换句话说此时车辆方向信息无法计算，因此车辆方向信息无法在在线地图匹配算法中使用。幸运地是，现有嵌入车辆中的 GPS 设备通常可以测量瞬时的车辆方向（也称为车辆航向），Velaga 等人已经利用该信息来更准确地捕捉 GPS 点的

真实位置[39]。受此启发，我们也尝试研究如何使用收集到的车辆航向来提高概率计算的性能（即准确度）。

最近关于地图匹配的工作主要集中在通过重新构建车辆行进路径来填充采样GPS 点之间的距离间隙。两个连续 GPS 点所在的两条候选匹配边通常是断开的，它们之间的间隙甚至可能很大。这方面的研究工作主要是来解决轨迹匹配中的“稀疏性”和“不确定性”问题[23]。为了准确地推断车辆行进路径，研究人员使用了诸如路网拓扑结构（例如链路连通性和邻接性）、道路属性（例如车道数量和道路速度限制）和运动定律之类的附加信息[29,40,41]。此外，研究者还提出了一些基于概率且更先进的算法（例如基于隐马尔可夫模型）来提高地图匹配的质量[24,27,42-45]。为了推断车辆的行进路径，我们首先应该找到每对连续 GPS 点之间的若干候选路径，然后通过相邻 GPS 点和道路网络拓扑结构全面地对候选路径进行筛选。当推断每对连续 GPS 点的行进路径时，Dijkstra 算法和 A*算法是最常用的路径搜索算法[44]。由于连续 GPS 点的每组候选边集合中两两元素之间都需要做最短路径计算，因此路径搜索花费的时间成本非常高[46]。在第 2 章，我们基于收集到 GPS 设备的车辆航向信息，进一步地提出了一种启发式路径搜索算法，来探索车辆航向在地图匹配的路径推断和筛选中的可用性，算法目标是突破最短路径搜索中的运行效率瓶颈。

1.2.2 轨迹压缩的国内外研究现状

已有许多数据压缩技术被提出并广泛应用于处理不同类型的数据，如图像、音频和轨迹数据。感兴趣的读者可以文献[47]以获得全面的概述。这里，我们主要回顾轨迹数据的压缩技术，根据工作原理，轨迹压缩算法可以大致分为两类。第一类是通过丢弃轨迹表式中一些不必要的原始数据元素，来实现轨迹压缩，而第二类是使用新的简洁的数据元素替换（编码）轨迹表示中原始的数据元素。

“丢弃数据”来实现轨迹压缩 该类压缩工作的原理是压缩轨迹中剩余的数据元素，可以很好地甚至完全地恢复被丢弃的数据元素。这类工作的代表性方法有Douglas-Peucker (DP)算法及其变体和 PRESS 算法等[18,48-53]。例如，用 GPS 点序列表示的原始轨迹可以被简化为包含更少 GPS 点的压缩轨迹。在压缩轨迹中被保留下来的 GPS 点通常被称为“特征点”，两个连续特征点之间的 GPS 点位置可以通过线性插值逼近而得到[16,17,54-56]。例如，文献[51,57]的作者提出了 BQS 算法及变种算法 FBQS，这两种算法通过构造凸壳并丢弃凸壳内的 GPS 点来实现在线数据压缩，算法的时间复杂度是$O\ n^2$，其中n为轨迹片段中 GPS 点的个数。相类似的，OPERB 算法也是一种在线数据压缩算法，但它的效率更高，计算复杂度为$o\ n$。最近一些工作也介绍了压缩过程中的轨迹预测[58]。例如，文献[58]根据历史轨迹数据，使用马尔可夫模型预测未来的轨迹。若实际轨迹与预测轨迹结果不一

致，则保留实际轨迹，否则丢弃。考虑到车辆只能在有限的空间范围内移动(即只能在道路上移动)，空间轨迹往往用边或者节点的序列来表示。我们容易知道，道路网络中的两节点之间可能会有若干条路径，而且每条路径中边的数量是不同的，有些路径中边的数量比其他路径多。基于这种观察，MMTC 算法利用边较少但相似度较高的路径来代替原始路径，因此需要保存的边的数量减少，轨迹得到压缩。路径也可以用一节点序列来表示，其中序列中任意两个连续节点在道路网络中是通过唯一边相连的。某些工作的空间轨迹压缩中，节点序列往往被简化为更短的序列，在新压缩的节点序列中，两个连续节点在道路网络中通常是断开的，但是中间路径可以通过最短或者最频繁的路径推断出。然而，路径的计算要么消耗高时间成本(在线版本)，要么占用极高的存储空间成本(离线版本)，因此此类算法带来的副作用会极大地降低地图匹配算法的性能。CCF 算法探讨了交叉路口在空间轨迹压缩中的可用性，具体来说它使用车辆经过的所有交叉路口的出边来代替匹配轨迹（注“出边”的定义参照第 3 章的定义 7），由于出边的数量远小于原始空间轨迹中边的数量，因此 CCF 算法可以节省大量的存储空间，实现对轨迹的压缩。然而，保留所有出边仍然是多余的，空间轨迹可以进一步被压缩。在本研究中，我们提出了一种充分利用交叉口的车辆航向变化的全新空间轨迹压缩算法（简称 HCC 算法），具体是算法只保留在交叉口航向变化显著的出边。算法背后的原理是，此类型的出边数量只占所有出边的一小部分，因此存储空间可以被极大的降低。此外，PRESS、CCF 和 HCC 算法都是空间无损的。在时间纬度上的轨迹压缩中，时间轨迹通常由 2D 平面中的一系列时间-距离点表示[30]。在用户给定的误差范围下，PRESS 和 CCF 算法根据该点对曲线形状的贡献量决定是否要保留该点。总之，在此类丢弃数据实现轨迹压缩的方法中，数据压缩之前和之后的表示中的每个数据元素都具有相同的物理意义。

“替换数据”来实现轨迹压缩 这类工作的原理是新生成的数据元素占用的空间比原来的数据元素少得多，其中最有代表性的算法是哈夫曼编码[59]。哈夫曼编码的核心原理是使用较短的二进制编码对轨迹表示中出现频率较高的数据元素进行编码，新生成的元素(即二进制编码)占用的存储空间比原先小得多。类似地，TED 算法使用变长二进制代码对时间轨迹中的每个数据元素进行编码[60]。唯一的区别是，它是根据点到所在边头节点的距离对每个数据元素进行二进制编码，读者可以参考[60]了解更多细节。在这两种案例中，在数据压缩前后轨迹表示中的数据元素总数是相同的。另一种代表性的编码技术是 Run-Length-Encode 编码(简称 RLE 算法)[61]。新的数据元素是二元组，分别表示轨迹表示中数据值及其出现频率，在这种情况下，数据压缩后的数据元素总数应该小得多，最终有助于减少数据。但当轨迹表示中数据冗余很少时，RLE 算法会失效。还有一些基于字典的编

码方法，可以有效地压缩包含大量重复数据的轨迹表示[62,63]，新的数据元素是构建字典中相应的索引。轨迹表示也能被语义信息所替代。例如，[64]将空间轨迹转换为道路名称。[65]利用交通网络(如公交、电车和火车线路)的高级语义来替代原始轨迹表示。总之，当使用“替换数据”来实验压缩上，压缩前后轨迹表示中的每个数据元素的物理含义都发生了更改。由于我们的工作同时使用了霍夫曼编码和RLE 算法，所以我们设计的算法属于这一范畴。但与其他工作相比，我们更进一步地构造了一组霍夫曼树(即霍夫曼森林)，这是因为在不同的道路或区域生成的轨迹表示中的原始数据元素值差异非常大且分布也非常不均匀。

1.3 论文的组织结构

本文针对在线轨迹匹配和压缩关键技术进行了研究，为解决匹配和压缩的技术难点，如突破匹配算法效率的瓶颈以及在压缩质量和效率之间取得平衡等问题，我们提出了全新的算法框架。本文共分为六个章节，详细内容安排如下：

第一章是本文的绪论。首先介绍了轨迹匹配和压缩的研究背景，并对国内外的相关研究工作进行了详细的阐述与总结，然后进一步分析研究目标和内容，最后简要介绍了本文的组织结构。

第二章是在线轨迹匹配的研究工作。首先介绍轨迹匹配的研究背景和现状，然后探索车辆航向提高算法准确度和效率的可能性，并设计出高质量的匹配算法SD-Matching，接着通过实施广泛的实验评估，证明 SD-Matching 算法的效率远高于最优秀的同类算法，而且推测出的车辆行驶路径与真实行车路径高度吻合，即准确度高。最后，我们开发了一款半自动标记软件，为实验提供真实行车路径数据。

第三章是空间维度上在线轨迹压缩的研究工作。首先介绍空间轨迹压缩的研究背景以及现状，然后探索交叉路口处车辆航向变化在压缩中的有用性，并设计出高质量的压缩算法 HCC，最后使用真实数据集对其进行性能评估（压缩率和效率），实验结果证明与其他基准算法相比，HCC 算法在压缩率和效率之间取得良好的平衡。

第四章是时间维度上在线轨迹匹配的研究工作。首先介绍时间轨迹匹配的研究背景以及现状，然后提出一种全新的时间轨迹表示，由距离序列（D）、加速度序列和瞬时速度序列（AV）和时间序列（T）组成，接着针对这三个部分分别开发不同的出压缩器，最后通过实验评估，证明了压缩框架良好的性能。

第五章是移动环境下轨迹匹压缩系统 VTracer 的实现。在线移动环境对系统的性能提出了更高的要求，特别是计算效率，我们专门针对算法的效率进行工程

优化，并在真实环境下对系统的性能进行了测试，如匹配准确度、压缩率、内存消耗等。实验结果表明，VTracer 系统有着良好的性能。

第六章是总结与展望。该章对全文研究进行总结，并指明未来的研究重点。

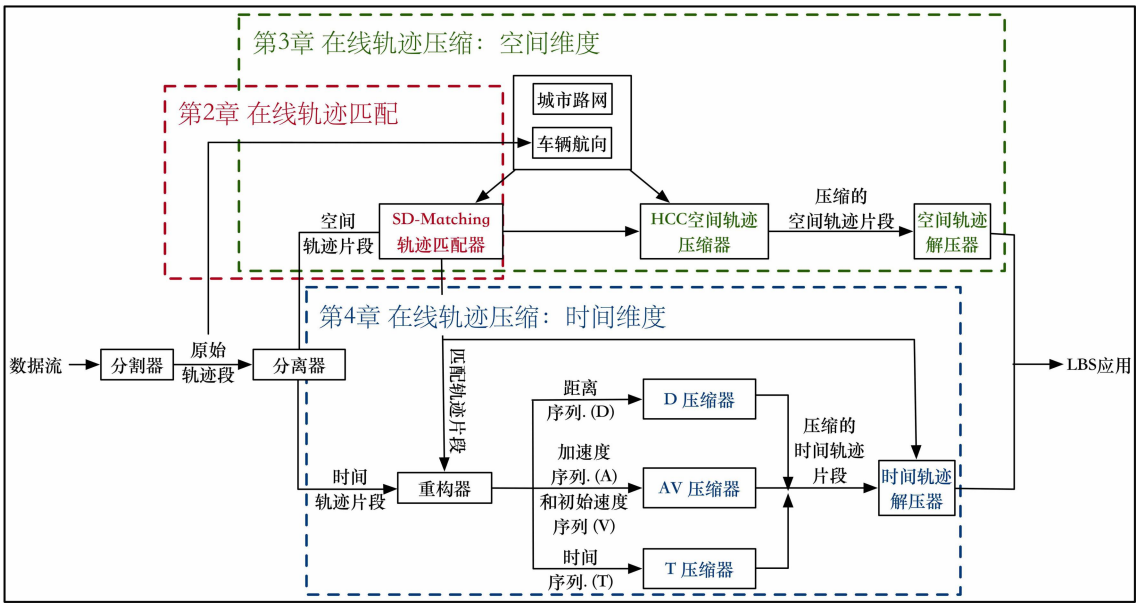


图 1.1 主要研究内容

Figure 1.1 Main Research Content

2　在线轨迹匹配算法

本章研究在线轨迹匹配，特别是克服其效率瓶颈。内容安排如下：2.1 节是本章的前言，介绍了地图匹配算法的研究背景和现状；2.2 节介绍了一些主要概念，并对设计算法 SD-Matching 进行了概述；2.3 节详细阐述了 SD-Matching 算法；2.4 节全面评估了 SD-Matching 算法的性能；2.5 节介绍了一款本文设计的半自动标记软件 **TLabel**，为实验提供真实数据；2.6 节对本文进行了总结。

2.1 前言

近年来，移动 GPS 设备广泛普及，被嵌入车辆中收集用户的移动轨迹信息。车辆的轨迹信息能支持路径规划、交通状况检测等应用[6,11,12,14,66]，是智慧交通服务的数据基石。为了节省能量和传输成本，通常 GPS 设备只间隔地记录车辆位置信息，这导致了轨迹数据的“稀疏性”和“不确定性”。由于设备的测量误差和数字地图的失真，观察到的 GPS 点位置是不准确的，且通常未坐落在道路上。存在误差的 GPS 轨迹数据会降低使用该数据应用的性能。车辆的运动轨迹受限于空间铺设的道路，为了找到 GPS 点的“正确的”位置和车辆在路网中真实的行驶路径，将轨迹点匹配到路网上（即地图匹配）已经成为解决上述问题的有效方法[23,26-28]。最近，车辆常被用作交通“探针”，来监测道路的交通状况。这种任务必须实时地掌握车辆实际行驶道路[5,24,25,67]。因此，在线地图匹配是非常必要的。

地图匹配算法包含三个阶段，即识别 GPS 点的真正位置、推断车辆可能的行驶路径和筛选出真正的行驶路径。Quddus 等人写了一篇关于地图匹配的优秀文献综述[28]。早期的地图匹配工作通常只是为了找到 GPS 轨迹点的匹配边，但此类工作只能克服 GPS 设备的测量误差[33,41]。为了确定给定 GPS 点的真实位置，基于 GPS 点在路网中的观察位置和设备测量误差范围，首先确定一组候选边集合，其中每个候选边都有成为真正匹配边的不同概率。研究者利用地理信息（如‘点到曲线’的距离，或者‘点到形点’的距离[28]）来计算这个概率，GPS 点的真实位置往往可以通过将 GPS 点投射在最高概率的匹配边上来获得。如何准确的计算概率值是此类工作的关键，以前的工作就曾利用 GPS 点的方向信息来提高概率计算的准确度。GPS 点的航向被定义为前后两个 GPS 点的连线方向（也称为车辆的移动方向）[37]。但如果采样点过于稀疏，GPS 点的航向信息可能太粗糙而无法被使用。更糟糕的是，在线环境中，下一个 GPS 轨迹点无法得知，因此该航向信息无法在在线地图匹配中使用。幸运地是，现有的嵌入车辆中的移动 GPS 设备通常可以测量瞬时航向信息，Velaga 等人已利用该信息更准确地找到真实位置[39]。因此，与以前

的工作类似，本文尝试研究如何使用收集的航向信息来提高本研究中概率计算的准确性。

最近关于地图匹配的工作主要集中在通过重建车辆行进路径来填充采样的GPS 点之间的距离间隙。具体来说，两个连续 GPS 点的候选边通常是未连接，而且它们的间隙甚至可能很大，因此研究工作主要解决轨迹存在的“稀疏性”和“不确定性”等问题[23]。为了准确推断路径，研究者使用了诸如道路网拓扑（例如，链路连通性和邻接性）、道路属性（例如，车道数量，速度限制）和运动定律之类的附加信息[29,40,41]。此外，研究者还提出了一些更先进的概率算法（例如，基于隐马尔可夫模型）来提高地图匹配质量[24,27,42-45]。为了推断车辆 GPS 轨迹的真实路径，首先推断每对连续 GPS 点之间的若干行进路径，然后通过相邻 GPS 点和道路网拓扑结构全面地对路径进行筛选。在推断每对连续 GPS 点的路径过程中，研究人员常用 Dijkstra 和 A*算法[44]。由于连续 GPS 点的每组候选边之间都需要计算最短路径，因此时间开销成本非常高[46]。在本章节中，基于收集的航向信息，进一步提出了一种启发式算法来探索航向信息在路径寻找和筛选中的可用性，算法着重于克服最短路径运算中的运行效率瓶颈。

2.2 主要概念和算法概述

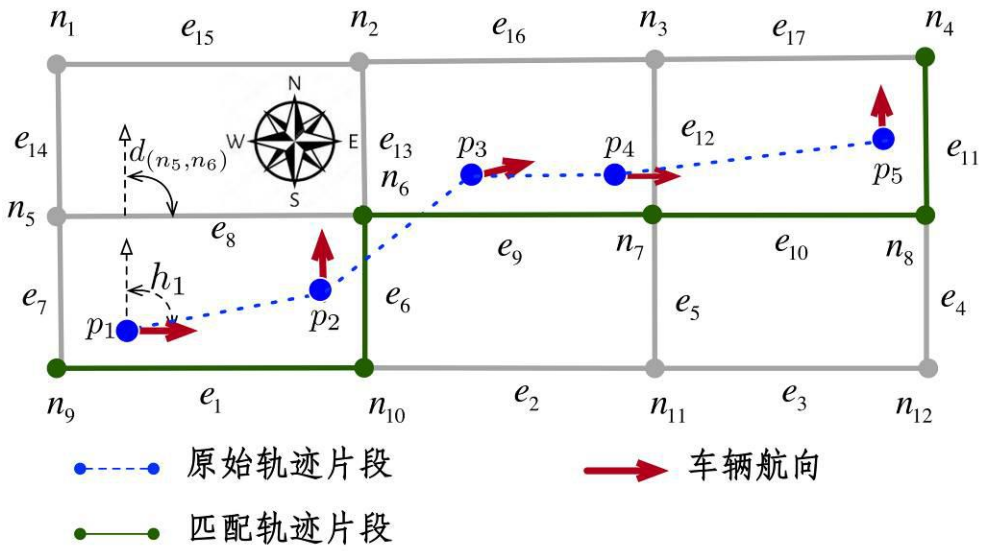


图 2.1 主要概念的例证

Figure 2.1 Illustration of main concepts used in this section

定义 1 （路网） 路网是一个图，记为 $G(N,E)$，它由边集 E 和节点集 N 组成。集合 E 中的元素是有向边 e_i，有向边分别包含一个头节点 n_h 和尾节点 n_t。N 中的元素 n 是包含经纬度坐标的点，来表示车辆的空间位置。

定义 2 （边的方向） 边 e_i 具有两个方向，分别记为 $d(n_h,n_t)$ 和 $d(n_t,n_h)$。$d(n_h,n_t)$ 定义为从节点 n_h 到节点 n_t 的边方向，以正北方向作为基准。边的方向可以通过两个节点的经纬度计算得到。如图 2.1 中的 $d(n_5,n_6)$ 是边 e_8 从头节点 n_5 到尾节点 n_6 的一个方向。

定义 3 （车辆航向） 车辆航向记为 h 是指在采样位置处车辆的车头指向，同样是以正北方向为基准，h（单位是度）的范围是 $[0,360)$。该信息可以直接从原始轨迹中访问得到。例如，图 2.1 所示，车辆在 p_1 点处的车辆航向是 h_1。

定义 4 （GPS 点） GPS 点（p_i）记录了车辆的时空信息，包括时间戳 t_i、地理空间坐标 (lat_i,lon_i)、瞬时速度 v_i 和车辆航向 h_i，记为 p_i $(t_i,lat_i,lon_i,v_i,h_i)$。

定义 5 （原始轨迹片段） 原始轨迹片段 $_i$ 由一系列 GPS 点组成，可以表示为 $_i$ $p_i,p_{i\ 1},\ ,p_{i\ l}$，其中参数 l 控制了片段长度。原始轨迹数据流 T 是由无数的轨迹片段 $_i$ 组成，记为 T $_{1},_{1},$ 。

定义 6 （匹配轨迹片段） 给定一个原始轨迹片段 $_i$，其匹配的轨迹片段 $_m$ 是车辆在路网中真实的行进路径，记为 $_m$ $e_i,e_{i\ 1},\ ,e_n$，其中连续的两个边是相连的。匹配的轨迹流是由无数轨迹匹配片段 $_{mi}$ 组成，记为 T_m $_{m1},_{m2},$ 。

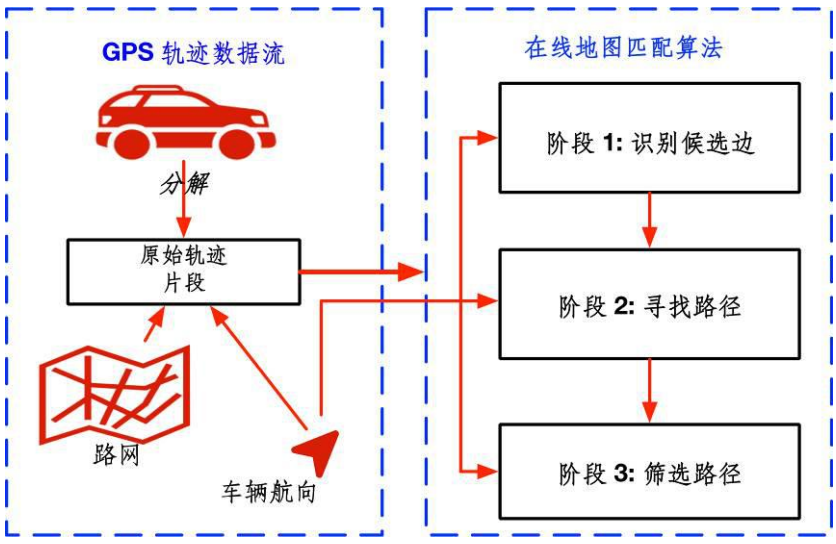


图 2.2 提出的 SD-Matching 算法概述

Figure 2.2 Overview of the proposed SD-Matching algorithm

SD-Matching 算法的概述如图 2.2 所示，其中算法包含了三阶段。当原始的 GPS 轨迹数据流输入算法中时，算法首先将数据流分割成等长的短轨迹片段，片

段是基本的处理数据的单元。给定一个轨迹片段，算法综合地整合了路网和航向信息，通过三个阶段处理轨迹片段。具体来说，在第一个阶段，我们利用空间概率和方向概率计算每个 GPS 点“附近”的边是真正匹配边的可能性高低；在第二个阶段，利用车辆航向来缩小搜索空间并加速路径搜索；在第三阶段，考虑车辆航向和路网拓扑结构，筛选给定原始轨迹片段的真实行进路径。

2.3 SD-Matching 算法

正如算法概述的介绍，我们设计的在线地图匹配 SD-Matching 算法包括了三个阶段，即识别候选边、推断和筛选路径。在本节中，我们将详细介绍这三个阶段。

2.3.1 识别候选边

对于给定的 GPS 点 p_i，我们首先确定半径为 r（例如 100 m）的圆内的一组候选匹配边。注意点 p_i 通常具有若干候选边，尤其是在密集的路网中。为了找到该点的真正正确的匹配边，减少候选边的数量是有必要的。因此对于每个候选边，我们首先利用空间概率和方向概率来计算点 p_i 的侯选边是正确匹配结果的可能性高低，然后选出前 k 个概率最大的边，对它们展开进一步的处理。

空间概率 空间概率被定义为点 p_i 匹配到候选边 e_j 的可能性，概率是基于 p_i 和 e_j 之间的欧式距离计算得到的。从点 p_i 到边 e_j 的距离可以表示为 $H_{e_j}^{p_i}$ $\min\{dist(p_i,c_1),dist(p_i,c_2),dist(p_i,c_3)\}$，函数 $dist(x,y)$ 的功能是返回点 x 和点 y 之间的欧式距离，点 c_1 和点 c_2 分别是边 e_j 上的两个端点，点 c_3 是点 p_i 在边 e_j 上的垂直投影点。注意如果点 c_3 未落在边 e_j 上，距离值 $dist(p_i,c_3)$ 手动设置为无穷大 。

一般来说，GPS 设备的定位噪声可以建模为距离值 $H_{e_j}^{p_i}$ 的标准正态分布，这种分布描述点 p_i 与边 e_j 匹配的可能性高低[29,41]。因此我们基于式（2.1）来计算空间概率。

$$G_1(H_{e_j}^{p_i}) \quad \frac{1}{\sqrt{2}\ _1} e^{\frac{(H_{e_j}^{p_i})^2}{2\ _1^2}} \tag{2.1}$$

其中参数 $_1$ 是定位测量误差的标准偏差。为了估计 $_1$，我们首先使用当前优秀地图匹配算法离线获得原始轨迹在路网上的匹配结果，因此对于轨迹中所有 GPS 点，其自身到对应匹配边的距离可以容易地计算出来，最后我们基于距离的分布推导出 $_1$ 值。

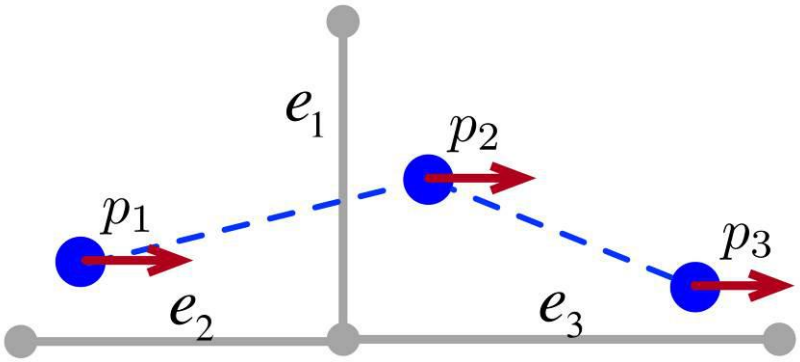


图 2.3　航向信息在地图匹配中的实用性

Figure 2.3 An illustrative example demonstrating the utility of heading direction in map matching

然而，仅依靠空间概率来识别候选边会导致错误的匹配结果。如图 2.3 就给出了这样的示例。若基于空间概率，点 p_2 的匹配边为 e_1，但实际情况下这种匹配结果明显是错误的，因为在点 p_2 处的车辆行进方向更接近边 e_3 的方向。基于该案例的思考，我们应该引入车辆航向来避免造成错误的侯选边识别。方向概率的过程详情如下：

方向概率　方向概率被定义为点 p_i 匹配到候选边 e_j 的可能性，概率是基于点 p_i 处的车辆航向 h_i 与边 e_j 方向的角度差计算得到的。 p_i 和 e_j 之间的角度差 $A_{e_j}^{p_i}$ 为 $min\{h_i \quad d_{th}, h_i \quad d_{ht}\}$。与空间概率类似，可根据式（2.2）推导方向概率。

$$G_2(A_{e_j}^{p_i}) \quad \frac{1}{\sqrt{2} \quad _2} e^{\frac{(A_{e_j}^{p_i})^2}{2 \quad _2^2}} \tag{2.2}$$

其中 $_2$ 是方向测量误差的标准偏差，可以用与估算 $_1$ 相同的方式来估算 $_2$ 的值。

基于概率 G_1 和 G_2，我们计算综合概率 G，其定义为空间概率和方向概率的几何平均值，如式（2.3）所示。

$$G \quad \sqrt{G_1(H_{e_j}^{p_i}) \quad G_2(A_{e_j}^{p_i})} \tag{2.3}$$

如前文所述，轨迹中 GPS 点具有若干候选匹配边，每个边都有一个综合概率值 G，我们识别出概率最高的前 k 个候选边（即 $top \quad k$）。注意候选边的数量 k 影响地图匹配算法的性能（即准确性和效率），我们将在 2.4 节中对其进行评估。

2.3.2 寻找路径

两个连续点 p_i 和 $p_{i \ 1}$ 的候选边在路网中通常是不相连的，它们的距离甚至可能很远。因此，我们需要找到它们之间的潜在路径。每个 GPS 点有 k 个候选边，很容易理解，对于两个连续点，一共存在 k^2 个可能的路径。在以前的工作中，通常使用 A *或 Dijkstra 算法来发现这种路径，但路径搜索的时间复杂度高，非常耗时。更糟糕的是，由于需要找到所有可能的路径，路径搜索的时间开销就更高了。为

了解决这个问题，我们充分利用车辆航向来缩小路径搜索的范围，并且利用该信息作为路径搜索的有效指引器，加速搜索过程。

在详细介绍 SD-Matching 算法之前，我们提供了一个示例来说明车辆航向对于提高路径搜索效率是有效的。图 2.4（左）展示了基于 A*算法找到从开始节点到结束节点的路径的过程。大矩形是潜在的搜索区域，其中黑色正方形颗粒是无法行走的物理屏障，车辆只能在白色正方形颗粒上行走。在输出路径之前，A*算法需要搜索所有灰色单元格（左图中蓝色实线限定的区域）。作为比较，右图展示了我们设计的路径搜索算法实际覆盖的搜索区域（右图中的灰色单元格）。可以明显看出，我们设计的算法灰色单元区域面积比 A*算法要小得多，这意味着在路径搜索时使用航向信息能够提高搜索效率。

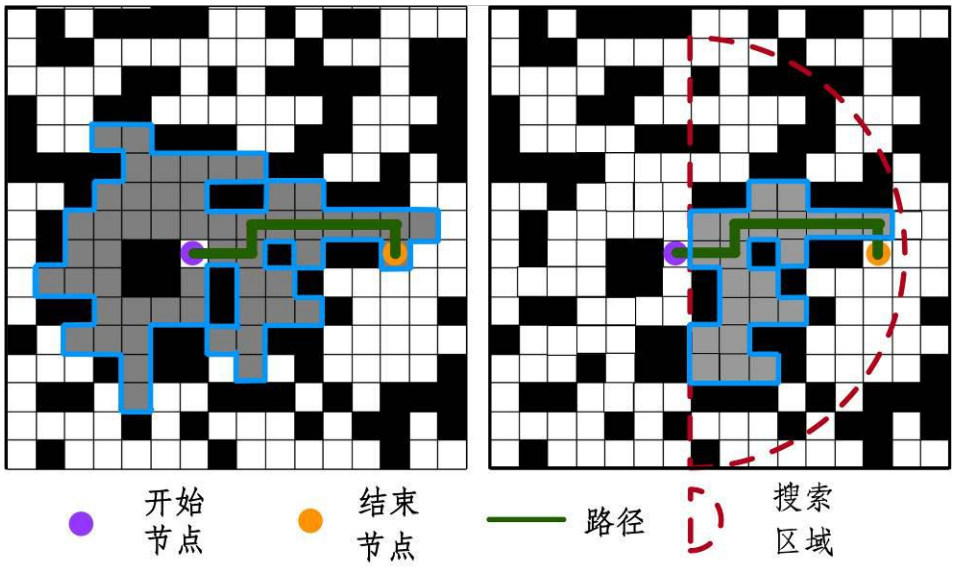


图 2.4 对比有无考虑车辆航向后路径的搜索范围大小

Figure 2.4 An illustrative example of searching zone comparison with and without the consideration of heading direction

路径搜索算法包含两个步骤，即确定潜在搜索区域和在潜在搜索区域内进行路径搜索。

第 1 步 确定潜在的搜索区域 潜在搜索区域是一个扇区（如图 2.5 中的灰色区域的所示），该扇形可以通过以下三个条件来确定：①扇区的顶点是点 p_i；②扇区的半径 r_{max} 由公式 $max(v_{p_i}, v_{p_{i+1}})$ t c 计算可得，其中参数 v_{p_i} 和 $v_{p_i\ 1}$ 分别是点 p_i 和点 $p_{i\ 1}$ 的车辆瞬时速度值；参数 c 是常数（本实验中我们设定 c 50 m），t 是 GPS 设备的采样时间间隔。注意，r_{max} 表示车辆从点 p_i 到点 $p_{i\ 1}$ 的最大行驶距

离；③扇形由两个半圆组成（图中的实线和虚线半圆），半圆的每个直径分别垂直于点 p_i 和点 p_{i-1} 的车辆航向 h_i 和 h_{i-1}。

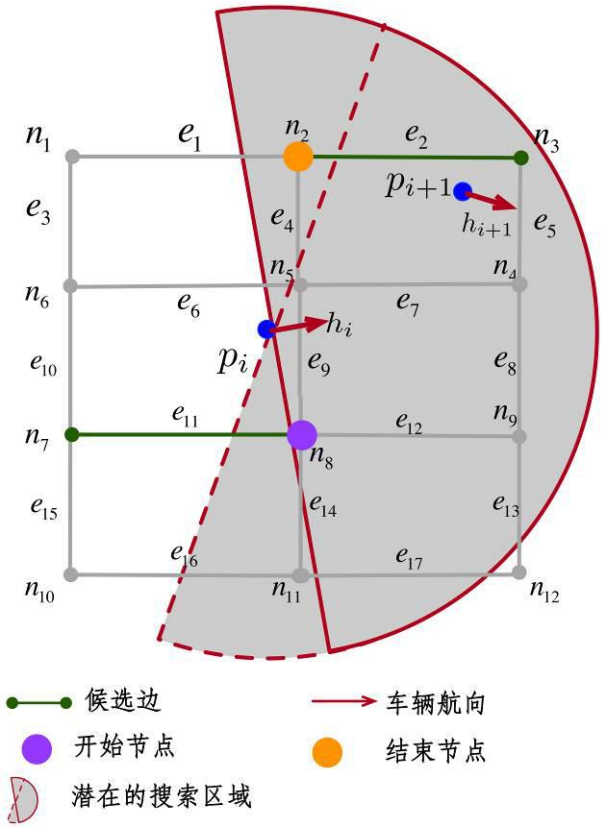


图 2.5 设计的两步路径搜索算法的说明性示例

Figure 2.5 An illustrative example of the proposed two-step path-finding algorithm

第 2 步 在潜在搜索区域内搜索路径 对于给定的点 p_i 和点 p_{i-1}，我们需要找到在潜在搜索区域内的 k^2 对候选边的 k^2 个可能路径（第 1 步的返回结果）。对于一对候选边，根据点 p_i 和点 p_{i-1} 处的车辆航向，可以很容易地确定路径的起始节点和结束节点。如图 2.5 所示，对于边 e_{11} 和边 e_2，我们很容易知道点 n_8 和点 n_2 分别是路径的起始节点和终止节点。因此，一对候选边的路径搜索问题被简化为寻找从起始节点到结束节点的路径。请注意，在搜索区域内可能没有从起始节点到结束节点的路径，一旦没有该路径，则意味着该对候选边之间不存在有效路径，这也是为什么 k^2 对侯选边的实际有效路径数远小于 k^2。

我们提出利用车辆航向作为导向器有效地找到一对以 $\langle n_s, n_e \rangle$ 为起始节点和结束节点的路径。具体步骤如下：对于潜在搜索区域内的每个节点，我们根据其位置和两个连续 GPS 点处（即点 p_i 和点 p_{i-1}）的车辆航向确定它们彼此的节点关系。对于节点 n_i，当且仅当它满足两个条件时，节点 n_c 可以是其子节点：①子节点 n_c

应该与节点 n_i 直接相连。此外，与节点 n_i 相比，节点 n_c 离节点 n_s 更近，如式（2.4）和（2.5）所示；②从节点 n_i 到子节点的方向应该与两个航向一致，在数学上满足式（2.6）。

$$dist(n_c,n_e) \quad dist(n_i,n_e) \tag{2.4}$$

$$dist(n_c,n_s) \quad dist(n_i,n_s) \tag{2.5}$$

$$\min(|h_i \quad d(n_i,n_c)|,|h_{i\ 1} \quad d(n_i,n_c)|) \quad 90^{\vdots\vdots} \tag{2.6}$$

其次，我们基于确定的节点关系构建搜索树，以起始节点 n_s 为根节点。如果该树不包含结束节点 n_e，这意味着我们不应该在此树中展开搜索，可以直接跳过此树。否则，我们可以应用深度优先搜索算法（*DFS* 算法）在此树中搜索路径[68]。一旦找到从节点 n_s 到节点 n_e 的路径，则终止对该树的搜索，直到检查完树中所有节点，整个搜索过程终止。

2.3.3 筛选路径

在两个连续的 GPS 点之间，通过前两步，我们最多可以获得 k^2 个可能路径。因此对于具有 l 个 GPS 点的原始轨迹片段，理想情况下，将存在 $k^{2(1\ 1)}$ 个可能路径来连接它们，然而实际路径的数量要比 $k^{2(1\ 1)}$ 小得多，这是因为①对于两个连续的 GPS 点存在少得多的路径（$\vdots\vdots\ k^2$）（某些节点对在第一步没有路径返回）；②不同 GPS 点之间在搜索区域内的路网碎片中并不是连通的。

对于所有可能路径，由于路径中的边成为 GPS 点真正匹配边的概率不同，因此路径 $_{mi}$ 成为真实匹配路径的概率也各不相同。我们基于路径 $_{mi}$ 是真实匹配路径的可能性进一步筛选可能路径，我们通过累加所有 GPS 点匹配到候选边的综合概率来衡量该可能性，如式（2.7）所示。

$$P_{mi} \quad \overset{l}{\underset{i\ 1}{}} G(p_i,e_j^{p_i}) \tag{2.7}$$

其中 $G(p_i,e_j^{p_i})$ 指的是点 p_i 正确匹配到边 e_j 的综合概率，具体参见式（2.3）。注意，给定原始轨迹 $_i \quad p_1,p_2, \quad ,p$ 的一条可能的匹配路径可以用 $_{mi} \quad e_j^{p_1}$ 挝 $e_j^{p_2} \quad e_j^{p}$ 表示，其中边 $e_j^{p_i}$ 指的是点 p_i 的第 j 个候选边；$e_j^{p_i}$ Ñ $e_j^{p_{i\ 1}}$ 指的是从边 $e_j^{p_i}$ 到边 $e_j^{p_{i\ 1}}$ 的路径。

2.4 实验评估

在本节中，我们进行了广泛的实验，选取现有的优秀地图匹配算法，计算它们和 SD-Matching 算法的有效性（第 2.4.2 节）和效率（第 2.4.3 节），以证明 SD-Matching 算法的优越性能。此外，我们进一步评估 SD-Matching 算法在各种密度路网下以及不同采样率下的算法的性能（第 2.4.4 节）。

2.4.1 实验设置

① 基准方法 为了展示 SD-Matching 算法的卓越性能，我们将它与当前最先进的地图匹配算法（即 ST-Matching 算法和 S-Matching 算法）进行了比较。ST-Matching 算法是目前最流行的地图匹配算法之一[29,41]，它与 SD-Matching 算法主要在以下两个方面有着不同：

第一，对于给定的 GPS 点，ST-Matching 算法基于空间和时间信息确定候选边，而我们的 SD-Matching 算法考虑空间和航向信息。也就是说，在计算对于给定 GPS 点的候选边的正确匹配概率时，它使用空间和时间信息，而我们的方法使用空间和航向信息。

第二，搜索两个连续 GPS 点之间的路径时，ST-Matching 算法采用 Dijkstra 算法（这个版本的算法记为 ST-Matching_v1）或 A*算法（这个版本的算法记为 ST-Matching_v1）。不同版本的 ST-Matching 算法拥有相同的匹配准确度，但 ST-Matching_v2 算法更高效。作为对比，我们提出一种充分利用车辆航向的新型路径搜寻算法。

S-Matching 算法与 ST-Matching 算法的区别仅在于 S-Matching 算法在识别候选边时仅利用空间信息（即仅基于空间概率）。综上所述，SD-Matching 算法最直接和明显的特征是它在地图匹配的不同阶段都利用了车辆航向信息。

② 数据准备 我们主要使用中国北京的三个数据集，第一个数据集是从网站 OpenStreetMap[69]下载的路网，该网站通过众包来收集数据[70-72]；第二个数据集是 50 辆出租车在 2015 年 9 月 15 日生成的原始 GPS 轨迹数据，包含超过 50000 个 GPS 点。另外所有出租车的采样率是相同的且恒等于 6 秒；第三个数据集是人工标记的车辆真实行进路径的数据，对于一辆出租车产生的每组轨迹，我们先手动将其分成若干包含相同 GPS 点数量的轨迹片段，轨迹片段的数量由它的长度 l 和每组轨迹集中 GPS 点的数量来确定。l 在实验中分别为 5、10、15、25、30，表 2.1 显示了在不同长度 l 下的轨迹片段的数量。对于每个片段，我们招募志愿者通过使用数字地图手动识别了它在路网上相应的匹配轨迹片段，人工标记的匹配轨迹可以视为原始轨迹的真实行驶路径。

表 2.1 不同长度 l 下轨迹片段的数量

Table 2.1 Number of raw trajectory segments under different ls

l	**5**	**10**	**15**	**20**	**25**	**30**
#轨迹片段数量	10366	5048	3456	2645	2569	1824

③ 评估标准 为了评估 SD-Matching 算法的性能，我们提出了两个标准，第一个是平均准确度，其定义为正确匹配的 GPS 点数量与要匹配的总 GPS 点数量的比例；第二个是时间成本，指的是输出一个匹配轨迹片段所需的计算时间。为了获得统计意义上的时间成本，我们使用平均时间成本和最大时间成本来评估匹配一组原始轨迹片段的效率。平均时间成本能够评估匹配算法的整体表现，而最大时间成本能够衡量匹配算法的最大时间延迟。

2.4.2 有效性评估

有两个重要参数可以影响 SD-Matching 算法的平均准确度。一个是轨迹片段的长度（即 l），另一个是给定 GPS 点候选边的数量（即 k）。我们研究了在不同 ls 和 ks 下，SD-Matching 算法的平均准确度。作为比较，我们也提供了基准方法的结果。

改变 l 图 2.6 统计了不同长度的轨迹片段下不同匹配算法的平均准确度。片段长度（l）从 5 增加到 30，间隔为 5。注意，因为 ST-Matching 算法的不同版本仅影响轨迹匹配的时间成本，因此我们只显示 ST-Matching 算法一个版本的结果。更具体地说，在路径搜索中，A*算法比 Dijkstra 算法需要更少的时间（关于时间成本的详细评估，参见第 2.4.3 节），但它们将返回相同的结果。

从图中可以看出，随着轨迹片段的增长，这三种算法的平均准确度都在增加。这可能是因为随着长度的增加，越多 GPS 点的空间信息可用于提高算法准确度。因为离线地图匹配算法可以使用完整轨迹的信息，所以离散算法通常比在线算法具有更高准确性。更重要的是，对于所有长度的轨迹片段，SD-Matching 算法有比 ST-Matching 算法有略微更高的平均准确度，而且它们的表现都明显优于 S-Matching 算法。总之，在平均准确度上，表现最好的是我们提出的 SD-Matching 算法，这证明了在地图匹配中，使用车辆航向能提高匹配算法的准确度。在这个实验中我们设定 k 6。

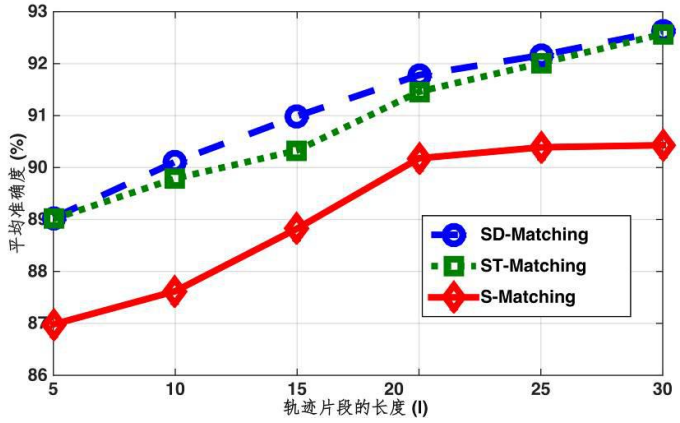


图 2.6 不同 ls 下不同算法的平均准确度

Figure 2.6 Comparison results of mean accuracy for different algorithms under different ls

改变k　图 2.7 统计了当每个 GPS 点选择不同候选边的数量k时三种算法平均准确度的结果。从图中可以观察到，如果我们为每个 GPS 点识别出更多的匹配边，则可以提高算法的平均准确度。这是因为，随着k的增加，候选边包含正确匹配结果的概率变得更大，因此可以获得更高的平均准确度。然而，当k从 3 增加到 6 时，平均准确度的提高非常有限。另外，对于所有的ks，SD-Matching 算法的性都能略好于 ST-Matching 算法，它们也都明显优于 S-Matching 算法。在这个实验中我们设定l=10 。

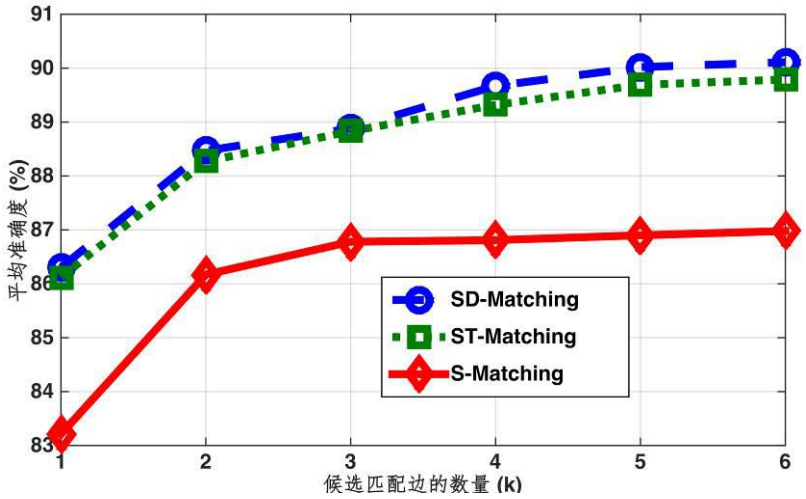


图 2.7　不同ks下不同算法的平均准确度

Figure 2.7 Comparison results of mean accuracy for different algorithms under different ks

2.4.3 效率评估

原始轨迹匹配所需的计算时间对于在线算法至关重要。与评估算法的有效性类似，我们研究了不同轨迹长度（ls）和候选匹配边（ks）下匹配算法的时间成本表现。为了更好地了解 SD-Matching 算法，我们还提供了算法在不同阶段的详细计算时间。

改变l　图 2.8 显示了不同轨迹片段长度下四个算法的平均时间成本。当l变大时，每个算法都需要耗费更长的平均计算时间。但是，所有算法的时间成本具有不同的增长速度（即曲线斜率不同）。例如，两种版本的 ST-Matching 算法时间成本增长比 SD-Matching 和 S-Matching 算法快的多，而后两种算法有着相似的计算时间开销，其中 S-Matching 算法的效率较高，这是因为它不需要计算方向概率来识别候选边。总之，SD-Matching 和 S-Matching 算法都是高效的，并且对于所有长度的轨迹片段来讲，响应时间不超过 0.5s。结合图 2.1 和图 2.2 所示的结果，从图 2.6 和图 2.8 中，我们可以得出结论，地图匹配准确度的提高是以增加计算时间为代价的。确定l的大小时，我们需要在准确性和计算时间之间进行权衡。例如，

对地图匹配算法而言，若设置l 5，虽然在 0.25 秒内返回结果，但是匹配算法的准确度可能太低，从而无法在实际中应用。图 2.9 显示了在不同ls 下四种算法的最大时间成本，可以观察到与平均时间相类似的趋势，唯一的区别是最大时间成本明显大于相应时间成本的平均值。例如，当l 30 时，ST-Matching_v2 算法的最大时间值大于 6 秒，而相应的平均值小于 2 秒。这项实验中我们设置k 6。

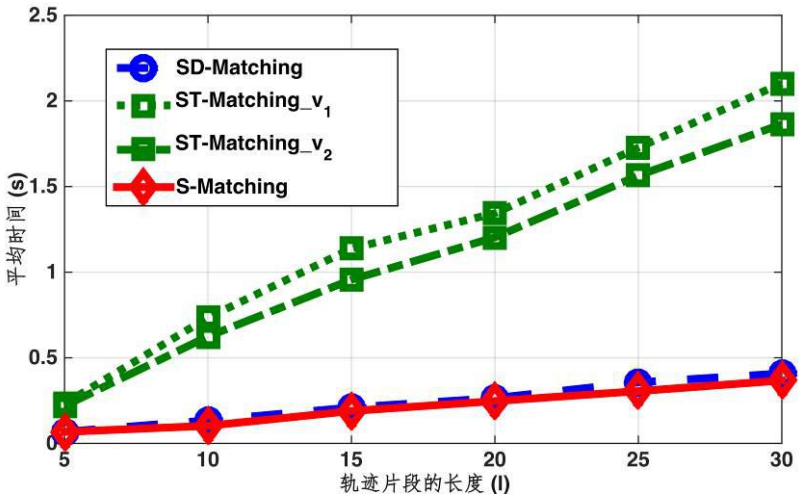


图 2.8 不同ls 下不同算法的平均时间成本

Figure 2.8 Comparison results of mean time cost for different algorithms under different ls

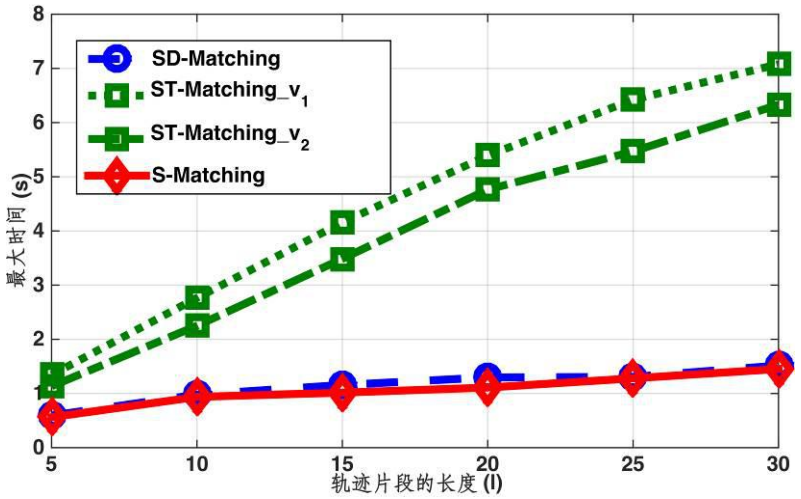


图 2.9 不同ls 下不同算法的最大时间成本

Figure 2.8 Comparison results of mean time cost for different algorithms under different ls

改变k 图 2.10 统计了当每个 GPS 点选择不同数量的候选边时四种算法的平均时间成本。从图中可以观察到，平均时间成本随着k 变大而增加。具体来说，随着k 变大，对于 ST-Matching 算法的两个版本，平均时间指数式地增加，然而

对于 SD-Matching 和 S-Matching 算法，平均时间增长缓慢。这是因为，对于 ST-Matching 算法，对每对开始节点和结束节点，Dijkstra 或 A *算法都会进行路径搜索，相邻 GPS 点之间路径的数量受 k^2 控制。而对于 SD-Matching 和 S-Matching 算法，路径搜索只会在包含起始节点和结束节点的树上执行。而在现实案例中，树的数量远小于 k^2。此外，使用 *DFS* 算法进行路径搜索也比 Dijkstra 和 A *算法更加高效。图 2.11 显示了四种算法在不同 *ks* 下的最大时间成本几乎是相应平均时间成本的 4 倍。对于 SD-Matching 和 S-Matching 算法，与平均时间成本相比，随着 *k* 变大，最大时间成本会显著增加。我们在本研究中确定 l　10。

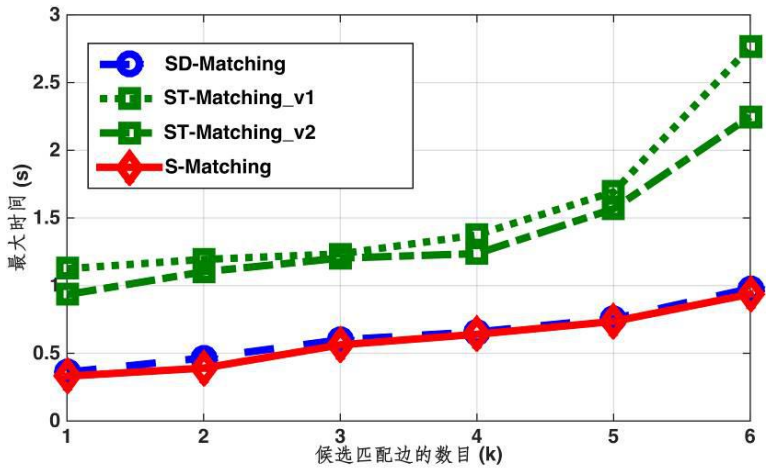


图 2.10　不同 *ks* 下不同算法的平均时间成本

Figure 2.10 Comparison results of mean time cost for different algorithms under different *ks*

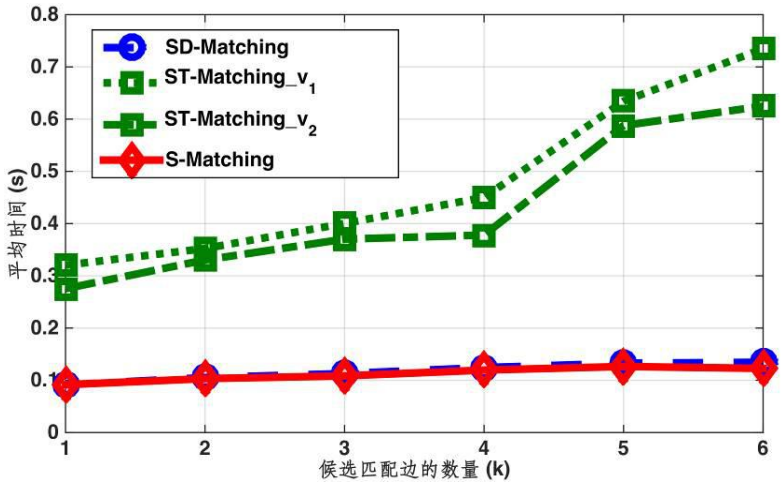


图 2.11　不同 *ks* 下不同算法的最大时间成本

Figure 2.11 Comparison results of maximum time cost for different algorithms under different *ks*

不同阶段的时间成本　图 2.12 统计了在不同的 *ls* 下，SD-Matching 算法在不同阶段详细的时间成本。从两个图中可以看出，大部分计算时间消耗于前两个阶段，

第三阶段的时间开销是微不足道的。第一阶段耗时的主要原因是为了识别*top k*候选边，我们首先需要计算给定 GPS 点与城市路网中所有边的距离，该过程是耗时的。但是通过实施一些索引技术可以很容易地加速这个阶段，具体加速方法的细节可以参考第 5 章对地图匹配算法第一阶段使用的索引技术。如（图 2.12 的）上图所示，当*l*越长，前两个阶段的时间成本呈线性增加，导致总时间成本也几乎呈线性增加，与图 2.8 显示的结果相同。如（图 2.12 的）下图所示，第一阶段的时间成本保持稳定，随着*k*变大，第二阶段和第三阶段的时间成本以指数级别增长，与图 2.10 显示的结果相同。该实验中的参数设置与前两项评估中的参数设置相同。

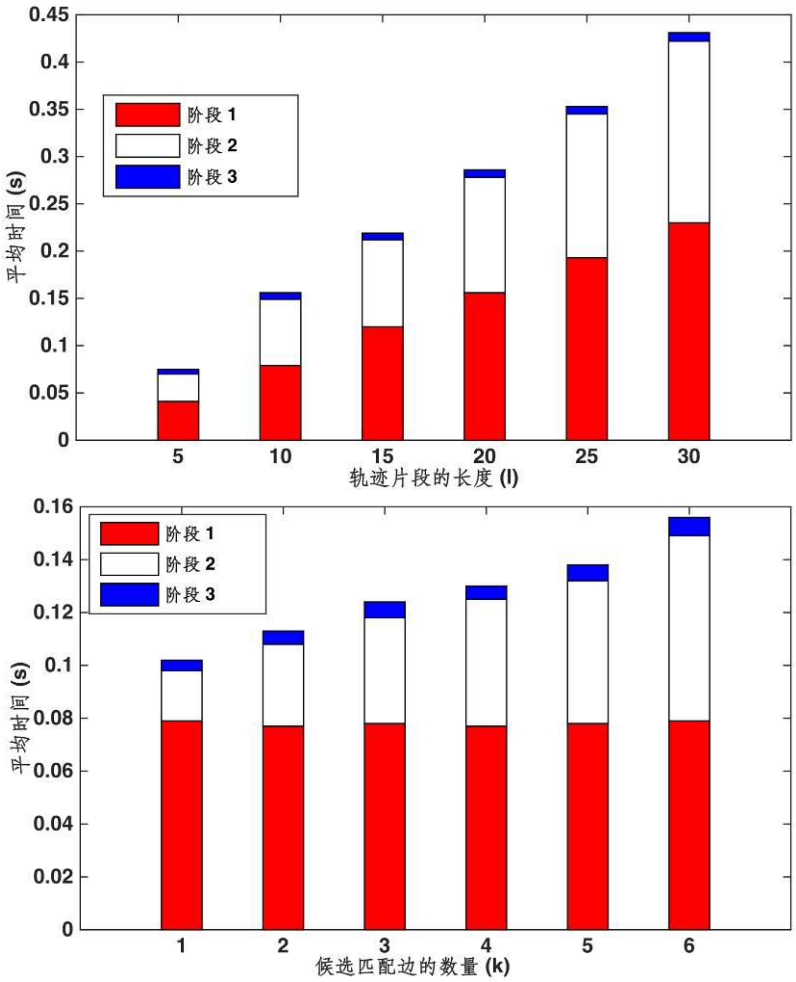


图 2.12 不同的*ls*和*ks*下 SD-Matching 算法不同阶段的时间成本

Figure 2.12 Detailed time cost at different stages in SD-Matching algorithm under different *ls* and *ks* , respectively

2.4.4 其他性能评估

我们进一步评估了不同路网密度和采样率下 SD-Matching 算法的性能，并得到了以下结果。在这个实验中我们设置k　6 和l　10 。

改变路网密度 我们将整个北京城市分为两个区域，即三环内的区域（区域 I）和三环外的区域（区域 II）。区域 I 路网密度是区域 II 的 6 倍，路网密度定义为每平方公里的节点和边的数量。表 2.2 不仅进一步说明了这两个区域的路网细节，还显示了 SD-Matching 算法的性能结果，包括准确性、平均时间成本和最大时间成本。我们从表中可以观察到，在道路稀疏的地区（即区域 II），SD-Matching 算法不仅获得更高的准确性，而且消耗了更少的计算时间，这可能是因为当路网简单时，第二阶段的路径搜索会更简单更准确。

表 2.2 不同路网密度下的结果

Table 2.2 Results under different road network densities

	区域 I	区域 II
面积（km^2）	≈334	≈5992
#节点	38000	10300
#边	45000	112000
准确度（%）	89.33	92.13
平均花费时间（s）	0.1513	0.1143
最大花费时间（s）	1.124	0.9562

改变采样率 为了研究不同采样率下 SD-Matching 算法的性能，我们使用简单的方法降低了采样率，这里的采样率是指 GPS 设备的采样间隔。算法的表现如表 2.3 所示。正如预期那样，准确性随着采样间隔的增加而下降[44]。这是由于两个连续 GPS 点之间的行程距离随着采样间隔的增加而变长，所以匹配结果存在更多的不确定性，算法的准确性明显降低。类似地，更低的采样率使得连续采样点之间的距离变长，路径搜索的时间开销因此增加。但是，在处理低采样率 GPS 轨迹数据时，与 Lou 等人的 ST-Matching 算法相比，SD-Matching 算法仍然实现了良好的匹配结果[29,41]。例如，即使采样率是 150 秒，算法的准确度也可以大于 80％。

表 2.3 不同采样率下的结果

Table 2.3 Results under different road network densities

采样率（s）	6	30	60	120	150
准确度（%）	90.11	86.19	82.40	81.46	80.36
平均花费时间（s）	0.13	0.18	0.46	0.56	0.83
最大花费时间（s）	0.97	1.03	1.74	3.69	4.21

2.5 半自动标记软件 TLabel

当评估地图匹配算法的准确度时，研究者不仅需要把原始 GPS 轨迹数据输入到算法得到对应的匹配结果，还需要将获得的匹配结果与车辆的真实行驶路径进行对比，从而计算匹配准确度，判定算法的优劣。一般来说，原始 GPS 轨迹数据丰富且易于获得，而车辆的真实行驶路径却难以获得。因为真实路径无法自动从设备获取，只能使用人工进行标记。而且原始轨迹数据是海量的，所以标记任务异常繁重。更糟糕的是，特别是在密集的道路网中标记任务是有难度的。这是因为，标记人员很难判断给定 GPS 点真正的匹配边，极易判断失误。为了解决以上问题，即提高标记速度并保证准确度，我们开发了一款名为 **TLabel** 的半自动真实行驶路径交互式标记软件。其中字母 T 有两层含义，第一层表示 True（真实），第二层表示 Trajectory（轨迹）。

软件 **TLabel** 有两个优点，①将原始 GPS 轨迹和相应城市路网集中在一个用户友好型的可视化交互界面上，可以方便标记人员观察数据并快速做出判断，提高了标记速度；②提前挑选出 GPS 点附近最有可能是真正匹配边的六条边，让标记人员从中筛选。这种做法过滤掉大量无效且很可能错误的候选边，因此提高了标记准确度。我们下面将详细介绍 **TLabel** 的可视化交互界面和操作指南。

可视化交互界面 界面主要包含五个组件，第一个是“数据加载端”，如图 2.13 中包含“1”的红色椭圆形所示，这个组件用来加载原始轨迹和路网数据。关于路网数据，用户既可以选择手动导入，也可以选择嵌入软件中样本城市的路网数据（如中国北京、重庆和杭州等城市）；第二个组件是“绘制命令按钮”，如包含“2”的红色椭圆形所示。用户每点击一次“Start Plotting”按钮，在第三个组件“可视化界面”，如包含“3”的红色椭圆形所示，就会出现一个红色的“+”，该符号表示等待匹配 GPS 轨迹点的观察位置；第四个组件是“侯选边筛选区”，如包含“4”的红色椭圆形所示。可视化界面中显示了当前 GPS 点的六条最可能的候选边，它们有着不同的颜色，分别对应侯选边筛选区的“Edge X!”的颜色。在图 2.13 的样例下，很明显紫色的候选边是真正的匹配边（即应该选择“Edge 1!”）。在该组件中，除了六条候选边，

TLabel 还给出了两个命令“Not sure!”和“No found!”。若用户选择了“No sure!”，则意味着无法判断六条中的哪一条为真正的匹配边；若选择 “No found!”，则意味着六条侯选边中不存在真正的匹配边；最后一个组件是当前“GPS 点索引号”，它能提醒标记者当前处理 GPS 点在数据集中的序号。更重要的是，一旦用户关掉软件停止标记，再次打开 **TLabel** 时，直接输入 GPS 断点的位置，即可从断点处继续标记数据。

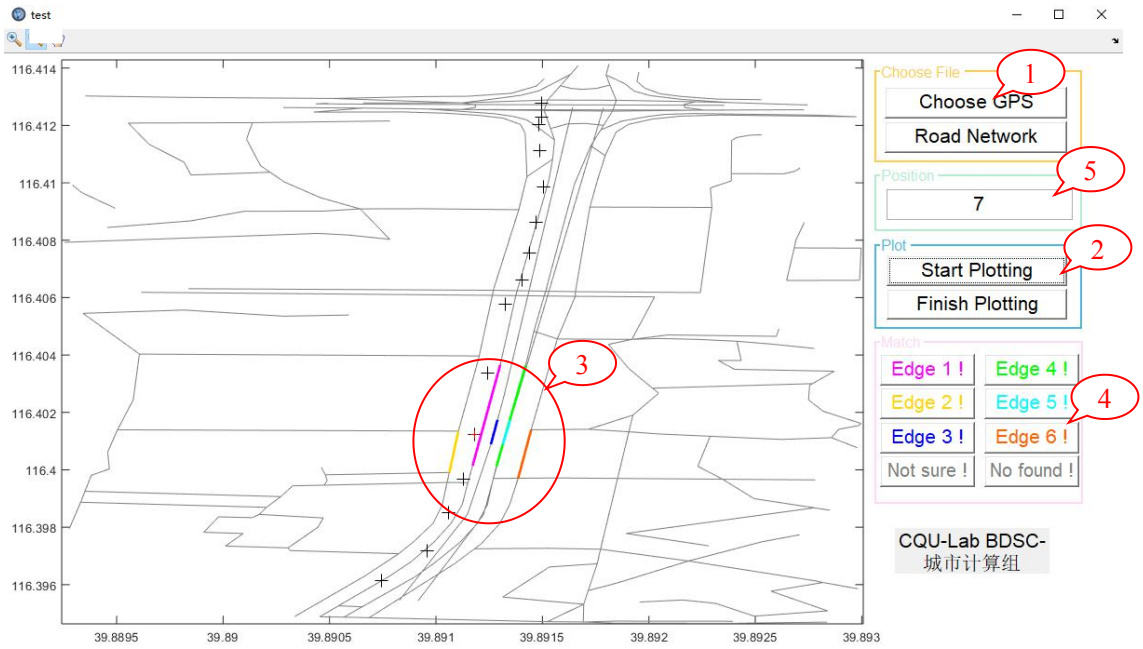


图 2.13 **TLabel** 的可视化交互界面

Figure 2.13 Visual interface of software **TLabel**

操作指南　根据上节的叙述，很容易获得 **TLabel** 软件的操作流程图。

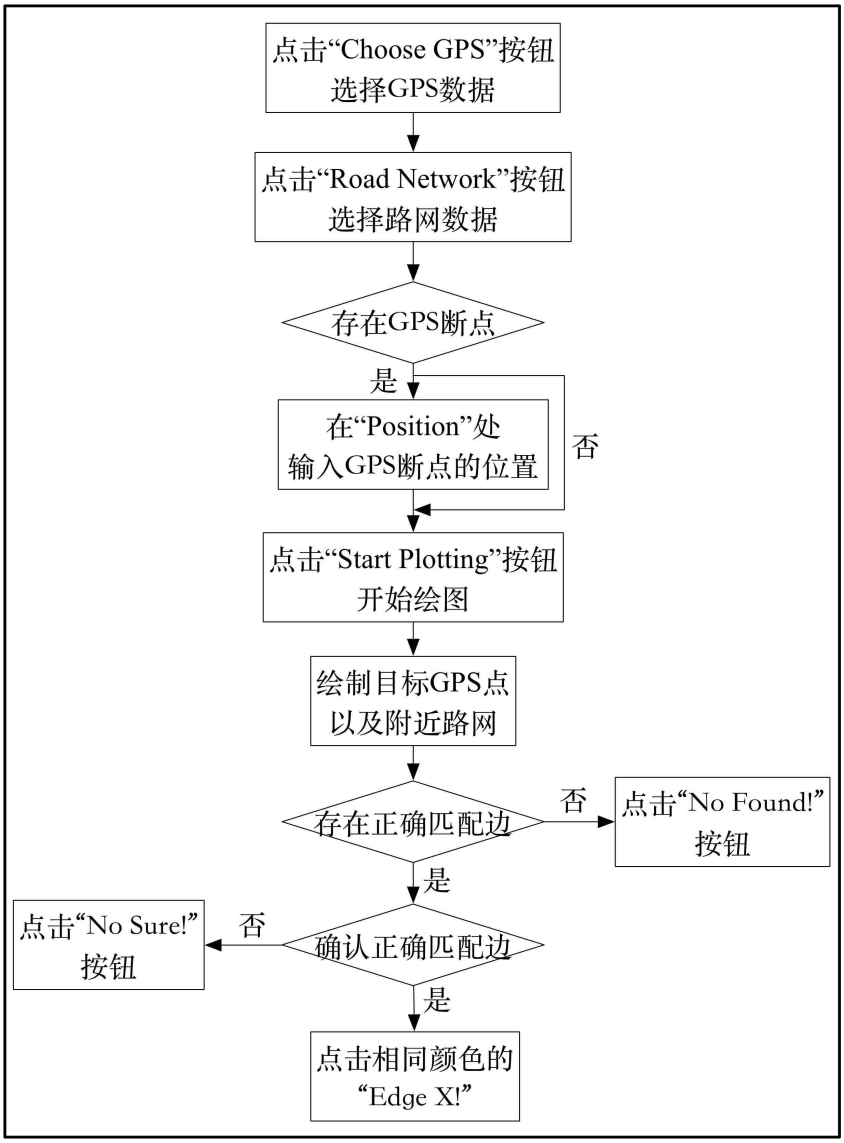


图 2.14 **TLabel** 的操作流程图

Figure 2.14 Operation flow chart of software **TLabel**

2.6 章节小结

在本章中，我们提出了全新的地图匹配算法，即 SD-Matching 算法。算法在每个阶段都利用了车辆航向，该信息不仅能提高地图匹配的准确度，还能降低计算时间成本。具体来说，车辆航向以可证明的方式缩小了路径搜索区域并作为路径搜索的有效引导器。通过大量实验证明，SD-Matching 算法在效率和准确度上皆优于当前最受欢迎的 SD-Matching 算法。另外，即使降低采样频率，SD-Matching 算法仍保持高准确度。

3　轨迹压缩：空间维度

本章研究针对空间轨迹进行高效压缩。内容安排如下：3.1 节是本章的前言，介绍了空间轨迹压缩算法的研究背景和现状；3.2 节介绍了一些主要概念，并对设计的算法 HCC 进行了概述；3.3 节详细阐述 HCC 算法以及解压算法；3.4 节全面评估了 HCC 算法的性能；最后 3.5 节对本章进行了总结。

3.1 前言

近年来，随着 GPS 设备的普及和移动互联网的成熟，海量移动物体的轨迹数据正以前所未有的速度和规模被积累和记录，车辆轨迹数据是其中一个典型的例子。与其他物体相比，车辆只能在底层路网上行驶，因此车辆具有特殊的空间约束。目前，移动车辆的位置通常以固定的时间间隔采样并报告给云端数据中心，这对于很多应用来说，是冗余的，而且导致了通信开销、存储和计算问题。更糟糕的是，由于越来越多的公共交通和私人车辆已经配置了 GPS 设备，并不断地将它们的轨迹数据发送到数据中心，导致了通信和存储成本的增加[15,16]。并且，大量的数据还阻碍了计算任务，包括可视化、模式挖掘等[17-20]。

在将轨迹数据发送到数据中心之前减小轨迹数据的大小，是缓解上述问题的有效途径之一[21,73]。最直接的方法是收集较少的轨迹数据，减少报告车辆位置的频率。然而，这种方法是存在问题的。这是因为，收集的数据可能过于稀疏，无法推断中间的详细驾驶路径，导致了低质量的城市服务应用。相比之下，在线轨迹压缩试图从寻找轨迹表示的角度出发来解决这些问题。虽然这是一个常见的解决措施，但这个问题目前为止还没有得到很好的解决。针对不同的应用，研究人员提出了许多轨迹压缩方法，其中最著名的是 Douglas-Peucker (DP)算法及其变体[48,53,58]。它们的核心原理是根据 GPS 点对轨迹形状的重要性，通过保留一些重要的点来简化轨迹。然而，由于 GPS 设备的定位误差，GPS 点的重要性可能会被错误地估计。由于路网数据的广泛性，车辆轨迹往往被匹配到路网中的某条路径上。这种做法 1)具有显著降低定位误差带来的副作用的潜力；2)能生成更自然、语义更丰富的轨迹表示。因此，一般来说，基于地图匹配的轨迹压缩优于非基于地图匹配的[55,56,74]。然而，轨迹匹配的任务通常是计算密集型和资源消耗型的[19,75]，而嵌入在移动车辆上的 GPS 设备的计算能力十分有限，无法承担如此繁重的任务。因此，本文的首要目标是开发一种在移动环境下有效的轻量级地图匹配算法。

通过地图匹配返回的匹配轨迹通常由路网中的一系列连通边表示[12,13]。然而，这种表示是冗余的，可以进一步的被减少，这就推动了对轨迹压缩的研究。从本

质上讲，轨迹压缩的目的是找到一个紧凑、简洁的轨迹表示。更准确地说，轨迹压缩的目标是去除轨迹中一些不必要的边，并且可以利用保留的边恢复已经丢弃的边。一个典型的算法是 PRESS，它利用最短路径的原理来实现轨迹压缩。更详细地说，给定一个从 e_i 到 e_j 的子轨迹，如果中间边序列恰好遵循从 e_i 到 e_j 的最短路径，则子轨迹可以表示为 e_i,e_j 。其原理是，仅使用一对边就可以很容易地推断(或恢复)中间舍弃的边序列。由于驾驶员在很多情况下倾向于选择最短路径，这样的压缩可以有效地减少每个轨迹中边的数量，而在之前每条边都需要被保存。但是 PRESS 需要计算和存储路网中任意两条边之间的所有最短路径，这在移动环境下是不可行的。例如，北京市所有最短路径的存储空间超过了 100GB。因此，在匹配轨迹的基础上开发一种经济有效的在线轨迹压缩算法是本文的另一个目标。

3.2 主要概念

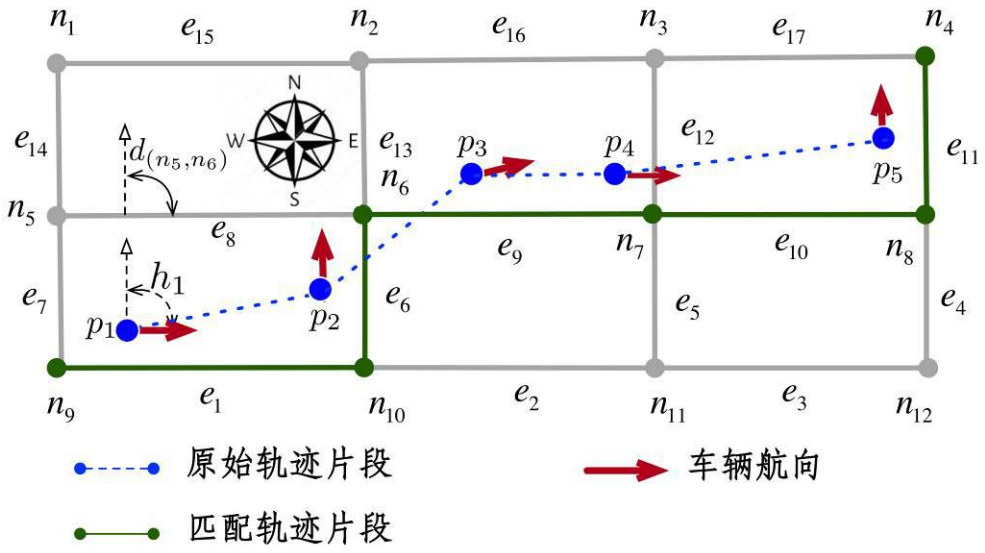


图 3.1　主要概念的例证

Figure 3.1 Illustration of main concepts used in this section

定义 1 （边的方向） 边 e_i 有两个方向，分别记为 $d(n_h,n_t)$ 和 $d(n_t,n_h)$。其中 $d(n_h,n_t)$ 定义为从节点 n_h 到节点 n_t 的边方向，以正北方向作为基准。边的方向可以基于两个节点的经纬度计算得到。如图 3.1 中的 $d(n_5,n_6)$ 是边 e_8 从 n_5 到 n_6 的一个方向。

定义 2 （车辆航向） 参见第 2 章的定义 3。

定义 3 （原始轨迹片段） 参见第 2 章的定义 5。

定义 4 （匹配轨迹片段） 参见第 2 章的定义 6。

定义 5 （压缩轨迹片段） 给定一个匹配轨迹片段 $_m$ e_i, e_n ，其压缩形式表示为 t_i, $_c$ ，t_i 是此匹配片段的开始时间。$_c$ 是 $_m$ 的子集，记为 $_c$ e_i, e_m ，其中 m n，并且每对连续边在路网中通常是不相连的。压缩轨迹流 T_c 由无数的压缩轨迹片段 $_c$ 及其起始时间组成，记为 T_c t_1, $_{c1}$, t_2, $_{c2}$, 。

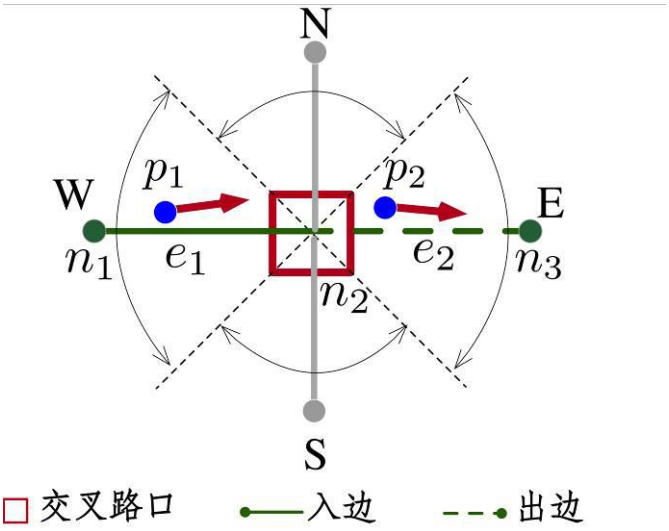


图 3.2 航向变化，入边和出边的图示

Figure 3.2 Illustration of heading change, in- and out-edges

定义 6 （在十字路口处车辆航向的改变） 交叉路口处的航向变化被定义为在车辆进入交叉路口之前的最后 GPS 点（例如 p_1）的车辆航向与离开交叉路口之后的第一个 GPS 点（例如 p_2）的车辆航向之间的角度差。根据角度值，航向变化 $(0 \sim 360)$ 分为了四类，如图 3.2 所示，它们分别表示为 $N(0 \sim 45, 316 \sim 360)$，$E(46 \sim 135)$，$S(136 \sim 225)$ 和 $W(226 \sim 315)$。属于 E 的变化表明车辆在经过十字路口时会直行，而属于其他三个类别的航向变化则意味着车辆在通过交叉路口时会转弯（左转、右转或调头）。

定义 7 （入边和出边） 如图 3.2 所示，入边和出边指的是车辆进入（例如边 e_1）和离开（例如边 e_2）交叉路口的边。出边是由入边和在交叉路口车辆航向的变化决定的。

为了简化航向变化的计算，我们简单地使用交叉路口处入边和出边方向的角度差来近似估计给定车辆在交叉路口处的航向变化。因为航向方向信息不再被保存在匹配轨迹片段中，通过这种方式还可以估计所匹配轨迹片段中的航向变化。

3.3 HCC 压缩算法及解压

3.3.1 算法设计动机

我们首先通过下面的例子来说明空间轨迹压缩算法的基本原理。导航系统向司机推荐一条从起点到终点的路线时，GPS 导航系统通常只在十字路口驾驶方向改变前提醒驾驶员(即直行/左转/右转/调头)，而不是路网中逐边指导。当推荐驾驶方向时，导航系统为了避免驾驶员分心，实际上提供了更简洁的轨迹表示，即在十字路口处提醒驾驶员更改的航向信息，如左转、右转等。受到这个观察的启发，文献[17]的作者们提出了一个顺时针压缩框架，称为 Clockwise Compression Framework(简称 CCF 算法)。匹配轨迹由在所有交叉路口的一系列出边表示，基于交叉路口的 ID 和车辆航向的改变，使用唯一代码对出边进行编码。压缩轨迹中元素的数量等于交叉路口的数量(不包括起始边和结束边)。由于交叉口的数量小于匹配轨迹中边的数量，因此采用 CCF 算法的压缩轨迹占用的存储空间较小。在实际生活中，我们通常在十字路口选择“直行”，这种情况下，不需要保存航向变化较小的出边。因此，在 CCF 算法的基础上，我们认为如果只保留具有显著航向变化的出边，则可以进一步减少压缩轨迹占用的空间。这样，与 CCF 算法相比，预计压缩轨迹中的元素数量会更少，这促使我们提出了基于航向变化的空间轨迹压缩算法。

为证明我们的改进能够提高压缩算法的性能（提高压缩率），我们进一步提供有关航向改变的统计数据，即驾驶员选择“直行”或是“转弯”的百分比。根据我们的数据，航向变化属于“直行”的百分比超过 92%，而航向变化属于“转弯”的百分比小于 8%。因此，如果用具有显著航向变化的出边来表示压缩轨迹，则可以大大减少轨迹中元素的数量，从而极大提高压缩性能。注意，在研究中，我们统计的航向变化的总次数超过 100000 次。因为压缩算法是基于航向变化的，我们将它命名为航向变化压缩算法 Heading Change Compression(简称 HCC 算法)。

3.3.2 算法细节

正如讨论的，我们设计的 HCC 算法的关键原理是只保留车辆转弯(或 u 型)时的出边，而不是保留在一个匹配轨迹片段中所有的出边。算法 3.1 总结了 HCC 算法的整个过程。注意，匹配轨迹片段($_m$)是 HCC 算法的基本处理单元。

算法 3.1 中的第 1 行表示压缩轨迹的初始化。具体来说，HCC 算法保存输入的匹配轨迹片段 $_m$ 中的第一条边 e_1；第 2～6 行表示循环操作，在循环中，HCC 算法对输入轨迹中剩余的边逐一进行扫描和检查，直到最后一条边；详细来说，对于每一个输入轨迹边 e_i，HCC 算法首先确定 e_i 与其边后一个边 $e_{i\ 1}$ 的共同节点(第 3 行)，然后判断该节点是否是一个十字路口(第 4 行)。如果是，那么继续确定车辆在该路口的航向变化(第 5 行)；如果车辆在十字路口被确定为“直行”，压缩

轨迹片段 $_c$ 中添加边 $e_{i\ 1}$ (第 6 行)；否则，HCC 算法就会跳过记录这一边，继续处理下一条边 $e_{i\ 1}$；最后，当 HCC 算法记录最后一条边 $e_{|\ m|}$ 时，整个过程终止，并输出最终的压缩轨迹片段 $_c$ (第 7 行)。可以观察到，对于任何输入的匹配轨迹片段，HCC 算法总是在压缩轨迹片段中保留第一个和最后一个边。HCC 算法的时间复杂度是 $O\ |\ _m\ |$ ，其中 $|\ _m|$ 是输入匹配轨迹 $_m$ 的边的数量。

算法 3.1 $HCC(\ _m, G(N,E))$
1： $_c$　e_1;
2：for i　2 **to** $|\ _m|$　1 **do**
3：　s　$identifyNode(e_i, e_{i\ 1})$;
4：　**if** $isInter\,\mathrm{sec}tion(s, G(N,E))$ **then**
5：　　**if** $\sim isGoStraight(e_i, e_{i\ 1})$ **then**
6：　　　$_c$　$_c$　$e_{i\ 1}$;
7： $_c$　$_c$　$e_{|\ m|}$

然而，鉴于路网的复杂性和所定义的航向变化类别的粗粒度(参见定义 6)，HCC 算法可能会导致压缩轨迹在还原时产生歧义。例如，同一航向变化可能会识别出不同的出边。我们用一个简单的例子来说明该问题。如图 3.4 所示，两个匹配轨迹有相同的起始边，例如，$_{m1}$　e_1, e_2, e_3, e_4, e_8 和 $_{m2}$　e_1, e_5, e_6, e_7, e_8 。根据我们设计的 HCC 算法，这两种轨迹的压缩轨迹是相同的，即 e_1, e_8 （在以后的描述中，此案例称为 CASE 1）。这是因为 e_1 和 e_2 之间的航向变化与 e_1 和 e_5 之间的航向变化都被识别为“直行”。在这种情况下，两条出边都被丢弃。明显这种情况是不正确的，应该避免。

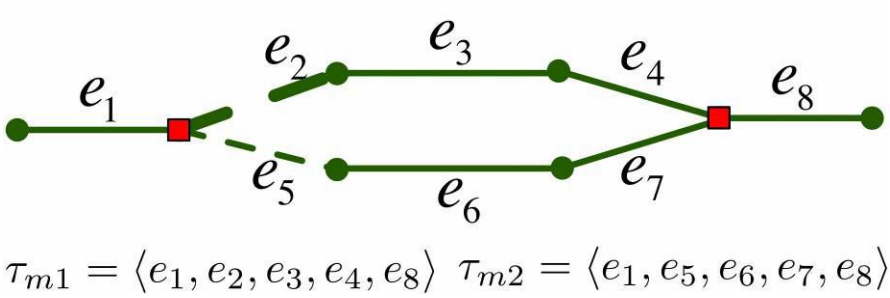


图 3.4 压缩轨迹还原时产生歧义的说明示例

Figure3.4 Illustrative example of trajectory ambiguity

为了解决这个问题，一个简单的做法是，如果交叉路口有不止一条边被识别为“直行”类别，即使一条输入的边被判定为“直行”，HCC 算法仍然保存条边。假如按照这种做法，图 3.4 所示的两个轨迹的压缩轨迹分别为 e_1,e_2,e_8 和 e_1,e_5,e_8 。然而，这种做法无疑增加了需要保存边的数量。在不增加存储空间的前提下，为了解决压缩轨迹中存在歧义的问题，我们将频繁边压缩 Frequent Edge Compression(简称为 FEC)原理嵌入到 HCC 算法中，具体如下。

FEC 算法 这个想法的灵感来源于驾驶员在穿越十字路口时可能更喜欢选择某一些出边，而不是其他的边，就像我们观察到驾驶员倾向于选择从起点到目的地的最短路径一样。因此，当驾驶员的轨迹包含这些频繁的出边时，这些边的数量可以进一步被减少。换句话说，频繁出现的边不需要被记录。利用 FEC 算法的原理，HCC 算法有望获得更好的压缩性能。我们再次以图 3.6 所示的两条轨迹为例，假设边 e_2 比边 e_5 更受欢迎，我们可以在压缩匹配轨迹 $_{m1}$ 时丢弃边 e_2。加入 FEC 算法后，压缩轨迹 $_{c1}$ 和 $_{c2}$ 分别为 e_1,e_8 和 e_1,e_5,e_8 。因为 CASE I 在我们的轨迹数据中很常见，所以这种改进的效果是显著的。

定理 1：HCC 算法的压缩率 (cr) 随轨迹片段长度的增长而增加。但是，当 l 过长时，继续增加长度，cr 增加变得微乎其微。

证明 我们假设原始轨迹数据集 T 包含 L 个 GPS 点，T 被切割成 n 个长度为 l 的轨迹片段，因此我们可以知道 $L \quad nl$ 。对于每一个原始片段，记为 $_i \quad p_i, \quad ,p_{i\ l\ 1}$ 。 $_{ci}$ 表示为相应的压缩结果，因此压缩率可以通过式（3.1）来计算得到。

$$cr \quad \frac{space(T)}{\sum_{i\ 1}^{n} space(\ _{ci})} \tag{3.1}$$

其中函数 $space(T)$ 指代原始数据集占用的存储空间，$space(\ _{ci})$ 指的是压缩轨迹片段 $_{ci}$ 占用的存储空间。根据 HCC 算法的原理，压缩轨迹片段的存储空间其实是所有保存的边的大小。从算法 3.1 我们可以知道，压缩轨迹片段的起始边和结束边总是保存在压缩结果中，因此 $space(\ _{ci})$ 可以通过式（3.2）表示。

$$space(\ _{ci}) \quad (2 \quad x_i)space(edge) \tag{3.2}$$

其中 $space(edge)$ 指的是一个边占用的存储空间，这个存储空间可以视为恒定的且易于获得。在我们的案例中，每个边用一个九位数表示。$2 \quad x_i$ 表示为压缩轨迹片段 $_{ci}$ 中边的数量。

通过将式（3.2）带入到式（3.1），我们可以获得式（3.3）

$$cr \quad \frac{space(T)}{2n \quad space(edge) \qquad \sum_{i\ 1}^{n} space(edge)} \tag{3.3}$$

对于一个原始轨迹数据集T，函数 $\sum_{i=1}^{n} x_i$ 取决于所有需要保存的边的数量，跟轨迹片段长度无关，其数值可以认为其是恒定的，我们记为C。当轨迹片段长度从l变成l l时，压缩率的变化可以通过式（3.4）计算得到。

$$\frac{cr_{i+1}}{cr_i} \quad \frac{2n \quad space(edge) \quad C}{2n' \quad space(edge) \quad C} \tag{3.4}$$

其中n'是长度为l l的轨迹片段的数量。我们可以轻易的知道n' n，因此 $\frac{cr_{i+1}}{cr_i}$ 1总是成立的，这暗示着随着l变成，压缩率也随之增加。

另外，如果l非常大，那么n和n'都是非常小的，因此在式（3.4）中的分子和分母上C都是比较大的。而$\frac{cr_{i+1}}{cr_i}$ 1，这暗示着当l过大时，随着l的增加，压缩率继续提高已经变得非常有限。

车辆位置估计和误差测量 根据定义 5 以及 HCC 算法的原理可知，压缩轨迹片段仅记录车辆进入压缩轨迹片段的时间戳，利用压缩轨迹的时间标签，找到用户感兴趣车辆的定位是非常容易的。我们假设压缩轨迹片段是$\{t_1, e_1, e_2, e_5\}$，用户查询t_3时刻车辆的位置，如图 3.5 所示。为了估计车辆在t_3时刻的位置，我们假设车辆在轨迹片段上是匀速运动的。车辆在轨迹片段上的行驶距离和时间分别为片段的长度和下一个轨迹片段时间戳与当前片段时间戳的差值，因此车辆的速度为行驶距离和时间的比值。根据时间差t_3 t_1以及车辆的行驶速度，我们可以估算出车辆的位置，即图中p_3"。真实位置是 GPS 点p_3在路网上的投影点p_3'，因此根据真实位置和估计位置，位置误差可以很容易计算出，即图中的 r。

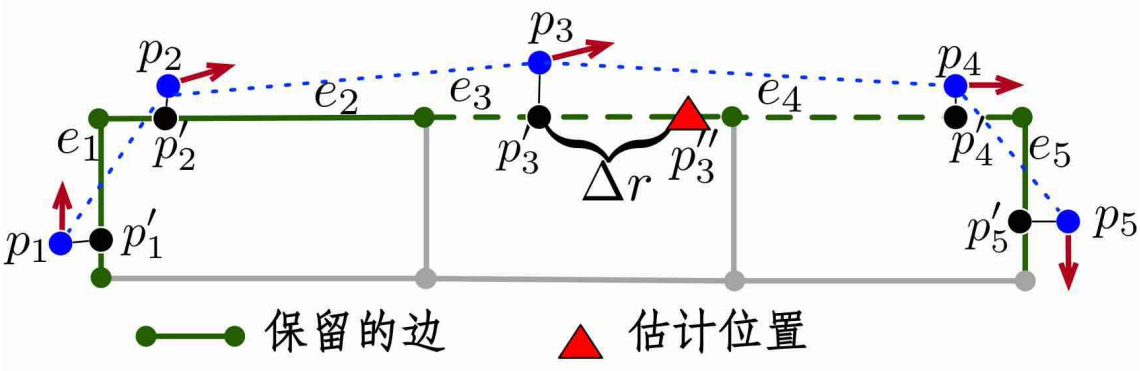


图 3.5　车辆位置估计和误差测量的说明示例

Figure3.5 Illustrative example of location estimation and error measurement in *when-and-where* query

3.3.3 轨迹解压

对给定的压缩轨迹，恢复其原始轨迹是很容易的。在图 3.6 所示的例子中，压缩轨迹为 $_{c2}$ e_1, e_5, e_8 ，第一个边 e_1 和最后一个边 e_8 分别对应原始轨迹的起始和结束边，唯一剩下的边 e_5 是第一个交叉口保留的出边。第二个交叉口没有保留出边，说明车辆在第二个交叉路口直行，即从边 e_4 行驶到边 e_8 。因此，我们可以正确地解压压缩轨迹。作为对比，压缩轨迹 $_{c1}$ e_1, e_8 ，只保留第一个和最后一个边，这表明汽车在交叉路口在最常见的出边处转了弯。

虽然直接嵌入到压缩轨迹中的语义信息很少，但通过与其他多源城市数据集进行耦合，轨迹解压能够揭示一些常见的语义信息[64,65]。例如，可以通过将总的行驶距离(每条边的总和)除以总时间来计算轨迹片段的平均速度，平均速度是交通状态的重要指标[11,20,32,76-78]；如果起始边和结束边是唯一的，则可以声明车辆处于驻停状态；此外，利用兴趣点和数字地图数据，我们还可以推断出轨迹周围的空间信息(如街道名称、车道数等)[20,65,74]。

3.4 实验评估

在本节中，我们基于从真实环境中收集到的 GPS 轨迹数据，进行了广泛的实验评估，以证明 HCC 算法的有效性和高效性。具体来说，首先评估了 HCC 算法的压缩率高低和计算时间长短；其次，检查了轨迹片段的长度边界；最后，测试了在不同密度的路网下和不同采样率时 HCC 算法的性能，以及评估了时间-地点查询的响应质量。

3.4.1 实验设置

① 基准算法 针对轨迹压缩，我们选择三个基准方法（即 PRESS、CCF 和 DP 算法）来比较 HCC 算法的性能，它们的详细信息如下。

PRESS 算法 对于给定的匹配轨迹片段，压缩轨迹中每对连续的边完全遵循最短路径。读者可以阅读文献[19]来了解详情。

CCF 算法 对于给定的匹配轨迹片段，CCF 只保留每个交叉路口的出边的 ID 和它的编码（以顺时针顺序）。读者可以阅读文献[17]来了解详情。

DP 算法 它基于线段简化算法，旨在在减少原始轨迹中 GPS 点的数量。如果 GPS 点到线段的距离小于误差界限，那么该 GPS 点要丢弃。因此，压缩率深受用户指定的误差边界的影响[48]。

备注：PRESS 和 CCF 算法都是基于地图匹配并且是空间无损的，而 DP 算法是基于原始轨迹片段（即 GPS 点序列）但它是空间有损的。类似于 HCC 算法，对于 PRESS 和 CCF 算法，轨迹片段的起始边和结束边也始终要保存。为了保证 PRESS 算法的计算速度，路网中任意两条边的最短路径通常是预先计算并且提前

存储好的。但由于最短路径的存储空间太大（例如北京市的最短路径存储空间超过了 100 GB），这在移动环境中是无法接受的。因此，为使其能在移动环境下工作，我们修改了原始的 PRESS 算法，即实验中的 PRESS 算法是修改版，具体是采用 Dijkstra 算法在线计算最短路径。

② **数据准备** 在实验中我们使用了来自中国北京市的两种不同类型的数据集，包括一个路网数据集和一个 GPS 轨迹数据集。路网数据集可以从 OpenStreetMap 免费下载和提取[72,79]，总计包含 141735 个节点和 157479 个边。GPS 轨迹数据集是 595 辆出租车在一周内（2015 年 9 月 15 日至 21 日）生成的，数据集的采样率是相同且恒定的，采样间隔值约为 6 秒，该数据集的大小超过 1.01 GB。

③ **评估指标** 我们使用压缩率（记为 *cr* ）来测量压缩算法的效果，其定义为原始轨迹的存储空间与压缩轨迹的存储空间的比值。很容易理解，如果 *cr* 越大，算法的压缩性能会更好。我们使用轨迹匹配阶段和压缩阶段的总耗时来评估轨迹压缩算法的效率。

3.4.2 有效性评估

轨迹片段的长度（ *l* ）是 HCC 算法中的一个重要参数，且是用户指定的，因此我们研究了在不同 *ls* 下算法的有效性（即压缩率的高低）。作为比较，我们给出了基准算法的压缩结果。

变化 *l* 图 3.6 展示了在不同 *l* 下不同压缩算法的对比结果。我们观察到 HCC 算法的压缩率随着轨迹片段的长度增长而增加，这与定理 3.1 一致。并且，其他三种算法的压缩率也随 *l* 的增长而增加。

从图 3.6 中可以看出，在不同 *ls* 下，所有算法的压缩率都比 DP 算法高得多，这证明了基于地图匹配的轨迹压缩算法有着卓越的性能。HCC 算法的压缩率介于基于地图匹配的三种算法之间，而 PRESS 算法的压缩表现最佳。HCC 算法的表现优于 CCF 算法，这是因为：CCF 算法保存了每个交叉口的出边信息，而 HCC 算法只保留有明显航向变化的出边信息，这些边只是所有出边中的小部分。PRESS 表现最好的原因可能是，在实际生活中，出租车司机普遍喜欢选择最短路径来运送乘客。在这种情况下，起始点和目的地中间的边可以被丢弃，因此产生最高的压缩率。但 PRESS 算法有一个主要的缺点，整个路网中任意两个边之间的最短路径应预先计算并存储起来，以节省压缩时间，这是一个非常耗时的过程。更糟糕的是，一旦城市路网发生更新，这个过程必须重新执行，因此在可持续维护上会产生一系列问题。

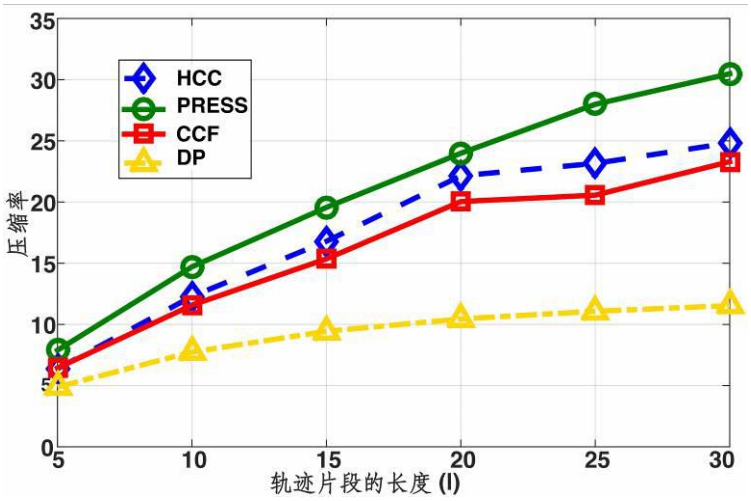


图 3.6 不同 ls 下不同轨迹压缩算法的压缩率

Figure 3.6 Comparison results of compression ratio for different algorithms under different ls

3.4.3 效率评估

类似于有效性评估，我们还研究了在不同 ls 下 HCC 算法的时间成本。请注意，此处的时间成本是指两个阶段的总时间，包括轨迹匹配和轨迹压缩的总计算时间。为了了解算法在不同阶段的时间成本，我们进一步给出了详细的时间分布结果。在这个实验中我们设定 k 6。

变化 l 图 3.7 展示了不同 ls 下 HCC 算法和基准算法的时间成本对比结果。当轨迹片段的长度变长时，所有算法都需要更长的计算时间。因为 DP 算法不需要地图匹配，所以它是节省时的。对于其他三种基于地图匹配的算法，正如预测的那样，在所有的 ls 下，PRESS 算法时间开销最大，这是因为它需要在线计算最短路径，因此比其他三种算法都要耗时。HCC 算法比 CCF 算法更高效，原因 CCF 算法需要进行附加操作，即搜索出边对应的编码，这个过程需要耗费一定时间。为了确保及时响应，CCF 算法对每一个交叉路口的出边提前按照顺时针顺序编码并保存在表格。通常，交叉路口的数量在路网中是巨大的，查询出边在编码表格对应的编码这项额外操作需要计算成本开销且是低效的。因此，结合图 3.6 和图 3.7 可以看出，HCC 算法在压缩轨迹时能在压缩率和时间成本开销之间做出很好的权衡。

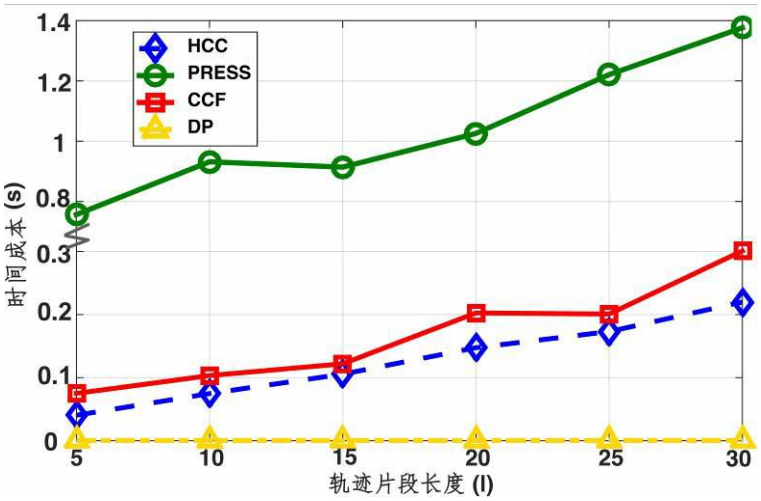


图 3.7　不同的 ls 下不同轨迹压缩算法的时间成本

Figure 3.7 Comparison results of time cost for different algorithms under different ls

不同阶段的时间分布　表 3.1 展示不同 ls 下 HCC 算法在轨迹匹配和轨迹压缩两个阶段时间成本的百分比。可以看出，HCC 算法的大部分时间都消耗在轨迹匹配阶段。例如，对所有轨迹长度 l，该阶段至少消耗了 83％的时间。而且，随着 l 越来越长，阶段 1 的时间成本占用更多。例如，当 l　30 时，阶段 1 的时间成本占总时间的 91％以上。根据统计结果，很容易知道基于地图匹配的轨迹压缩计算时间瓶颈实际上是地图匹配。为进一步加速在线轨迹压缩，需要使用更加有效的地图匹配算法。

表 3.1 不同的 ls 下 HCC 算法中轨迹匹配和轨迹压缩时间成本的百分比

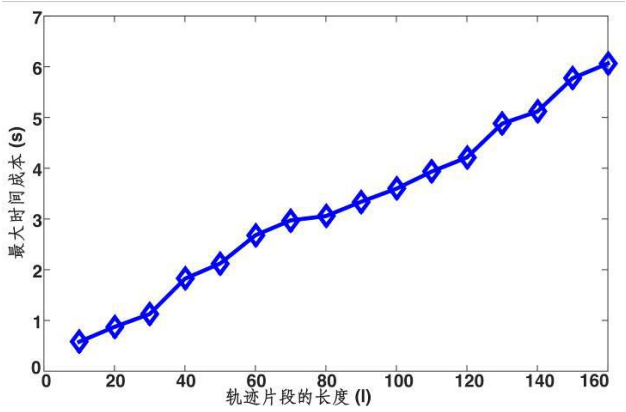
Table 3.1 The percentage of time cost at trajectory mapping and compression in HCC under different ls

ls	5	10	15	20	25	30
阶段 1	83.56	89.91	89.24	89.73	89.77	91.81
阶段 2	16.44	10.90	10.76	10.27	10.23	8.19

3.4.4 边界选择

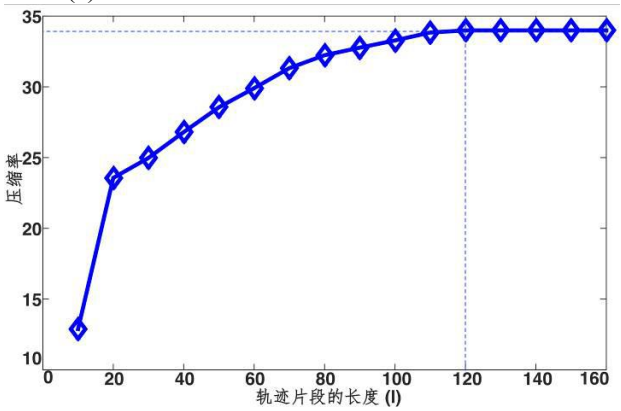
根据图 3.6 的评估结果可知，如果我们设置更长的轨迹片段 l，HCC 算法的性能会变得更好，但需要更长的计算时间，因此我们也对 l 的边界选择感兴趣。在边界情况下，所有轨迹片段都应该在新的 GPS 采样点即将到来之前被压缩，即所有轨迹片段的最大压缩时间成本应小于 GPS 轨迹数据的采样时间间隔（即 6 秒）。因此，为了得到 l 的边界值，我们以相等的间隔增加 l（即每次长度增加 10），并检查相应的最大压缩时间成本。我们增加长度直到最大值时间成本超过 6 秒。

在不同ls下的最大时间成本如图 3.8（a）所示。随着l的增加，最大时间成本几乎呈线性上升。当长度l接近 160 时，最大时间成本还不到 6 秒。我们还检查了将l从 10 变为 160 来压缩算法取得的压缩率。在定理 1 中证明，当l越大时，压缩率的提升是非常有限的。如图 3.8（b）所示，当l从 120 开始增加时，压缩率几乎保持不变。因此，没有必要选择过长的l。在我们的例子中，l的边界选择是 120。在此实验中我们设置k　6。



（a）不同的ls下 HCC 算法的最大时间成本

(a) Maximum time costs under different ls



（b）不同的ls下 HCC 算法的压缩率

(b) Compression ratio under different ls

图 3.8 l的边界选择

Figure 3.8 Results of the boundary choice of l

3.4.5 其他性能评估

不同的路网密度 我们将北京市分为两个区域，即三环内（区域 I）和三环外（区域 II）。区域 I 中每平方公里的路网节点数和边数（即路网密集程度）比区域 II 高了近 6 倍。表 3.2 提供了这两个区域的更多统计数据，包括了 HCC 算法的性能（匹配精度，压缩率和时间成本）。可以观察到，与区域 II 相比，HCC 算法在

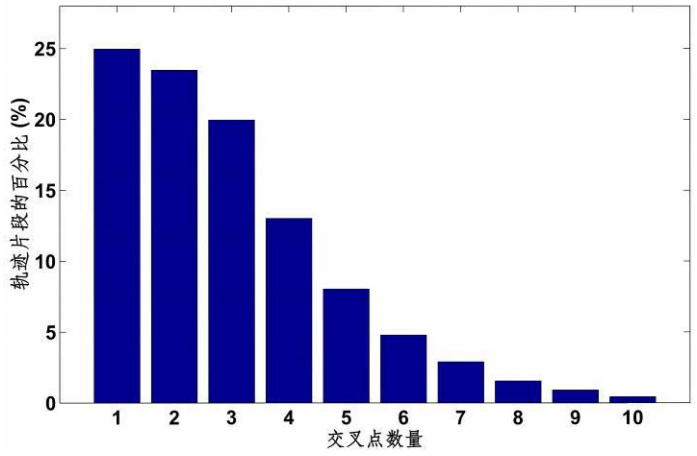
区域 I 中有更高的压缩率。但在路网稀疏的地区（即区域 II），HCC 算法消耗的时间更少。这可能是由于①司机更倾向于在区域 II 的十字路口选择“直行”；②区域 II 中的路网较简单，地图匹配中的路径查找效率更高。在该实验中我们设置 $k \quad 6, l \quad 10$ 。

表 3.2 不同路网密度下的结果

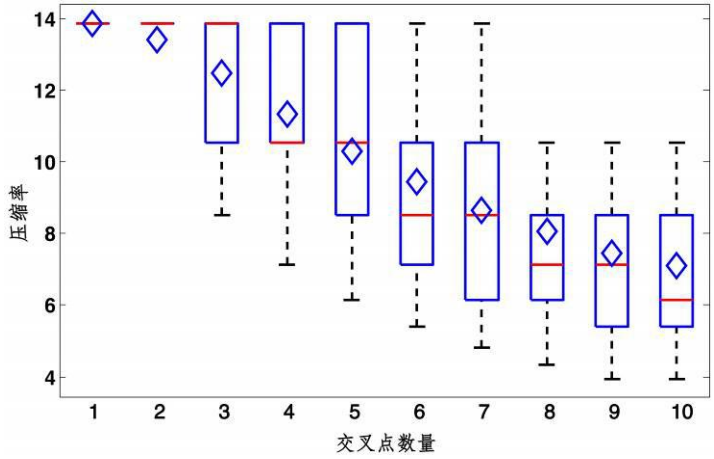
Table 3.2 Results under different road network densities

	区域 I	区域 II
面积（km^2）	334	5992
#节点	38000	103000
#边	45000	112000
压缩率	11.24	11.94
时间消耗（s）	0.102	0.077

不同的交叉路口数量 我们调查了压缩率与交叉路口数量之间的关系。对于每个轨迹片段，结合路网数据我们很容易知道车辆行进中交叉路口的数量。本实验中使用的轨迹片段总数约为 1600000，所有轨迹中交叉路口数的直方图如图 3.9（a）所示。可以看出，大多数轨迹片段中的十字路口数量小于 5。对于拥有相同数量交叉路口的每组轨迹片段，我们计算相应的平均压缩率和其他一些基本统计，结果用箱线图（即第 25 个百分位数，第 50 百分位数，第 75 百分位数，最大值和最小值）表示，如图 3.9（b）。另外，平均值用菱形标记。可以观察到，压缩率的平均值和箱线图中的值通常随着交叉口数量的增加而减少。例如，当交叉路口的数量只有一个时，压缩率非常高。这是因为，压缩率与有着明显航向改变的交叉路口的数量密切相关，当轨迹片段只有一个十字路口时，那么至多保留该路口的一个出边。随着交叉路口数量的增加，交叉路口处车辆航向明显变化的概率越来越大，保存更多出边的概率增加，这导致了压缩率的降低。正如预期的那样，当轨迹片段具有 10 个交叉路口时，压缩率小得多，平均压缩率值约为 7。在该实验中我们设置 $k \quad 6, l \quad 10$ 。



（a）



（b）

图 3.9：交叉路口数量的统计（a）；交叉路口数量与压缩率之间的关系（b）

Figure 3.9 The relationship between the compression ratio and the number of intersections

不同的采样率 为了研究 HCC 算法在不同采样率下的表现，我们采用了一个降低采样率的简单方法，这里采样率指的是 GPS 设备的采样间隔时间。压缩结果如表 3.3 所示，随着采样间隔的增加，HCC 算法的压缩率明显下降。关键原因是原始轨迹数据的存储空间随着采样时间间隔的增加而减小，而压缩轨迹的存储空间保持不变或增大。另外，当采样时间间隔为 150 秒时，压缩率几乎接近 1，这意味着当数据稀疏时，数据压缩的空间非常有限。至于效率方面，压缩具有更高采样率的轨迹数据时，HCC 算法需要更长的计算时间。这是因为随着采样时间的增加，两个连续的 GPS 点之间的行程距离越长，结果中包含越多的不确定性，因此使地图匹配更加复杂。在这个实验中我们设置 $k \quad 6, l \quad 10$ 。

表 3.3 不同采样率下 HCC 算法的压缩性能

Table 3.3 Results under different sampling rates.

采样间隔（s）	6	30	60	120	150
压缩率	14.11	6.27	3.52	1.54	1.13
时间消耗（s）	0.078	0.122	0.427	0.517	0.786

时间-地点查询的响应质量 压缩后的轨迹数据支持一些基于地理位置服务的查询，其中最常见的是“时间-地点查询”，用户可以知道在某时刻车辆的具体位置。若查询输出的位置和车辆的真实位置距离相近，则说明查询的响应质量高。表 3.4 记录了在不同轨迹片段长度下，查询结果的平均距离误差。从表中可以看出，由于车辆的总行驶距离随着 l 的增加而变长，因此驾驶行为也变得更加复杂且难以预测车辆位置，导致查询的平均误差变大。然而，与文献[17,19]的结果相比，HCC 算法仍具有优良的查询响应质量。例如当 l 等于 10 时，我们方法的平均误差小于 145 米，在现实生活中大多数查询服务可以接受该误差。在这个实验中我们设置 k　6, l　10 。

在本章，我们只是简单地处理了时间信息，即保留轨迹片段的起始时间戳。而在第 4 章，我们针对时间维度上的轨迹信息提出了更精巧的压缩方法，能以较小的存储空间保存更多的时间信息。当再次执行时间-地点查询时，查询结果的质量更高。例如当 l　10 时，最大误差已经降低到 30 米，具体细节请参阅第 4 章。

表 3.4 不同长度 l 下时间-地点查询误差

Table 3.4 Results of mean error in when-and-where query under different ls

ls	5	10	15	20	25	30
平均误差	112.8	144.7	187.9	200.5	236.4	271.3

3.5 章节小结

在本章中，为了寻找简洁紧凑的空间轨迹表示，我们提出了基于交叉口航向变化的空间轨迹压缩算法，即 HCC 算法。我们使用中国北京市的真实数据集将设计的算法与当前优秀的基准算法进行实验对比，证明了 HCC 算法在轨迹压缩的效率和质量方面都具有卓越的性能。

4　轨迹压缩：时间维度

本章研究在时间维度上对轨迹进行高效压缩。内容安排如下：4.1 节是本章的前言，介绍了在时间维度上的轨迹压缩现状和本章的主要技术贡献；4.2 节介绍了一些主要概念，并对系统进行了概述；4.3 和 4.4 节详细介绍了时间轨迹数据表示、数据压缩和解压缩算法；4.5 节全面评估了压缩框架的性能；最后，4.6 节对本章进行了总结。

4.1 前言

近年来，随着全球定位系统(GPS)设备和移动互联网的普及和成熟，包括出租车、公交车、物流货车等在内的 GPS 车辆越来越多[11,13,32,41,80]。它们产生了大量的 GPS 轨迹数据，不可避免地造成了可持续的通信和存储成本。在车辆上安装 GPS 装置的最初目的之一主要是为了车队管理，即了解其实时位置，用于监控、跟踪和调度等[30,74]。因此，包括空间坐标、车速在内的信息需要实时传输到数据中心。为了保证管理质量，车辆经常以远超必要的频率向数据中心报告其位置，这不仅会产生大量的传输和存储开销，而且会加重数据中心端的计算负担。更具体地说，由于 GPS 设备存在定位误差[55,56]，为提高原始数据的准确性，研究者通常采用地图匹配来识别车辆的“真实”位置。由于大量数据被发送到数据中心，因此需要更多的计算资源来保证及时响应。

通过减少报告位置的频率或在发送到数据中心之前丢弃一些位置信息来减少数据是一种常见的解决方案。显然，这种简单粗暴的方法在实际应用中是有问题的。这是因为，1）保留的两个连续 GPS 点之间的距离差距可能太大，无法解锁车辆的具体位置[66,81]；2）保留点的定位质量是随机的，这使得车辆管理(如跟踪任务)极具挑战性。为了避免上述问题，目前收集到的数据是通过云端数据中心进行数据压缩，但这种离线操作只能降低存储成本。更糟糕的是，目前几乎所有的数据约简方法都直接忽略了时变速度信息，而这些信息在了解驾驶风格、探测路况等方面发挥着至关重要的作用[34,82]。并且，由于速度是基于多普勒效应检测出来的，即使能轻易推断出两个连续 GPS 点之间的平均速度，也不能将其视为冗余信息[83]。因此，时变速度信息应单独进行处理。

本章的目标是开发一个在线有效的数据压缩框架，其原理是在设备端收集更多的原始数据，但向中心发送少量并由人工控制的数据。具体来说，与以前的解决方案相比，本文的框架具有以下创新性和独特性。

从离线到在线 本文在车辆端进行数据压缩。通过引入移动边缘计算的原理，可以同时降低通信和存储成本[51,84]。随着移动设备(如智能手机)计算能力的提高，这种解决方案是可行的。一方面，无论是通过 WiFi 还是蓝牙，GPS 设备与驾驶员智能手机之间的数据传输都是高效、免费的；另一方面，以前在数据中心端完成的数据压缩任务可以很容易地转移到智能手机上。因此，将压缩后的数据发送到数据中心，能减少数据传输中的通信开销并降低云端存储数据的成本。

与速度相关的全新轨迹表示 本文提出了一种充分考虑所有时变信息的数据表示方法。与[19,28,53,85]相似，本文也将原始轨迹分为两个部分，并对其分别进行数据压缩。一般情况下，空间轨迹由路网中一系列连通边表示，这些边记录了车辆的出行路径，它们只提供空间信息，空间轨迹的压缩方法已经在第三章节进行了阐述；时间轨迹由时间-距离曲线表示，该曲线表示车辆沿行进路径行驶时的时间信息。空间轨迹压缩主要基于地图匹配，并且是无损的。而时间轨迹压缩通常基于线段简化，且是有损的[19,81]。由于研究人员对时间轨迹压缩的关注相对较少，本章主要关注时间轨迹压缩。并且，时变速度信息在本章节中也得到了很好的处理和表示。在此基础上，从寻找一种更精确更完整的数据格式和表示入手，提出了一套新的误差约束的数据压缩算法。由于从数据中心端的压缩数据可以有效地推断出车辆的实时位置，便于监控、跟踪、调度等任务。因此，本文的压缩方法可以支持多种查询，包括地点查询(即，车辆在指定时间的位置)和时间查询(即，当车辆位于给定的已访问位置时)；此外，由于新数据表示还处理时变速度信息，因此还可以支持在给定时间间隔下查询速度方差和查询感兴趣道路的交通状况。此外，还可以支持更多潜在的时空数据挖掘应用，如隐式嵌入到时变速度信息中驾驶员的驾驶风格或爱好[5,11,32]。

4.2 主要概念和算法概述

定义 1(路网) 路网是一个图$G(N,E)$，它由边集E和节点集N组成。N中每个元素n是包含经纬度坐标的点，来表示其空间位置。E中的每条边e_i都有两个节点(即，n_a和n_b)，以及$d(n_a,n_b)$和$d(n_a,n_b)$两个边方向。$d(n_a,n_b)$表示节点n_a到n_b的边方向，可以根据节点的经纬度很容易计算出来。在本例中，n_a为头节点，n_b为尾节点。图 4.1 中的$d(n_1,n_2)$是从n_1到n_2的一个边方向。

定义 2 (GPS 条目) GPS 条目记录了运动物体的时空信息。一个条目r_i通常由五个元素组成，即一个时间戳t_i，一个地理空间坐标(x_i,y_i)，瞬时速度v_i和一个以正北方作为方向基准的车辆航向$h_i(0^{\circ} \le h_i < 360^{\circ})$。如图 4.1 所示，$h_1$为在时间戳$t_1$时刻车辆的行驶方向。$r_i$由$r_i$ $(t_i,lat_i,lon_i,v_i,h_i)$来表示。其中时间戳和地理空间坐标定义为 GPS 点，记为p_i (t_i,lat_i,lon_i)。

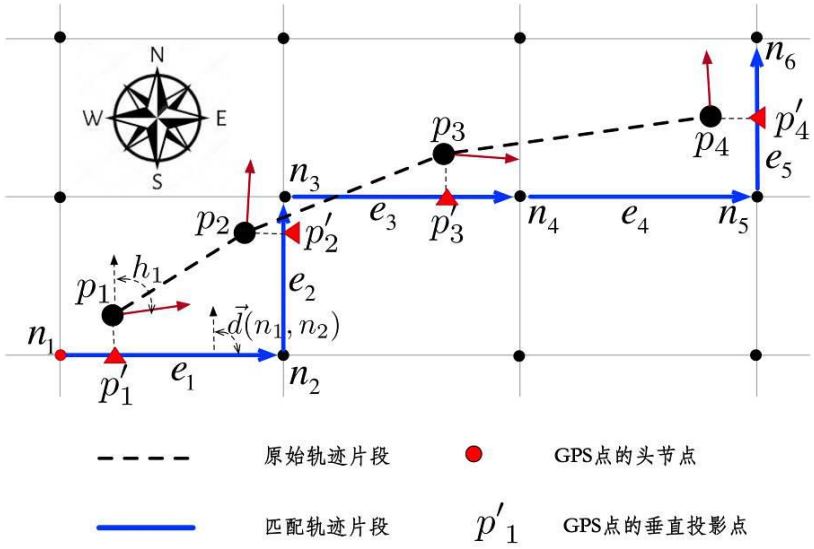


图 4.1　主要概念的例证

Figure 4.1 Illustration of main concepts used in this paper

定义 3(原始轨迹片段) 原始轨迹片段 Tr 是一组按时间顺序排列的 GPS 条目的序列，由 Tr　r_i, r_{i+1}，　，r_{i+l} 表示。l 是用户指定的参数，它控制原始轨迹片段的长度。

定义 4(匹配轨迹片段) 给定一原始轨迹片段 Tr，其匹配轨迹片段 $\overline{Tr}$ 是车辆在道路网络中真实行驶的路径。Tr 由节点序列 n_i, n_{i+1}，　，n_m 表示，节点序列中任意两个连续的节点 (n_i, n_{i+1}) 由一条唯一的边相连接。

为了得到原始轨迹片段对应的匹配轨迹片段，可以求助于地图匹配算法，如第二章节介绍的 SD-Matching 算法。

定义 5(两点之间的网络距离) 两点之间的网络距离定义为路网中从一点到另一点的空间路径长度。如果 GPS 点不坐落在边上，则点 p_i 与点 p_{i+1} 的网络距离定义为从匹配轨迹片段上 p_i 的投影点 p_i' 到 p_{i+1} 的投影点 p_{i+1}' 的路径长度。在本章的余下部分，网络距离被简称为距离。

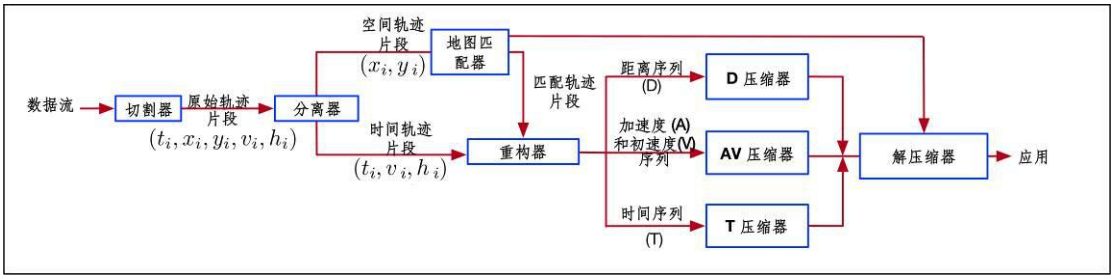


图 4.2 DAVT 算法框架

Figure 4.2 The framework of DAVT system.

如图 4.2 所示，我们提出了一种在时间维度上对原始轨迹片段进行压缩（即对时间轨迹进行压缩）的算法框架，称为 DAVT，由切割器、分离器、地图匹配、重构器以及 D、AV 和 T 三种压缩器等七个组件构成。切割器首先将原始 GPS 轨迹数据流分为非重叠且等长的轨迹片段，每个原始轨迹片段都是数据压缩的基本数据单元；分离器将每段分解为一对空间轨迹片段和时间轨迹片段。对于空间轨迹片段，利用地图匹配得到它的匹配轨迹片段；将所得的匹配轨迹片段和时间轨迹片段输入到重构器中，重构器将返回一个新的时间轨迹表示，这种表示由三种序列组成，即距离序列(D)、加速度和初速度序列(AV)、时间序列(T)(详情见第 4.3 章)；然后分别用 D、AV、T 压缩器降低每个单元的空间大小(见第 4.4 章)，最后得到紧凑的时间轨迹片段。在支持车辆实时管理和其他基于离线位置服务的应用之前，应该先解压数据(参见第 4.4 章)。

4.3 DVAT 轨迹表示及优点

对于一个原始轨迹片段 Tr，其全新的时间轨迹采用以下 DAVT 格式，表示为 $Tr \quad D, AV, T$ 。

D（时间轨迹片段的距离序列） 使用序列 $D\ Tr$ 来记录车辆的准确位置。与记录二维坐标不同，我们直接用一维坐标来表示位置。具体来说，沿着路网中匹配的轨迹，我们记录从 Tr 的起始节点到任意 GPS 点 p_i 的累计距离来表示第 i 个采样时刻车辆的实际位置，这样的距离称为 L_i。我们也用 d_i 表示两个相邻点 $p_{i\ 1}$ 和 p_i 之间的距离。d_1 是一个例外，它表示 p_1 到空间路径起始节点的距离。因此，数学方程式（4.3）成立。如果保留所有的 d 元素，那么车辆在任意采样时刻的位置 L_i 可以递归地推断出来。

$$L_i = L_{i\ 1} \quad d_i \quad d_1 \qquad {}^{j\ i}_{j\ 2} d_j \tag{4.3}$$

由于基于我们的观测，d_i 会随 GPS 采样时间间隔变化，我们将其归一化为相对距离。具体做法是将 d_i 除以时间间隔的平方 $t_i \quad t_{i\ 1}{}^2$，用 $r\ d_i$ 来表示相对距离，因此距离序列可以表示为 $D\ Tr \quad r\ d_1, r\ d_2, \cdots, r\ d_l$ 。同样，$r\ d_1$ 等于 d_1 与采样时间间隔的平方之比。如果距离序列 $D\ Tr$ 包含一些相同的距离值，那么它包含重复元素并且可以通过编码压缩进一步减少该序列的存储空间。

AV（时间轨迹片段的加速度和速度序列） 用序列 $A\ Tr$ 和序列 $V\ Tr$ 记录原始轨迹片段中另外两个时间维度的信息，即平均加速度和瞬时速度。其中，$A\ Tr$ 由连续两个点 p_i 和点 $p_{i\ 1}$ 之间的平均加速度 $\overline{a_i}$ 序列组成，表示为 $A\ Tr \quad \overline{a_1}, \overline{a_2}, \quad , \overline{a_{l\ 1}}$，其中 $\overline{a_i}$ 定义为两个连续 GPS 点(即点 p_i 和点 $p_{i\ 1}$)的瞬时速度差与时间间隔的比值。$V\ Tr$ 只存储原始轨迹片段的初始速度 v_1，这是因为其

他速度可以通过结合将保留的速度v_1、加速度a_i和时间戳t_i推导出来，如式(4.1)所示。

$$v_i \quad \begin{matrix} v_1, i \quad 1 \\ v_1 \quad {}_{j\ 2}^{j\ i}(t_j \quad t_{j\ 1}) \quad \overline{a_{j\ 1}}, i \quad 2 \end{matrix} \tag{4.1}$$

T(时间轨迹片段的时间序列) 用序列$T\ \ Tr$记录原始轨迹片段的时间戳，用$T\ \ Tr = t_1, t_2, \quad , t_i$表示。时间戳的表示有助于关联 D 和 AV 序列，这是因为它们中的每个元素都受到时间约束。例如，序列$D\ \ Tr$中的d_i与序列$T\ \ Tr$中的t_i是一一对应的。通过将$T\ \ Tr$与其他序列相结合，可以支持常见的基于位置的查询服务，这将在下面的论述中介绍。

综上所述，采用 DAVT 格式的时间轨迹片段可以表示为
$Tr \quad (r\ \ d_1, r\ \ d_2, \quad , r\ \ d_l, a_1, a_2, \quad , a_{l\ 1}, v_1, t_1, t_2, \quad , t_l)$。

与已有的表示形式相比，由于 D 和 AV 格式的效率高且信息熵小，因此使用 D 和 AV 格式更有利于编码压缩和查询。此外，这种表示还能支持常见的基于地理位置的查询服务，具体如下。

D 的优点 与 TED 和 PRESS 相比，我们提出的距离表示$r\ \ d_i$不仅利于压缩，而且便于查询。在文献[60]中，TED 使用相对距离$r\ |p_i|$。具体来说，给定一个落在边e_i上的点p_i，$|p_i|$定义为点p_i与定位边e_i的头节点之间的距离。由于边e_i的长度变化较大，因此 TED 将$|p_i|$归一化得到相对距离，具体做法是 TED 将$|p_i|$除以边e_i的长度，得到相对距离$r\ |p_i|$。这种表示方式强调了每个点的自身位置，但忽视了在空间中与其他点的关联关系，这可能会降低查询效率，例如查询在一段时间内物体的移动距离。在这种情况下，TED 在计算距离之前必须推断出这段时间的真实行进路径。更糟糕的是，如果用户指定的时间过长，推断真实路径的过程会非常耗时。

另一个表示，即 PRESS，它使用d_i'表示 GPS 点的位置信息。d_i'定义为自原始轨迹片段开始，车辆在时间戳t_i处行驶的累计距离。实际上，如式（4.2）。很容易看出，随着时间戳的变大，序列中的d_i'始终在增加，这说明序列中几乎不包含重复信息，因此该表示方法具有较大的信息熵值。因此，这种位置表示方法对于编码算法来说是不可压缩的。为了更进一步验证我们的想法，我们从理论上分析了不同位置表示方法的熵值高低。具体做法是，我们使用 1000 个轨迹片段样例来计算它们的平均熵。每个表示方法的熵可由式（4.3）计算，其中s_i是距离序列中的第i个符号，$f\ \ s_i$是s_i在所有符号中的百分比[86]。结果如表 4.1 所示，可以看出，d_i'的熵最大，而其他两种表示形式的熵值是相似的。因此，我们可以得出一个结论，我们提出的表示$r\ \ d_i$更有利于编码压缩。

$$d_i' \qquad {}^{j\ i}_{j\ 2} d_j \tag{4.2}$$

$$^{l}_{i\ 1} l_i f\ \ s_i\ \ log_2\ \ f\ \ s_i \tag{4.3}$$

表 4.1 不同位置表示方法的信息熵值

Table 4.1 Entropy of different location representations

#距离序列	d_i'	$r(\lvert p_i \rvert)$	$r(d_i)$
熵值	3.3219	3.1090	3.0346

AV 的优点 AV 序列以一种利于压缩的方式丰富了现有的时间轨迹表示，也可以支持与加速度和瞬时速度相关的查询。据我们所知，现有的表示法只关注 GPS 点，而忽略了 GPS 条目，因此也损失了重要的时变速度信息。连续的瞬时速度可以捕捉细粒度的用户驾驶风格并更好地检测交通状况。在工作中，我们使用 AV 序列来清楚地表示时变速度信息及加速度信息。同样，所提出的 AV 序列可能包含相同的元素值，因此它也是利于压缩的。

支持常见的基于地理位置的查询服务 这里，我们展示了时间轨迹表示方法可以支持的四种常见的基于地理位置的查询。注意，基于这些常见查询，还可以通过不同查询组合来支持其他高级复杂的查询。另外，如果未压缩的时间轨迹表示方法支持这些查询，那么压缩后的轨迹表示方法也可以通过解压数据来支持相同的查询。

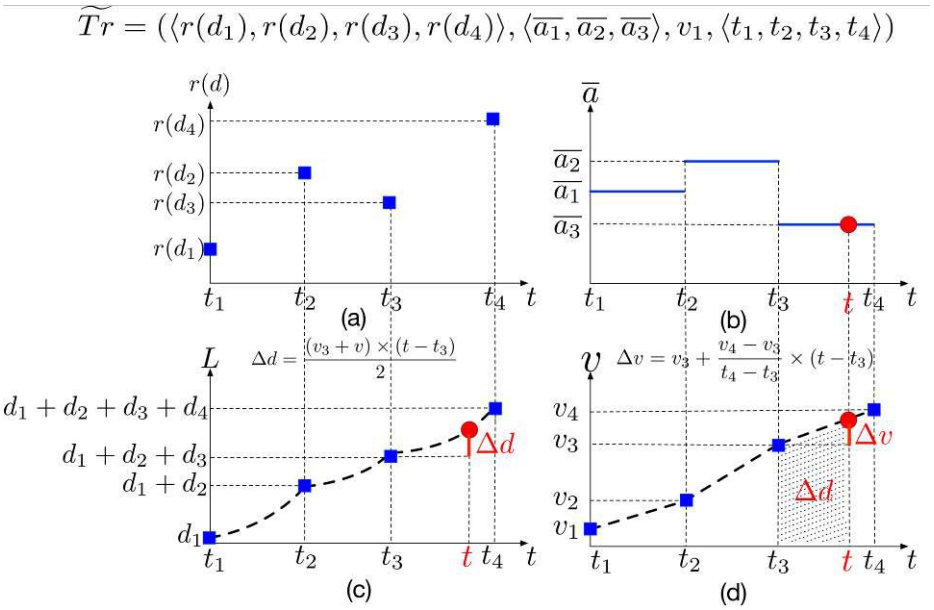


图 4.3: 支持常见的基于地理位置的查询服务的示例

Figure 4.3 Illustrative examples of supporting various common LBS queries

① 加速度查询 查询命令 $acc\ \ Tr,t\ $ 返回 t 时刻运动车辆的加速度值 a。我们假设在任意两个连续时间戳内车辆将以均匀加速度移动，特别是当时间间隔很小的时候，这种假设是非常合理的。我们首先确定时刻 t 所处的时间区间(如 $t_i,t_{i\ 1}$)，利用区间内的平均加速度 $\overline{a_i}$ 近似 a。例如，在 t 时刻的加速度 $(t_3\ \ t\ \ t_4)$ 为 $\overline{a_3}$，结果如图 4.3(b)所示。

② 速度查询 查询命令 $velocity\ \ Tr,t\ $ 返回给定 t 时刻运动车辆的速度值 v。与加速度查询类似，我们首先根据式(4.1)计算时间戳 t_i 和 $t_{i\ 1}$ 的瞬时速度 v_i 和 $v_{i\ 1}$，其中 $(t_i\ \ t\ \ t_{i\ 1})$。因此，可以通过 v_i 和 $v_{i\ 1}$ 之间的线性插值来计算 t 时刻的瞬时速度 v。在上一个例子中，在 t 时刻的 v 等于 $v_3\ \ \ v$，结果如图 4.3(d)所示。

③ 地点查询 查询命令 $where\ \ Tr,t\ $ 返回车辆在给定 t 时刻所处的位置 $p\ \ \ x,y\ $。点 p 的计算包含三个步骤:我们首先要获得匹配轨迹片段的起始点与点 p 的距离，用 L 来表示，基于 L 可以很容易地将 p 定位在其所在的边 e_i 上，同时也可以得到点 p 与边 e_i 头节点的相对距离 $r(|\ p\ |)$，最终得到点 p 的坐标 (x,y)，如式(4.3)所示。其中 x_a,y_a,x_b,y_b 为边 e_i 的两个节点的地理空间坐标。

$$\begin{array}{l} x\ \ x_a\ \ r(|\ p\ |)\ \ (x_b\ \ x_a) \\ y\ \ y_a\ \ r(|\ p\ |)\ \ (y_b\ \ y_a) \end{array} \tag{4.3}$$

补充说明如何计算 L。计算 $t\ (t_i\ \ t\ \ t_{i\ 1})$ 时刻的 L 可分为两步，第一步是根据 D 序列先求出所匹配轨迹片段起点到点 p_i 的距离，记为 $L_i\ \ (d_1\ \ \ \ )$。第二步是计算点 p_i 与 p 之间的距离，记为 $\ d$。例如，t 时刻落在区间 $\ t_3,t_4\ $ 内，其 L 等于 L_3 (即 $d_1\ \ d_2\ \ d_3$)和 $\ d$，如图 4.3(a)和(c)所示。

④ 时间查询 查询命令 $when(Tr,\ x,y\ )$ 返回车辆在路网中给定位置 $p\ \ \ x,y$ 处的时刻 t。我们首先计算位置 p 与起始点之间的距离，记为 L。我们可以很容易知道 L 落在距离间隔 $[L,L_{i\ 1}]$ 上，这表明车辆需要 $t\ \ t_i$ 时间从位置 p_i 到位置 p，且这段时间的行驶距离长度为 $L\ \ L_i$。此外，从点 p_i 到 p 的初速度和匀速加速度分别为 v_i 和 a_i。因此，我们可以得到一个数学方程式（4.4），解方程后得到时间 t 如式（4.5）所示。。

$$L\ \ L_i\ \ \frac{1}{2}\ \ 2v_i\ \ a_i\ \ t\ \ t_i \tag{4.4}$$

$$t_i\ \ \frac{v_i\ \ \sqrt{v_i^2\ \ 2a_i\ \ L_i\ \ L}}{2a_i} \tag{4.5}$$

其他复杂的查询命令还可以通过几个基本查询组合来实现。比如查询车辆位于位置 $p\ \ \ x,y\ $ 处的加速度。在这种情况下，可以通过将时间查询和加速查询结合起来得到查询结果。

4.4 DAVT 压缩算法及解压

根据上节可知，时间维度上的轨迹表示由 D、AV、T 三个序列组成，我们的目标是得到简洁紧凑的时间轨迹片段。因此，我们设计了三个压缩器(即，D、AV、T 压缩器)分别降低每个序列的存储成本。

4.4.1 D 压缩器

步骤 1:距离序列的离散化

距离序列$D\ Tr$ 由 $r\ d_1$,$r\ d_2$ ， ,$r\ d_l$ 表示。注意，对于其中任意一个元素$r\ d_i$ ，它理论上可以是一个无限的字符串。为了减少存储空间，我们采用了一种直观的方法，我们基于式（4.6）离散化序列中每一个元素$r\ d_i$ ，从而得到其相应的离散值$r\ d_i$ '，其中 代表散度程度和$i\ \ 2\ r(d_i)/$ 。

$$r(d_i)\quad \begin{matrix} \frac{i}{2}, if\ |r(d_i)\ \ \frac{i}{2}|\ |r(d_i)\ \ \frac{(i\ \ 1)}{2}| \\ \frac{(i\ \ 1)}{2}, if\ |r(d_i)\ \ \frac{i}{2}|\ |r(d_i)\ \ \frac{(i\ \ 1)}{2}| \end{matrix} \tag{4.6}$$

假设一个距离元素$r\ d_i$ 是 1.2345678 和 0.5，其相应的离散值$r\ d_i$ 是 1.25，因为距离序列在离散化后会包含更多的重复元素。为了节省存储空间，离散值还可以进一步被压缩。同时$D(\dddot{Tr})$ $r\ d_1$ ',$r\ d_2$ ', ,$r\ d_l$ ' 替代了原始的距离序列$D\ Tr$ ，我们将$D(\dddot{Tr})$ 输入后续步骤中进行进一步压缩。

注意，离散化将不可避免地导致精度的损失，压缩结果中保留的距离值肯定会偏离真实值。 值越大，距离序列$D(Tr)$ 损失的信息更多。根据具体应用可以接受的最大误差范围，用户可以指定一个合适的 值。

步骤 2:利用霍夫曼树对距离序列进行编码

我们进行了一项基于大样本距离序列的观测研究。对于每个距离序列，我们发现序列中只包含有限数量的唯一值，换句话说唯一值的数量远小于l。例如，当 和l分别设置为 1.5 和 10 的时候，超过 70%的距离序列只有 4 个惟一值。这个有趣的现象对于我们压缩数据来说，采用霍夫曼编码是最完美的选择，因为该方法是众所周知的对重复数据进行编码的最优方法。我们利用霍夫曼编码压缩距离序列中的每个元素$r\ d_i$ '，并得到压缩后的结果$r\ d_i$ "。最终新的压缩序列$D\ \dddot{Tr}$ 可以表示为 $r\ d_1$ ",$r\ d_2$ ", ,$r\ d_l$ " 。

霍夫曼编码的主要原理是利用变长二进制编码对序列中的每个元素进行编码。元素在序列中出现的频率越高，预期用的二进制编码就越短。对于一个输入的距离序列$D(\dddot{Tr})$，我们简要介绍霍夫曼编码的过程，主要包括霍夫曼树的构建(见算法 4.1)和基于构建的霍夫曼树进行元素编码。

霍夫曼树构建 算法 4.1 总结了霍夫曼树的构建过程。输入一个距离序列 $D(\dddot{Tr})$，输出序列 $D(\dddot{Tr})$ 对应的霍夫曼二叉树。第 1~3 行是算法的初始化，它为树的构建做了准备工作。第 4~9 行是一个循环，演示了树的构建过程。具体来说，我们首先创建一个集合 R 来存储输入序列 $D(\dddot{Tr})$ 中的唯一值 $r\ d_i\ '$ (第 1 行)，然后创建集合 F 存储唯一值 $r\ d_i\ '$ 的频率 $f\ r\ d_i\ '$ （第 2 行）。接下来，我们创建一个集合 S 来存储霍夫曼树中的节点(第 3 行)。集合 S 中第 i 个节点有两个属性分别命名为 $s_i.r$ 和 $s_i.f$，它们表示节点的 ID 和权重值，其取值分别为 $r\ d_i\ '$ 和 $f\ r\ d_i\ '$。初始化结束后，循环开始，循环直到集合 S 中只有一个节点时才结束(第 4 行)。详细来说，我们首先选择集合 S 中频率最小的两个节点 s_x 和 s_y (第 5 行)。其次，基于两个节点 s_x 和 s_y，我们创建它们的父节点 s_z。节点 s_z 是 s_x 和 s_y 频率的总和。最后，我们从集合 S 中删除节点 s_x 和 s_y，并在其中添加节点 s_z (第 8～9 行)。

算法 4.1 霍夫曼树构建
输入： $D(\dddot{Tr})$　$r(d_1)',r(d_2)',\ \ ,r(d_l)'$.
输出： $D(\dddot{Tr})$ **对应的霍夫曼树**
1：创建一个集合 R　$\{r(d_i)',i\ \ 1,\ \ ,m\}$.
2：创建一个集合 F　$\{f(r(d_i)'),i\ \ 1,\ \ ,m\}$.
3：创建一个集合 S　$\{s_i \mid s_i.r=r(d_i)',s_i.f\ \ f(r(d_i)),i\ \ 1,\ \ ,m\}$.
4：while $\mid S\mid !\ \ 1$ **do**
5：　选择两个频率最小的节点 s_x **和** s_y
6：　构造一个父节点 s_z**，它的子节点分别是** s_x **和** s_y
7：　$s_z.f\ \ s_x.f\ \ s_y.f$，$s_z.r\ \ Null$
8：　从集合 S **中移走** s_x **和** s_y
9：　把 s_z **添加在集合** S **中**

假设一个距离序列样本 $D(\dddot{Tr})$ 为 $6,6,6,6,6,6,5,7,7,8$，其集合 R 和 F 如图 4.4 左图所示，距离序列的霍夫曼树如图 4.4 右图所示。在树中，一个圆圈表示一个节点 s_i，圆圈中的数字表示该节点的频率权重(即 $s_i.f$)。红色数字表示节点的 ID(即 $s_i.r$)。为简单起见，树中隐藏节点名称为 *Null* 的 ID。

元素编码 根据所构造的霍夫曼树，我们对距离序列 $D(\dddot{Tr})$ 中每个元素进行编码。具体来说，$D(\dddot{Tr})$ 中的唯一值都由树中的一个叶节点来表示，对于某个叶节点，它对应的二进制编码就是从根到自身的路径所对应的二进制代码，如图 4.4

即从根节点到叶节点的蓝线。作为一种常见的约定，二进制 0 代表左链路，而二进制 1 表示右链路。重新回到距离序列样本 $D(Tr)$ $6,6,6,6,6,6,5,7,7,8$ ，右图虚线框中显示了 $D(\ddot{Tr})$ 中的唯一值对应的二进制编码，因此距离序列 $D(Tr)$ 被压缩为 $D(Tr)$ 1111110000101001 。如果 $D(\ddot{Tr})$ 中每个元素（十进制）都用 4 个比特表示，经过编码后，压缩后的距离序列从原先占用 40 比特减小到 16 比特。

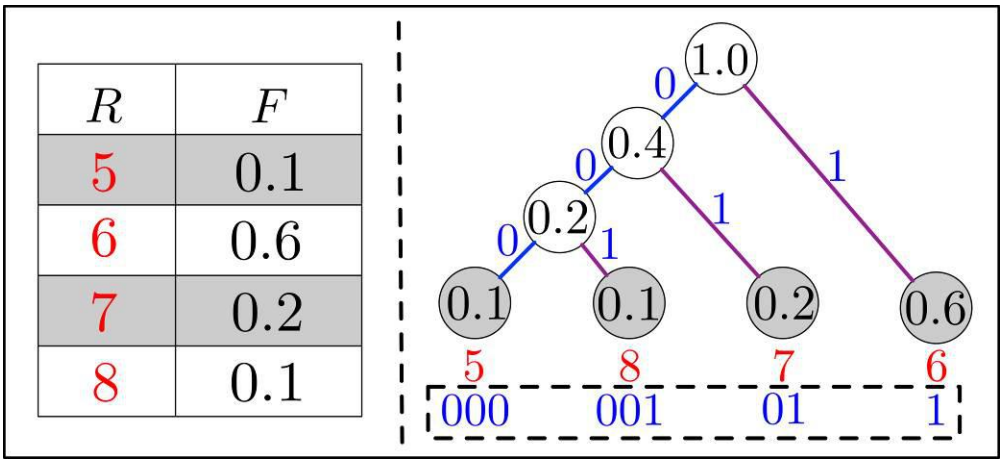


图 4.4 霍夫曼树构建和元素编码的一个示例

Figure 4.4 An illustrative example of the Huffman tree building and element encoding.

步骤 3:通过霍夫曼森林对所有距离序列进行编码

有两种简单的方法可以对所有距离序列进行编码。第一个方法是把所有的序列放在一起，根据组合的长序列建立一个统一的霍夫曼树。然而，这种情况下建立的树必定包含大量的节点和层。这是因为，长距离序列的值范围要大得多，而且还包含大量惟一值。另一个方法是，我们可以为每个距离序列建立一个霍夫曼树。但是，这种简单的方法将产生大量冗余的树，造成计算和内存资源的极大浪费，特别是在移动环境中，这个问题非常明显。可见，这两种直观的方法在实践中并不可行。为了减少潜在的问题，我们提出建立一个包含有限数量的霍夫曼树的森林，其中每棵树只负责对所有距离序列中的一小部分进行编码。我们将在下一节中证明我们的方案是可行的。

根据定义，距离序列中元素 $r\ d_i$ '的值取决于两个连续 GPS 点的距离。众所周知，GPS 点的采样时间间隔通常是恒定时，所以距离是由车辆的移动速度决定的。换句话说，元素 $r\ d_i$ '的值取决于车辆的行驶速度。对于一个距离序列，它的速度很有可能在一个小范围内波动，这也是为什么建立在单个距离序列之上的霍夫曼树很简单的原因。这里，我们利用生成距离序列的道路类型信息对不同的距离序列进行聚类，其原理是相同的道路类型具有相似的交通规则和相同的限速，

所以在通过相同类型的道路时生成的距离序列期望呈现一个接近的值分布。对于每个距离序列簇，我们将它们组合成一个较长的距离序列，并在此基础上建立一个新的霍夫曼树，其过程与算法 4.1 相同。正如预测的那样，构建的树包含少量的叶子节点。我们的霍夫曼森林总共需要 8 棵树，这是因为从 OpenStreetMap 下载的路网一般包含 8 种常见的道路类型标签，即“高速公路”、“干线”、“主要”、“次要”、“高等”、“住宅”、“服务”及“无法分类”。

在已建霍夫曼森林的基础上，给出一个新的待压缩的距离序列，首先根据它的道路类型来确定霍夫曼树，然后对序列中的元素进行编码。

4.4.2 AV 压缩器

步骤 1:加速度序列的离散化

加速度序列 $A\ Tr$ 由 $\overline{a_1},\overline{a_2},\ \ ,\overline{a_{l\ 1}}$ 表示。与 $r\ d_i$ 类似，$A\ Tr$ 序列中的每个元素 $\overline{a_i}$ 也是一个无限字符串，因此我们基于式（4.7）对其进行离散化，得到离散值 $r(d_i)'$，其中　表示离散程度且 $i\quad 2\ \overline{a_i}/\quad$ 。离散化后，$A\ Tr$ 已经被 $A\ Tr\quad \overline{a_1}',\overline{a_2}',\ \ ,\overline{a_{l\ 1}}'$ 取代了。同时，离散化将不可避免的失去准确性。因此，根据不同应用对于加速度信息损失的要求，用户可以指定不同的　值。

$$\overline{a_i}\quad \begin{matrix} \dfrac{i}{2}, if\ |\overline{a_i}\quad \dfrac{i}{2}|\ |\overline{a_i}\quad \dfrac{(i\quad 1)}{2}| \\ \dfrac{(i\quad 1)}{2}, if\ |\overline{a_i}\quad \dfrac{i}{2}|\ |\overline{a_i}\quad \dfrac{(i\quad 1)}{2}| \end{matrix} \tag{4.7}$$

步骤 2:通过霍夫曼编码加速度序列

我们还对大样本加速度序列进行了观测研究，得到了与距离序列相似的现象。因此，我们也采用霍夫曼编码对一个加速序列 $A\ Tr$ 进行压缩，压缩结果记为 $A\ Tr\quad \overline{a_1}'',\overline{a_2}'',\ \ ,\overline{a_{l\ 1}}''$ 。霍夫曼编码的详细过程已经在 4.4.1 节中解释过了。

步骤 3:通过霍夫曼森林编码所有加速度序列

与对所有距离序列进行编码的动机相同，我们提出了建立霍夫曼森林来对所有加速度序列进行编码，目的是为了识别具有相似分布的加速度序列簇。

众所周知，加速度序列内的元素值与速度方差密切相关，因此元素值主要受驾驶风格、交通状况、周围空间环境等因素的影响[5,78]。驾驶风格是高度主观且因人而异的，但临近道路的交通条件和空间环境可能高度相似。因此，聚类加速序列有两种可能的解决方案，即，识别具有相似驾驶风格的用户产生的加速序列和识别在近距离空间域内产生的加速序列。这里，考虑到我们收集的轨迹数据并没有明确表示出驾驶风格的信息，因此我们主要是基于穿越道路的邻近程度来聚合加速度序列。具体来说，我们首先将整个城市分割成 $N\quad N$ 个大小相等的网格

单元，并简单地将同一网格单元中道路行驶时产生的加速度序列进行聚类。因此，我们总共有 N^2 个簇，需要构建 N^2 个霍夫曼树（霍夫曼森林）。实际上，霍夫曼树的数量远远小于 N^2。这是因为，一方面，有大量的不可到达的网格单元(如河流、山脉、建筑物)，这些网格无法产生霍夫曼树；另一方面，在某些网格单元中生成的加速序列可能会产生相同的霍夫曼树，重复的树可以删除。

速度(V)压缩器:在速度序列中只有一个元素 v_1(即轨迹片段的初始瞬时速度)，我们仅对 v_1 进行简单压缩，即四舍五入。

4.4.3 T 压缩器

步骤 1:转换时间序列

时间序列 $T\ \ Tr$ 用 $t_1, t_2, \ \ , t_l$ 表示。序列中的每个元素 t_i 都是一个巨大的 UNIX 数字(例如，1545188400)，而两个连续时间戳(t_i 和 $t_{i\ 1}$)之间的时间间隔 $t_{i\ 1}$ 通常较小(如 6 秒)。因此，为了节省内存空间，我们将序列中的每个时间戳 $t_i\ (i\ \ l\ \ 1)$ 替换为时间间隔 $t_{i\ 1}$。最终，原始时间序列 $T\ \ Tr$ 可以转化为 $T(Tr)\ \ t_1, t_2, \ \ , t_l\ \ t_1, T(Tr)$。新的时间序列方向 $T\ \ Tr$ 由两部分构成，即时间间隔序列 $T(Tr)$ 和初始时间戳 t_1。接下来，我们将展示编码时间间隔序列 $T(Tr)$ 的详细过程。

步骤 2:对时间间隔序列进行编码

对于某些时间间隔序列，序列中的所有元素(t_i)都是相同的，等于 GPS 点的采样时间间隔。然而，对于一些时间间隔序列，由于 GPS 信号的延迟或不可用，每个序列中的元素在采样时间间隔值附近轻微地上下波动。我们根据序列所包含的元素是否完全相同，将时间间隔序列分为为高质量序列和低质量序列。然后我们分别对高质量序列选择 Run-Length Encoding(RLE)算法，对低质量序列选择霍夫曼编码，具体细节如下。

利用 RLE 算法压缩高质量的时间间隔序列 RLE 算法通常用于压缩高度重复的数据。它的主要原理是若序列中相同的值出现在许多连续元素中，则存储为该值(记为 t_i)及其连续出现的次数（记为 $Num(\ t_i)$）。例如，给定一个序列 $T(Tr)\ \ 6,6,6,6,6,6,6,6,6,6,6,6$，那么它的压缩结果为 $(6,10)$，其中 6 和 10 分别为 t_i 和 $Num(\ t_i)$，我们很容易知道序列的压缩结果仅占 8 个比特的存储空间。

利用霍夫曼编码压缩低质量的时间间隔序列 使用霍夫曼编码对低质量的时间间隔序列进行编码更有效。我们再次使用图 4.4 所示的例子来证明这种优越性。给定一个时间间隔序列 $T(Tr)\ \ 6,6,6,6,6,5,7,7,8$，当使用 RLE 算法时，压缩结果是 $6,6$，$5,1$，$7,2$，$8,1$，压缩结果占用高达 32 比特的存储空间。相反，如果使用霍夫曼编码，压缩结果的数据大小只有 16 比特。为了建立一个通用的霍夫

曼树，这里，我们采取一个简单的做法，即将足够数量的低质量时间间隔序列放在一起形成一个长序列，并在此基础上建立一个霍夫曼树。与前面压缩距离序列和加速度序列的情况不同，这种简单的做法是可行的，这是因为低质量序列的相连时间戳间隔值仅在采样时间间隔值上下轻微波动，所以新形成的长序列将具有非常少的唯一值。

为了区分编码时间间隔序列的压缩方法，我们有意在压缩结果之前添加一个二进制方法标签，称为 ml。具体来说，标签 ml 1表示使用 RLE 算法进行压缩的情况，而 ml 0表示使用霍夫曼编码进行压缩的情况。最后序列 $T(Tr)$ 表示为 $(ml,\ T(Tr)')$，$T(Tr)'$是序列 $T(Tr)$ 的压缩结果。

步骤 3:对所有时间序列进行编码

我们用同样的过程来编码所有的时间序列。具体地说，当一个新的时间序列 $T(Tr)$ 到达时，我们首先将它转换为一个新的时间序列(参见步骤 1)，它由两个组件组成，即时间间隔序列 $T(Tr)$ 和初始时间戳 t_1。然后根据步骤 2 中讨论的方法对该时间间隔序列进行编码。最终压缩后的时间序列 $T\ Tr$ 表示为 $T\ Tr\quad t_1,(ml,\ T\ Tr\ ')$。

4.4.4 DAVT 解压器

与数据压缩算法相似，对三种序列分别进行数据解压，但该过程要简单得多。这里，我们简要概述如何解压它们。

对于给定的压缩距离序列 $D(\dddot{Tr})$，我们首先利用相应的空间轨迹获取相关的道路类型信息，然后根据道路类型信息确定压缩序列时使用的某棵霍夫曼树，最后根据这棵树的信息，解压距离序列变得非常简单。以压缩距离序列 $D(Tr)\quad 1111110000101001$ 为例，利用图 4.4 右图的霍夫曼树，我们可以很容易地将二进制代码解码成 $D(\ddot{Tr})\quad 6,6,6,6,6,5,7,7,8$。加速度和速度序列的解压过程与距离序列的解压过程非常相似，唯一的区别是确定霍夫曼树的方法不同。解压加速度序列时，我们根据生成轨迹的城市网格位置选择解压需要的霍夫曼树。要解压时间序列，通常需要两个步骤：第一步，根据二进制方法标签确定解压方法；第二步，用不同方法对时间序列进行解压。如果方法标签是 ml 1，则通过解码 RLE 算法进行数据解压；否则，数据的解压过程与距离序列的解压过程相同。值得注意的是，在这种情况下只有一个通用的霍夫曼树。例如，如果压缩的时间序列是 $(1,6,5)$，那么解压结果应该是 $6,6,6,6,6$。

4.5 误差边界的理论分析

我们从理论上证明了上述压缩算法的误差是有界的。

定理 1：损失的距离的上界是$\frac{}{4}$ T_2 $\sum_{i=2}^{l}\frac{}{4}(t_i - t_{i-1})^2$，$$ 是离散程度，T 是最常出现的采样时间间隔，t_i 为第 i 个 GPS 点的绝对采样时间戳。

证明 距离序列中的每个元素 $r(d_i)$，我们很容易获得它的离散值与本身的差别上限是$\frac{}{4}$，在数学上式（4.8）是成立。

$$|r(d_i)' - r(d_i)| \quad \frac{}{4} \tag{4.8}$$

因此，根据距离表示中的相对距离定义，我们得到式（4.9），

$$|d_1' - d_1| \quad \frac{}{4} \quad T^2, |d_i' - d_i| \quad \frac{}{4}(t_i - t_{i-1})^2 \quad i \quad [1,l] \tag{4.9}$$

通过简单的替换，最大累积距离误差（即 $\sum_{i=2}^{l}(d_i' - d_i)$）的上限是 $\frac{}{4}$ T^2 $\sum_{i=2}^{l}\frac{}{4}(t_i - t_{i-1})^2$。

最坏的情况发生在距离序列中的每个元素由于数据离散化而产生最大正误差。

定理 2：加速度损失的上界是$\frac{}{4}$，$$ 是加速度序列的离散度。

证明：根据加速度表示中的加速度定义，可以很容易地证明。

4.6 实验评估

4.6.1 实验设置

① **基准算法**：我们在研究中使用了三个基准算法进行实验比较，具体如下。

PRESS 算法 在 PRESS 算法中时间轨迹片段由二元组序列 t_i, d_i 表示，其中 t_i 和 d_i 分别表示为采样时间戳和自发动机启动以来在空间路径上的累计行程距离。因此，该序列可以看作是二维平面上的时间-距离曲线。在此基础上，与 Douglas-Peucker(DP)算法的原理相似，数据压缩只需要保持平面上的特征点即可实现。读者可以文献[19]了解更多细节。

CCF 算法 它与 PRESS 算法有相同的时间轨迹表示方法。唯一的不同之处在于，CCF 算法的原理是在用户定义的误差范围内进行线性拟合，拟合一条最优直线来近似原点曲线。换句话说，拟合线生成了一组全新的点(除了第一个点)，但导致了额外的存储成本。读者可以文献[17]了解更多细节。

TED 算法 时间轨迹由两种序列表示，即 $r \quad p_i$ 和 t_i。$r \quad p_i$ 和 t_i 分别表示 p_i 到所处边的头节点的相对距离和采样时间。对于距离序列，TED 算法采用编码的原理来压缩数据。$r \quad p_i$ 的值越大，用于编码的二进制字符就越多。对于时间序列，

则通过丢弃具有相同时间间隔的时间序列来实现数据约简。读者可以文献[60]了解更多细节。

注意：三种基准算法均未考虑对轨迹数据中记录的时变速度信息进行压缩。为了使比较公平，在评估中，我们只展示了 DT 压缩器的性能结果。

② **数据准备** 实验中我们使用了中国北京市的一个路网数据集和一个出租车 GPS 轨迹数据集。其中路网可以从 OpenStreetMap 免费下载，总共包含 141735 个节点和 157479 条边。GPS 轨迹数据是 595 辆出租车在一周内(2015 年 9 月 15 日至 21 日)生成的，包含了超过 1700 万个 GPS 点。另外，GPS 点采样时间间隔约为 6 秒，由于 GPS 信号的时延，约有 20%连续时间戳的时间间隔上下波动(即 5~12 秒)。原始轨迹数据的存储空间超过 1.01GB。

③ **评价指标** 我们使用以下指标来评估性能。

压缩率 压缩率(cr)定义为原始数据占用的存储空间与压缩数据占用的存储空间之比，很容易知道cr总是大于 1。其他数据压缩器也用该指标来衡量压缩器的压缩效果。

压缩和解压时间 我们分别用数据压缩和解压的时间成本来衡量算法的效率。

时间同步的网络距离 对于距离序列，时间同步的网络距离($TSND$)用于衡量数据压缩引起的距离损失。具体来说，我们只是使用最大的$TSND$来量化距离误差。采样时刻t_i的距离误差(dE_i)为压缩前后基于距离序列计算出的距离之差。L_i记为基于原始距离序列(离散化和编码前)计算出的t_i时刻沿空间路径从起点开始的累计距离值；L_i'是指根据压缩距离序列(离散化和编码后)计算出的t_i时刻的累计距离值。$TSND$在数学上可以被表示为$Max_{i\ 1}^{l}\ dE_i$，其中l为轨迹片段的长度。很容易理解，$TSND$与 和l相关。

时间同步的加速度误差 对于加速度序列，时间同步的加速度误差($TSAD$)用于衡量数据压缩引起的加速度损失。具体来说，我们使用最大的$TSAD$来量化加速度误差。采样时刻t_i的加速度误差($accE_i$)为压缩前后基于加速度序列计算的加速度之差，分别用$accE_i$和$accE_i'$表示。其中，$accE_i$为基于原始加速度序列(离散化和编码前)计算出的在t_i时刻的加速度值；$accE_i'$是指基于压缩后的加速度序列(离散化和编码后)计算出的在t_i时刻的加速度值。$TSAD$在数学上可以被表示为$Max_{i\ 1}^{l}(accE_i)$，其中l为轨迹片段的长度。同样很容易理解，$TSAD$与 和l紧密相关。

④ **运行环境** 实验评估都是在 MATLAB（2018 版本）中运行的，电脑的型号是 Intel (R) Xeon (R) CPU E3-1275 PC 上的 32 GB RAM，操作系统是 Windows 10。

4.6.2 训练 DAVT 压缩器

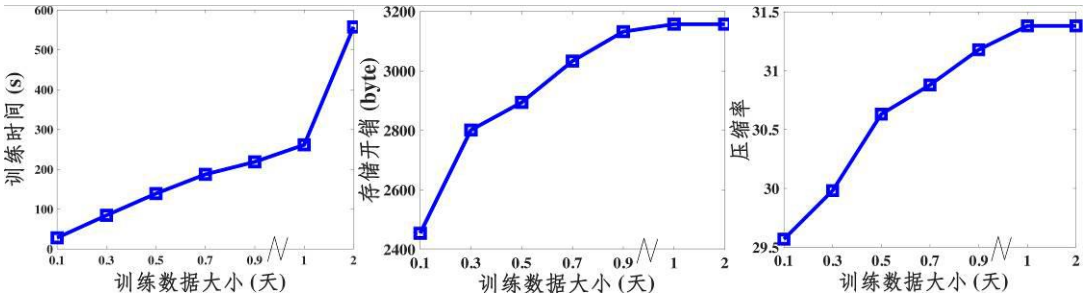
在我们提出的算法能够工作之前，一个重要的步骤是离线训练，即构建霍夫曼树和霍夫曼森林。因此，我们评估了所使用数据的大小与训练时间长短、保存霍夫曼森林需要的存储空间大小和算法的压缩率高低的关系。使用更少的数据可以有利于压缩算法框架的维护，并使其更具竞争力。这里为了简单起见，数据的大小是由生成轨迹数据的持续时间来度量的，持续时间(以天为单位)与生成的 GPS 点数量之间的关系如表 4.2 所示。

表 4.2 训练数据的持续时间与 GPS 点数量之间的关系 (以天为单位)

Table 4.2 # of GPS points vs time duration (in day)

持续时间	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**	**1**	**2**
GPS 点	261k	784k	1302k	1822k	2351k	2439k	5177k

在不同大小的训练数据下，算法的性能训练结果如图 4.5 所示。如图 4.5(a)所示，正如预测的那样，随着训练数据的增大，训练时间几乎呈线性增长。如图 4.5(b)和(c)所示，在达到稳态之前，离线训练模型所占用的空间和压缩率都随着训练数据的大小的增加而逐渐增大。这是因为当训练数据一开始变大的时候，会识别出更多的唯一的霍夫曼树。但是，当训练数据的大小达到一定的阈值时(即一天的训练量)，训练时间、存储空间和压缩率保持不变。结合图 4.5 所示的训练结果，我们决定使用一天的轨迹数据来训练我们的模型，这是因为此时的训练量可以平衡训练时间成本和压缩性能。在这个实验中，我们设定 $\quad 0.5, \quad 0.5, l \quad 10, N \quad 9$。



（a）训练时间　　（b）存储开销　　（c）压缩率

（a）Training time　　（b）Storage cost　　（c）Compression ratio

图 4.5 不同数据大小下的训练性能

Figure 4.5 Training performance under different data sizes

在为 AV 压缩器构造霍夫曼森林时，模型训练的另一个关键参数是城市网格的数量（即 N ）。因此我们测试了不同 Ns 下的压缩器性能的训练结果，结果如表 4.3 所示。从图中可以看出，随着城市网格单元的增多，需要更长的训练时间。并且随着网格数量的增加，训练模型的存储成本指数级增长，而压缩率的提高却十分有限。综上所述，我们在剩下的实验中固定了 N　9，这是因为它能平衡训练成本和压缩性能。在这个实验中，我们设定　0.5,　0.5, l　10 。

表 4.3　不同 Ns 下的压缩性能

Table 4.3 Training performance under different Ns

N	**1**	**4**	**9**	**16**	**25**
训练时间（秒）	231	240	262	269	285
存储空间（比特）	220	900	3157	7700	16600
压缩率	31.0	31.4	31.7	32.1	32.8

4.6.3 信息损失研究

除了对误差边界的理论分析外，我们还对实际情况下的信息损失感兴趣。因此，我们研究了不同参数设置下测试数据的距离和加速度信息损失。

距离信息损失　如前所述，我们使用 $TSND$ 来衡量距离序列的距离损失。对于一组距离序列，我们简单地使用所有序列中最大 $TSND$ 来量化所有距离序列中的最大距离信息损失。我们认为在实践中最坏情况下的距离损失更有意义。如果最大距离损失仍然不差，则有损压缩框架可以接受。

图 4.6 显示了在不同　s 和 ls 下，最大距离损失结果。正如预期的那样，随着变大，数据离散化造成的距离信息损失变得严重。即使我们设置　为 1.5，最大距离误差也不超过 60 米，这表明 D 压缩器是相当准确的。从图 4.6 的右图可以看出，最大 $TSND$ 随轨迹片段的长度增长而线性增加。这是因为随着轨迹片段的长度的增长，距离误差积累效应会变得更加严重(详见 $TSND$ 的定义)。

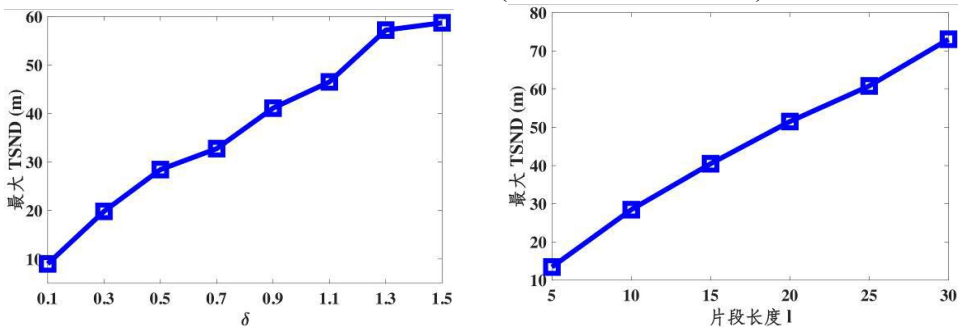


图 4.6　在不同　s (左)和 ls (右)下最大的距离损失

Figure 4.6 Distance loss under different　s (left) and　ls (right)

加速度信息损失 在相同的动机下，我们还利用最大的*TSAD*来量化所有加速度序列中的加速度损失。我们调查了在不同 *s* 和*ls* 下，最大的*TSAD*，图 4.7 显示了测量结果。可以看出，如果 变大，与距离损失的情况类似，这是因为更多的信息将被丢失。如图 4.7 右图所示，在不同*ls* 下，*TSAD* 的最大值保持不变，这是因为*TSAD* 的定义为两个连续采样时间之间的速度方差，因此*TSAD* 不随*l* 值累加。

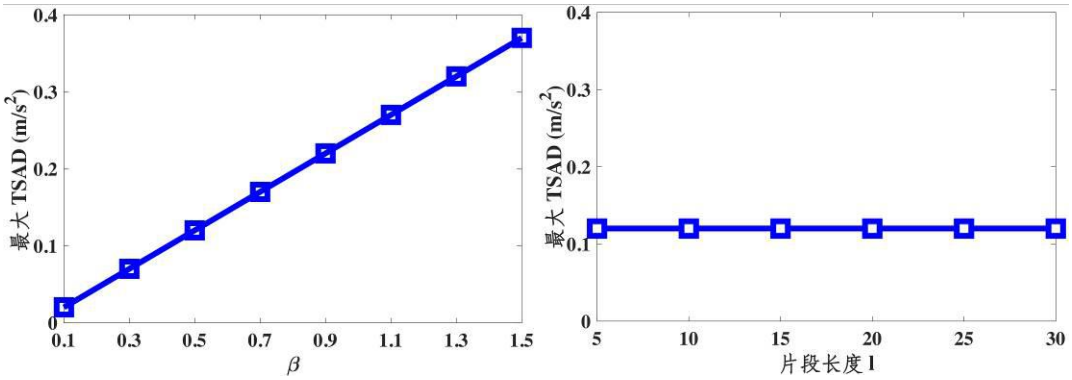


图 4.7 不同 *s* (左)和*ls* (右)下最大加速度损失

Figure 4.7 Acceleration loss under different *s* (left) and *ls* (right)

4.6.4 评估 DT 压缩器的性能

三个基准算法都是有损压缩的，它们通常只能在预先设定错误范围后才能工作。根据距离损失的研究结果(如图 4.6 所示)，我们可以选择合适的参数(即)来预计距离的误差范围。例如，如果我们期望距离误差小于 30 米，那么可以设置 0.5。在此情况下，我们从压缩率、压缩时间和解压时间成本等方面比较 DT 压缩器与其他三条基准算法的压缩性能，其结果如下。

压缩率 图 4.8 分别给出了在不同距离误差和*ls* 下的压缩率，从图 4.8 的左图可以看出，四种压缩算法的压缩率都随着距离误差的增大而稳步提高。这很容易理解，如果我们允许更大的距离信息损失，压缩性能会更好。此外，所提出的 DT 压缩器的压缩率始终高于其他三种基准算法，这证明了新的时间轨迹表示和我们设计的压缩算法的有效性。

图 4.8 右图为不同轨迹片段长度下的压缩率结果。对于所有四种算法，当*l* 变大时，压缩率随之变大。同样，提出的 DT 压缩器性能始终优于传统的压缩器。对于 DT 压缩器和 TED 算法，它们都包含两个部分(即距离和时间)。对于距离数据部分，压缩数据的大小由用于编码每个元素的二进制代码和总元素的数量决定。因此，其大小应保持不变，并独立于段的长度。而时间数据部分，随着*l* 的增加，时间相关片段的数量减少，导致占用空间减少。需要注意的是，对于每个时间段，

经过数据压缩后，其大小通常是恒定的。因此，如果轨迹片段变长，压缩后的数据占用的空间就会减少。

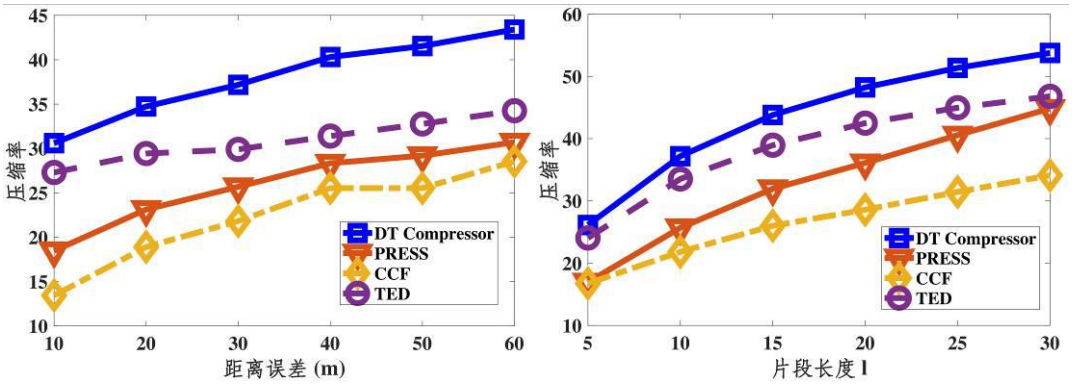


图 4.8　不同距离误差(左)和 ls (右)下的压缩率

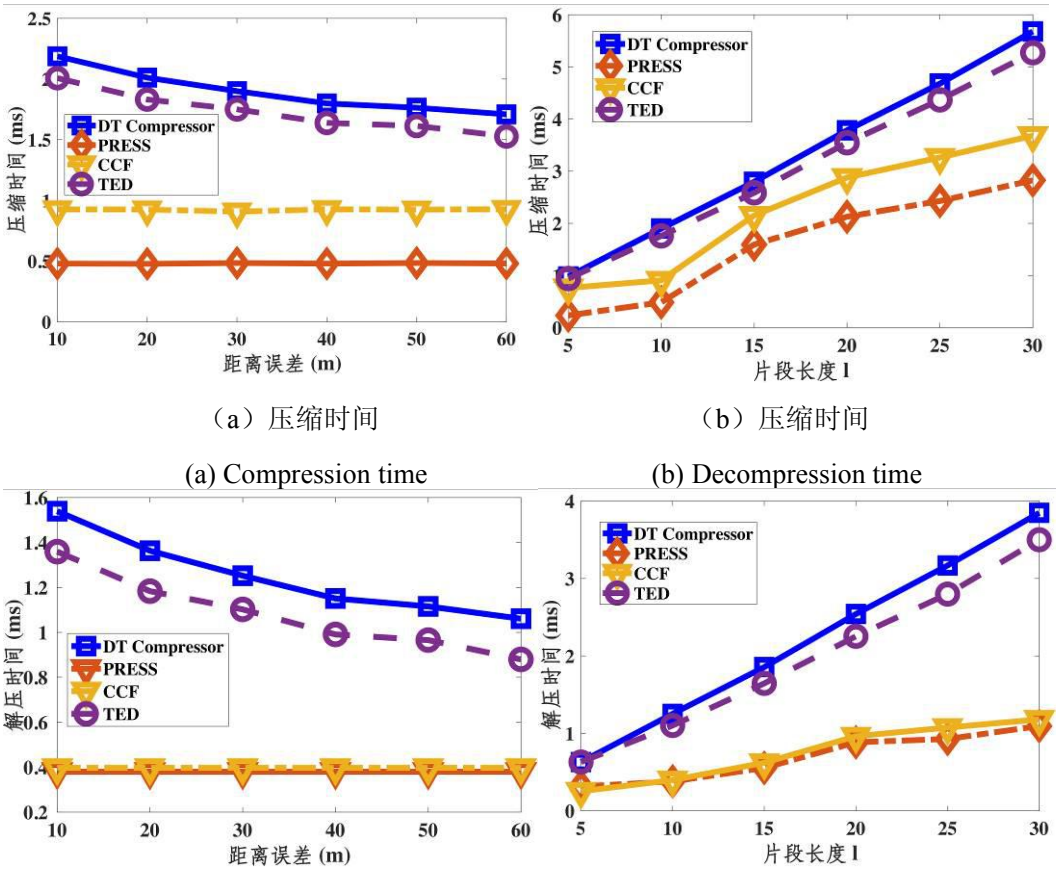
Figure 4.8 Comparison results on the compression ratio under different distance errors (left) and ls (right)

压缩和解压时间成本　图 4.9 为不同距离误差和 ls 下压缩时间和解压时间成本的结果。从图 4.9(a)和(b)可以看出，对于 DT 压缩器和 TED 算法，如果允许较大的距离误差，则压缩数据所需的时间较少。同时，在此情况下，数据解压所需的时间更少。在 DT 压缩器中，一个更大的距离误差表示使用一个相对较大值　进行数据离散化，在这种情况下，更简单的霍夫曼树结构和更少的树足以编码所有的数据元素。当树变简单时，压缩和解压时间因此降低。对于 TED 算法，距离误差越大，数据离散化的分辨率越粗糙，导致离散后序列包含的唯一值减少，因此数据压缩和解压都变得更加容易和高效。对于 PRESS 和 CCF 算法，根据它们的工作原理，可以得出时间成本与距离误差无关。例如，给定任意距离误差，PRESS 算法始终需要确定是保留还是丢弃当前数据点。因此，它们在面对所有距离错误时，花费稳定的时间在数据压缩和解压中。

图 4.9(c)和(d)展示了轨迹片段长度与时间成本的关系。对于四种算法，随着 l 的增大，数据压缩和解压的时间成本会提高。这个观察很容易理解，因为如果设置一个更大的 l，DT 压缩器需要处理更多的数据元素，从而需要消耗更多的计算时间成本。我们还可以得出结论，压缩率的提高往往是以增加压缩和解压时间(即牺牲效率)为成本的。

对于四种算法，数据解压都比数据压缩更有效，并且以毫秒级的粒度完成。这是因为解压不需要预处理原始轨迹片段，特别是地图匹配的计算成本非常高。在时间开销方面，DT 压缩器在数据压缩和解压的性能略差于 TED 算法。这是因为我们采用了更加精密和精确的时间压缩器来处理时间部分。同时，时间部分的

解压也花费了额外的计算成本。相比之下，PRESS 和 CCF 算法在数据压缩和解压时的效率都要高得多。



（a）压缩时间　　（b）压缩时间

(a) Compression time　　(b) Decompression time

（c）解压时间　　（d）解压时间

(c) Compression time　　(d) Decompression time

图 4.9 不同距离误差和 ls 下压缩和解压时间成本

Figure 4.9 Comparison results on the compression and decompression time cost under different distance errors and ls

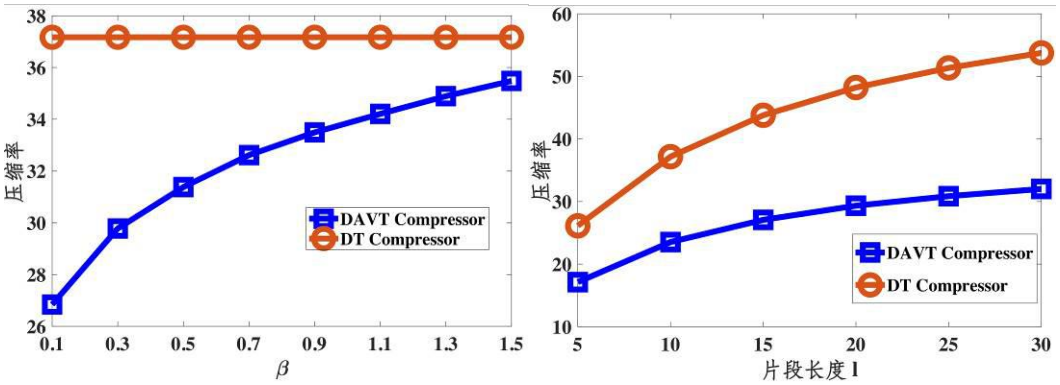
4.6.5 DT 压缩器 VS DAVT 压缩器

由于 DAVT 压缩器对时变速度信息的处理非常谨慎，而不是直接丢弃，因此与 DT 压缩器相比，可以预见 DAVT 在压缩效果和效率上都会下降。为了了解性能退化到何种程度，我们定量测量了 DT 压缩器与 DAVT 压缩器压缩率、压缩和解压时间成本之间的差异，结果如下:

压缩率 图 4.10 为 DT 与 DAVT 压缩器的压缩率对比结果。从图(a)中可以看出，在不同的 s 下，DT 压缩器压缩率保持不变，这是因为它与 无关。对于 DAVT 压缩器，一个更大的 值代表在数据离散化中的更粗分辨率和更多的信息损失，从而压缩率得到提高。这与 DT 压缩器的压缩率随 s 的增加而降低的原因是相同

的。更重要的是，当我们设置了一个更大的　值时，观察到的两种压缩器的压缩率的差距逐渐缩小。

如图 4.10(b)所示，当l值变大时，DT 和 DAVT 压缩器的压缩率都增大。对于 DAVT 压缩器，它又包含两个部分(即加速度和速度)。与距离部分相似，数据压缩后加速度占用的空间与段的长度无关。如果取更大的l值，会产生更少的速度和时间数据，最终导致压缩率的增加。



（a）压缩率　　（b）压缩率

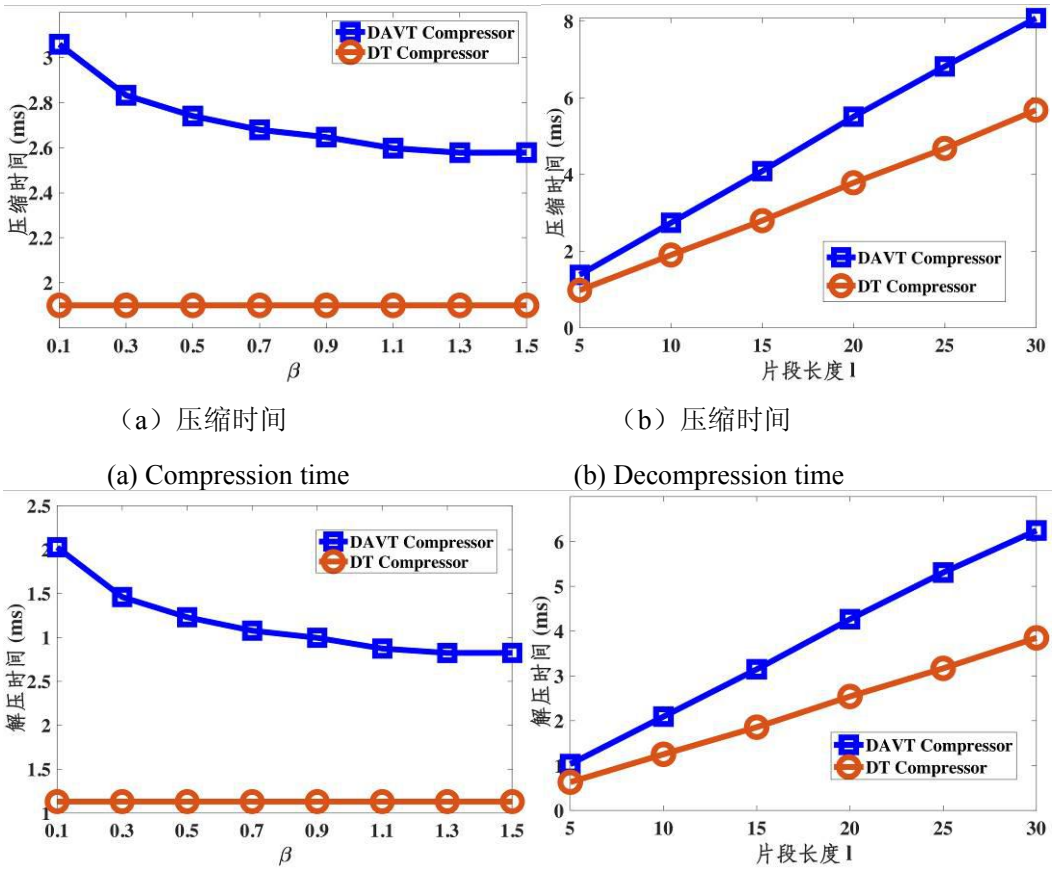
(a) Compression ratio　　(b) Compression ratio

图 4.10　不同的　s 和 ls 下 DT 和 DAVT 压缩器的压缩率

Figure 4.10 Comparison results between DT and DAVT Compressor on the compression ratio under different　s　and　ls

压缩和解压时间成本　图 4.11（a）（b）显示了在不同　s 和 ls 下压缩和解压时间成本上的比较结果。很容易理解在不同　s 上，DT 压缩器的压缩和解压时间应当是相同的，这是因为 DT 压缩器独立于　。当　变大时，DAVT 压缩器的压缩和解压时间随之减少，这与 DT 压缩器的效率随　的增加而提高的原因是相同的。这两种压缩器都非常高效，可以在几毫秒内做出响应。

图 4.11(c)和(d)为两种压缩器的对比结果。从图中可以看出，对于 DT 和 DAVT 两种压缩器，其压缩和解压时间均随片段增长而线性增加。原因是两台压缩器压缩每个数据元素的时间相同，而 DAVT 压缩加速度信息需要的时间相对较多。我们还可以观察到，数据解压通常比数据压缩花费更少的时间。



（a）压缩时间　　（b）压缩时间

(a) Compression time　　(b) Decompression time

（c）解压时间　　（d）解压时间

(c) Compression time　　(d) Decompression time

图 4.11 不同的 s 和 ls 下 DT 和 DAVT 压缩器的压缩和解压时间成本

Figure 4.11 Comparison results between DT and DVAT Compressors in terms of compression and decompression time cost under different s and ls

4.7 章节小结

我们提出了一种在时间维度上全新的轨迹数据表示和对应的压缩框架—DAVT。具体来说，轨迹数据被重构为三个部分,即距离（D）、加速度和速度（AV）、时间(T)和三个压缩器分别约简每个部分。对于 D 和 AV 部分，我们利用相似的霍夫曼森林结构对数据元素进行有效编码，但编码原理不同。对于 T 部分，首先我们将大的绝对时间戳转换为小的时间间隔，并根据数据质量提出了两种不同的编码技术来实现数据压缩。据我们所知，我们是压缩时变速度信息的先驱之一，该信息可以支持更多类型的查询。此外，我们还从理论上证明了压缩算法的理论误差上限。最后，我们使用从中国北京收集到的大型出租车轨迹数据集对系统进行了全面的评估。实验结果表明，新设计的压缩器性能在压缩率上优于其它基准方法。

5　移动环境下在线轨迹压缩的实现

本章介绍了在真实移动环境下部署轨迹压缩系统。本章内容安排如下：5.1 节陈述了设计的轨迹压缩系统的优点；5.2 节阐述了系统的主要设计原理；5.3 节介绍了优化系统效率的两种策略；5.4 节中全面评估了压缩系统在真实环境下的性能；最后 5.5 节对本章进行了总结。

5.1 前言

随着各种 GPS 设备的广泛应用，多种轨迹数据可以轻易大规模获取，车辆轨迹数据就是典型代表[66]。这样的轨迹数据不仅可以用于直接跟踪车辆的行驶路径，还可以实现大量智能的城市服务，如了解城市和交通动态[74]，推荐驾驶路线[72]，推断城市建筑功能[11]等。因此，近年来越来越多的研究者关注用于城市计算挖掘任务的轨迹数据。为了节省通信和存储成本，车辆仅以稀疏的时间间隔（即，较不频繁地）将其 GPS 位置报告给数据中心。例如，根据出租车是否被乘客占用，嵌入在车辆中的 GPS 设备以 1 或 2 分钟的时间间隔将其位置发送到数据中心[13]。但是，这种简单而原生的解决方案也引发了一些严重的问题。例如，在两个连续的 GPS 点之间产生很大的不确定性[48]。更糟糕的是，GPS 噪音和道路网络的复杂性使得驾驶路径之间的推断更具挑战性[66]。因此，可启用的应用程序非常有限。

为了在成本节约和轨迹数据的可用性之间进行权衡，一种有效的方法是在线轨迹数据压缩。即通过在 GPS 设备端更频繁地收集 GPS 位置，却将较少但完整的数据上传到数据中心。因此，计算出简洁且空间无损的压缩轨迹是必不可少的。然而，这样的计算通常需要大量的资源，而 GPS 设备本身无法承担繁重的任务[87]。因此，为了使该想法可行，系统还应满足一些其他要求。

计算能力 一般来说，在线轨迹匹配是高性能轨迹压缩的先决条件，匹配任务是计算密集型的。因此，系统需要强大的计算能力。因为 GPS 设备计算能力有限，因此现有的 GPS 设备并适合承担计算任务。例如，对于当前安装在出租车上的 GPS 设备[57]，嵌入的 RAM 只有 4KB 大小。

低延时 GPS 设备端密集地收集 GPS 轨迹数据，为了保证系统能够及时对轨迹数据进行在线压缩，系统需要较低的延迟。即在新的数据点到来之前，对缓冲区中的轨迹数据要完成压缩。

轻量级 该系统应该是轻量级的（即占据计算设备少量的存储空间和计算内存）。这是因为车辆经常处于高速移动状态，嵌入式计算的空间也在移动环境中受

到限制。而且，在移动环境下，系统的能量供应有限，但轻量级的系统通常消耗较少的能量。

设计原理 为了满足上述系统要求，受到移动边缘计算的启发[84,88]，本文提出利用司机的移动电话作为本地计算单元来承担计算任务的想法。更具体地说，①移动电话非常适合繁重的计算，因为它们具有强大的计算能力并且在驾驶时几乎是空闲的。②GPS 设备收集的密集轨迹数据可以轻松且无延迟地传输到移动电话上，其中的传输延迟可以忽略不计。例如，通过蓝牙将数据点从 GPS 设备发送到移动电话只需不到 0.2 毫秒。与 GPS 设备的采样时间间隔相比，当采样密集时，该采样时间间隔通常在 1～10 秒的范围内，这种传输时间可以被忽略。更重要的是，本文进一步提出了一种具有成本效益的在线轨迹压缩算法，以及时响应。③本文将在 Android 智能手机平台上部署线轨迹压缩系统。该应用程序非常轻巧，只占用移动电话的非常小一部分空间和内存。该应用程序还具有高能效，例如 12 小时运行时消耗的电池电量不到 10％。此外，本文还开发了一个前端可视化器，可以很好地告知驾驶员他们的轨迹和附近的空间背景（例如兴趣点）。

为什么选择手机作为系统中的边缘设备？边缘设备被定义为驻留在数据源（例如，原始 GPS 轨迹）和云的数据中心之间的计算或网络资源[84]。常见的边缘设备包括智能手机、基站、路边单元和 Wi-Fi 路由器。例如，[84]的研究人员提到，在可穿戴传感器和云端数据中心之间，智能手机可以扮演边缘设备的角色或微数据中心。此外，[89]的研究人员还将资源匮乏和耗能的计算任务从可穿戴设备迁移到移动电话，因为可穿戴设备通常受到计算能力和能源供应的限制。从上面的例子中，可以得出结论，弱计算能力的设备通常会卸载一些繁重的计算任务，并把这些任务迁移到智能手机上。对于其他边缘设备（例如，路边单元和基站），与移动电话相比，它们具有更强大的计算能力。因此，如果采用这些设备，可以减少系统的响应时间。但是，它们作为所本章系统中的边缘设备是不适合的。例如路边单元，一个关键的缺点是由于通信范围有限和基础设施的稀疏部署而导致设备之间会出现连接中断[90]，现有的基础设施不能强有力地支持在线轨迹压缩服务，特别是海量车辆同时上传其轨迹数据时。如基站，车载 GPS 设备与基站之间通过蜂窝网络（例如，3G，4G 和 5G）来通信[88,91-93]，但高成本通信将不可避免地而产生，这违背了本文的目标，即节省通信成本。此外，由车辆的高机动性可能会引起的大量基站切换，因此轨迹压缩进程会被阻碍[94,95]。相反，智能手机的使用不仅可以避免上述问题，而且可以规避其他缺点，尤其是可扩展性问题，因为设计的系统是分布式的。

本章的主要贡献 本章描述了一款名为 VTracer 的在线轨迹压缩系统的设计原理、工程实现以及实验评估。VTracer 的特点是，通过充分利用手机的优势，

将移动边缘计算理念带入轨迹数据挖掘任务的一个初步但非常重要的步骤-数据收集。通过使用 VTracer，数据中心可以为移动车辆获得无噪声、更简洁更完整的轨迹表示。此外，系统还可以实现存储、通信等方面的有效的成本节省。最后，本文使用中国重庆市产生的 13 个小时的实际驾驶数据（相当于行驶距离约 530 公里），测试了 VTracer 的系统性能，还将其与最先进的压缩技术进行比较，包括轨迹匹配质量、压缩率、内存、能耗以及实际情况下的应用程序大小。实验结果证明了 VTracer 的卓越性能。

5.2 系统概述

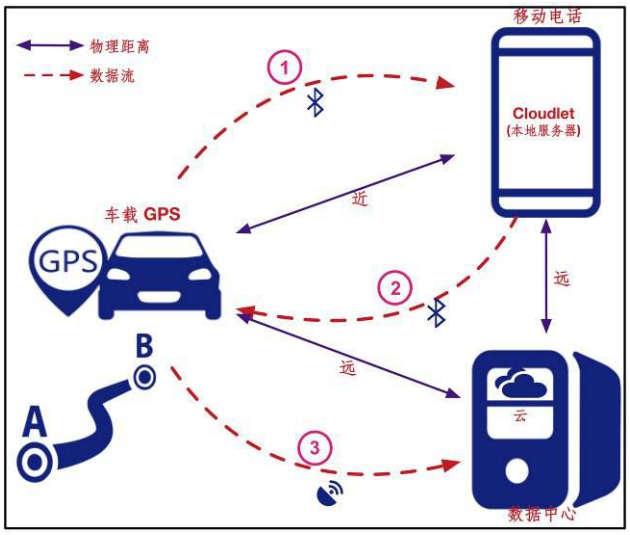


图 5.1 VTracer 系统概述

Figure 5.1 The system overview of VTracer

图 5.1 为 VTracer 的系统概述，它由三部分组成，即车载 GPS 设备、移动电话和云端数据中心。GPS 设备与手机之间的物理距离短且稳定，而 GPS 设备与数据中心之间的物理距离遥远且多变。GPS 设备采集车辆的实时轨迹数据，并通过蓝牙将数据连续发送到驾驶员的手机上(数据流如图①所示)。我们开发了一款名为 TrajCompressor 的应用，将原始轨迹数据流输入到 APP 中，APP 将数据流切割成等长的轨迹片段，并对其进行在线轨迹匹配与压缩（具体算法参照第 2～3 章），最后 APP 又借助蓝牙将压缩后的片段返回 GPS 设备(数据流如图②所示)。这里 GPS 设备和手机通过蓝牙进行连接，蓝牙通信是高效且稳定的，GPS 设备将压缩轨迹数据通过第三方带宽(GPRS、3G、4G 等)上传至云端数据中心(如图③标记的数据流)。如预期那样，VTracer 不仅能减少数据的存储成本，还能节省上传数据的通信开销。

车载 GPS 设备 GPS 设备以较高频率采集移动车辆的位置（如每 6 秒），然后将轨迹数据发送到手机端上进行数据压缩，最后 GPS 设备将从手机端传来的压缩轨迹数据上传到云端数据中心。传统方法不是压缩数据，而是简单地降低 GPS 设备端采集信息的频率。与传统方法相比，VTracer 系统不仅对 GPS 噪声具有良好的鲁棒性，而且能保存更多完整的轨迹数据。

移动电话 由于手机具有更强的计算能力，因此它扮演了本地计算服务器的角色。手机主要负责两个资源密集型的计算任务（即在线轨迹匹配和轨迹压缩）。另外，为了让驾驶员了解出行路径和周围的空间环境(例如兴趣点)，我们进一步开发了前端可视化功能，通过调用 AMap 提供的 api 接口，将相关信息显示在屏幕上。注意，为了节省手机电量，用户可以选择将这项功能禁用。

数据中心 数据中心是存储大规模车辆实时生成的压缩轨迹数据的数据仓库，这些数据可以支持广泛的基于地理位置的城市智慧服务，比如位置查询、跟踪以及数据可视化。

5.3 优化系统效率的两种策略

VTracer 系统部署了 SD-Matching 算法和 HCC 算法，从第 3 章的实验评估可知，在效率方面，虽然 SD-Matching 远胜于当前最优秀的地图匹配算法 ST-Matching，但它的时间复杂度仍远高于 HCC 算法，它占用了至少 83%的系统总计算时间。比如当l 30 时，系统消耗了约 91%的总计算时间在轨迹匹配上。由于移动在线环境和 GPS 设备的高采样频率对 VTracer 系统的计算效率提出了严格的要求，因此在实现 VTracer 系统时，我们需要通过系统的计算成本，优化系统的效率。为了达到这个目的，其中最关键步骤就是通过工程手段降低 SD-Matching 算法的计算时间，提高它的计算效率。

在介绍优化策略前，我们简单回顾一下第 2 章介绍的 SD-Matching 算法,它是一种利用给原始轨迹的空间信息和方向信息来推断出车辆真实行驶路线的算法。输入算法的原始轨迹数据流通常被分割成长度相同且非重叠的片段(即每个片段 GPS 点的数量相同但无重复)。对于每个片段，为了得到其对应的匹配轨迹片段，我们使用包含三个阶段的 SD-Matching 算法。具体来说，第一阶段是识别给定 GPS 点的*top* k 个候选边，第二阶段是推断连续两个 GPS 点间的路径，最后一个阶段是筛选给定轨迹片段的匹配路径。在第二阶段，我们已经充分利用车辆航向信息来缩小路径搜索区域和作为有效的引导器，极大的加速了地图匹配的速度。因此，在本节中我们从剩余两个阶段针对 SD-Matching 算法的效率进行优化，即采用索引技术加速候选边的搜索和利用剪枝技术加速候选路径的筛选。

策略 1：采用索引技术加速候选边的搜索 由于 GPS 设备存在测量误差，因此给定 GPS 点通常不“坐落”在道路上。为了识别它的真实位置，我们既考虑了点到“附近”边的欧式距离，也考虑了 GPS 点处的车辆航向与“附近”边方向的夹角差值。根据获得的空间距离和角度差值，我们进一步计算给定 GPS 点附近的若干条边(距离点小于 100 米)是正确匹配边的概率，并最终选择出其中概率最大的 k 条边。从第 2 章的实验评估“不同阶段的时间成本”，我们可知，这一阶段极其耗时，这是因为在识别 $top\quad k$ 个候选边时，SD-Matching 需要遍历给定 GPS 点到路网中所有边的距离。因为城市路网中的海量边，如北京路网约有 15 万条边，所以 SD-Matching 算法的效率被严重降低。为了加快侯选边的识别速度，我们采用了一个简单的网格索引策略，具体步骤如下：

① 我们把整个城市路网切割为 $N\quad N$ 个大小相等的长方形网格。在 VTracer 中，我们设置 $N\quad 30$，每个网格的面积约为 1.69 平方公里。

② 我们很容易获得落在切割后网格中的节点和边，我们称它们为路网碎片。路网碎片要远小于整个城市路。注意，若一条边横跨多个网格，那么多个网格的路网碎片都需要包含该边。

③ 根据给定 GPS 点的经纬度坐标，确定其落在的网格以及网格中相应的路网碎片，我们以路网碎片代替整个城市路网来加速识别过程。

策略 2：剪枝技术加速候选路径的筛选 理论上，若原始轨迹片段长度为 l，那么有 $k^{2(l\ 1)}$ 个可能的候选路径，最终的匹配路径是综合概率累积最高的那一条。然而，即使当 k 和 l 较小时，候选轨迹的数量也是极大的。我们需要计算其中每一条路径的累计概率，该过程过于耗时。因此，为了加速路径的筛选，我们使用了剪枝算法对筛选进行效率优化，具体步骤如下：

① 我们构造出有向树，其中树的节点是每个 GPS 点的候选边，树的有向边是在阶段 2 中获得的连接两个候选边的路径。②基于构建的树，我们应用一个树的修剪策略，即迭代删除无效节点来获得更简单的树。如果节点的入度或者出度等于 0，那么说明该节点是无效的（起始节点和结束节点除外）。

算法 5.1 详细描述了树的修剪策略。首先为输入树构造一个邻接矩阵(第 1 行)。第 2～11 行给出了无效节点识别和节点删除的迭代操作。当没有找到新的无效节点时，迭代将被终止(即第 2 行的保持不变)。在循环时，当发现无效节点时(第 3 行和第 7 行)，无效节点将从树上删除，并且树对应的相邻矩阵也将更新(第 4～5 行和第 8～9 行)。注意，当从树中删除一个节点时，它的所有连接边也一并删除。最后基于构建的树，利用深度优先搜索(DFS)算法找到所有的候选轨迹。

算法 5.1 树的剪枝算法

1： $A(T)$ //构造树 T 的邻接矩阵

2：while $|A(T)|$ **continues to reduce do**

3： **if** $IsIn\deg reeZero(n_i)$ **then**

4： $T \quad T \quad n_i$；//从 T 中移走节点 n_i

5： $Update(A(T))$ ；

6： **end if**

7： **if** $IsOut\deg reeZero(n_j)$ **then**

8： $T \quad T \quad n_j$；//从 T 中移走节点 n_j

9： $Update(A(T))$ ；

10： **end if**

11:end while

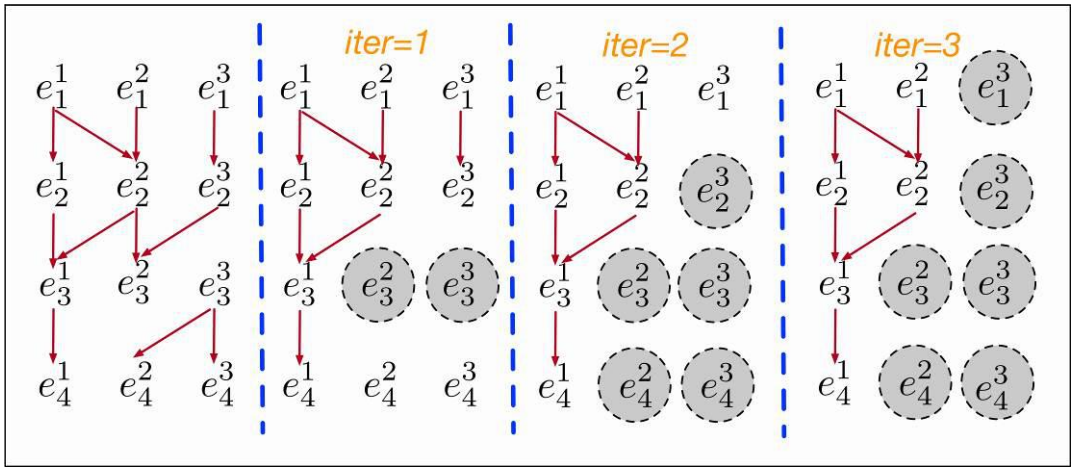


图 5.2 修剪策略的说明性示例

Figure 5.2 An illustrative example on simple graph pruning strategy

为使算法 5.1 更加清晰直观，我们使用一个说明性的例子，如图 5.2 所示。为便于演示，我们将 l 和 k 分别设置为 4 和 3。图中树的深度正好等于轨迹片段的长度 l，树的第 i 层对应第 i 个 GPS 点。树中每一层的节点数是相同的，等于候选边的数量 k。对两个连续 GPS 点的一对候选边，如果阶段 2（不）返回路径，则（不）存在箭头。在第一次算法迭代中，图中的节点 e_3^2 和 e_3^3 将被删除，这是因为它们被标识为无效节点；在第二次迭代中，三个新的节点将被识别为无效并被删除（e_2^3、e_4^2 和 e_4^3）；在最后一次迭代中，e_1^3 将被删除。可以看出，与原始树相比，新约简后的树要简单得多，这将使寻路变得更快捷高效。

5.4 实验评估

在本节中，我们将展示 VTracer 系统产生的压缩效果：① 系统的轨迹匹配质量高，② 系统压缩率高，③ 系统效率高且轻量级，可以在移动环境下工作且实时响应。同时，我们还做了一个关于在不同密度路网下的轨迹匹配和压缩性能的案例研究。

5.4.1 实验设置

① 在真实环境中的部署 图 5.3 为 VTracer 系统在真实环境中的部署情况。该系统通过两款智能手机（华为 P9 和 P10）部署在 2015 年生产的大众帕萨特汽车上。华为 P9 模拟 GPS 设备，用于实时采集车辆的 GPS 数据流，而华为 P10 作为主要的计算平台，负责繁重的计算任务。华为 P10 的内存为 4GB，具有强大的计算能力。在启动系统之前，两款手机需要通过连接蓝牙进行双向通信。

图 5.3 在实际环境中部署 VTracer 系统

Figure 5.3 Deployment of VTracer system in the real environment

我们开发了一个在 Android 操作系统上运行的应用程序来控制手机内置的 GPS 传感器，从而模拟车载 GPS 设备来收集轨迹数据，并通过编程定制默认采样率为 6 秒。数据采集器固定在手机支架上，确保行驶时它的指向始终与车辆的航向一致，如图 5.3 所示。数据流通过已建立的蓝牙链路发送到华为 P10 手机上，在这款手机上，我们还开发了另一款安卓应用，名为 TrajCompressor 的应用，该应用可以对不断流入的 GPS 数据进行在线压缩。

② 数据准备 为评估 VTracer 系统性能，我们准备了两个来自中国重庆市的主要数据集。第一个是路网，是从 OpenStreetMap 上下载的。据统计，它包含 30691 个边和 29461 个节点。第二个是原始 GPS 轨迹数据，包含大约 7600

个原始 GPS 点，这些点是从 2018 年 3 月 17 日到 4 月 21 日收集的。轨迹数据集的累计行驶距离约为 513 公里。

基准算法 采用两种基准算法系统，即 PRESS 和 CCF，以证明 VTracer 系统的良好性能。此外，我们在 Android 平台上实现它们并开发相应的应用步骤，即 PRESS 和 CCF。

PRESS 系统 对于由 n 个边组成的给定的匹配轨迹片段，压缩轨迹中每两个连续对之间的边精确地遵循最短路径。PRESS 系统需要使用 Dijkstra 算法计算最短路径 $n \quad 2$ 次。

CCF 系统 对于由 n 个边组成的给定的匹配轨迹片段，从起始边到结束边，CCF 系统仅保留出边的 ID 和代码（按顺时针顺序）。CCF 系统需要从包含数千个项目的预定义数据库中查找相应的边代码不超过 $n \quad 2$ 次。

③ 评价指标 在有效性方面，我们使用两个指标来衡量 VTracer，即用于量化地图匹配质量的平均匹配精度 (acc)；用于测量轨迹压缩性能的压缩率 (cr)。

我们将 acc 定义为 1 与错误匹配 GPS 点数量与观测 GPS 点总数之比的差值，如式（5.1）所示。

$$acc \quad 1 \quad \frac{N_{wm}}{N_{total}} \tag{5.1}$$

其中 N_{wm} 和 N_{total} 分别表示错误匹配的 GPS 点个数和观测到的 GPS 点总数。如果一个给定的 GPS 点没有匹配到“真实”边，那么它将被报告为错误匹配。值得注意的是，由于车辆的真实行驶轨迹只能事后才知道，所以“即时”计算地图匹配精度值是具有挑战性的，甚至是不可能的。因此，我们招募了若干名志愿者来手动标注每个 GPS 点的“真实”边。

在以往的研究中，cr 通常定义为应用数据压缩算法前后轨迹数据占用的磁盘空间之比。但是，对于 VTracer 系统，GPS 点一旦产生，就会立即发送到本地计算中心（华为 P10 手机）进行处理。在手机一端，系统从不将数据存储在硬盘中。为了使 cr 的计算可行，我们使用如下式来进行近似，如式（5.2）所示。

$$cr \quad \frac{N_{total} \quad c_1}{N_{total} / l \quad c_2 \quad N_{edge} \quad c_3} \tag{5.2}$$

其中 N_{total} 为本地计算中心接收和处理的 GPS 总点数；N_{total} / l 为 N_{total} / l 的整数值，也为轨迹片段数量；N_{edge} 是压缩轨迹中保留的边数；c_1 是粗略估计的一个 GPS 点所占字节的常数值；同样，c_2 和 c_3 分别表示时间和一条边占用的字节常量。在我们的实验中，我们设 $c_1 \quad 40$，$c_2 \quad c_3 \quad 9$。如果 cr 越大，那么压缩性能越好。与 acc 相比，实时监测 N_{total} 和 N_{edge} 的值来计算 cr，这种操作是简单可行的。更具体地说，我们只需要设置两个计数器，就可以得到在轨迹压缩过程

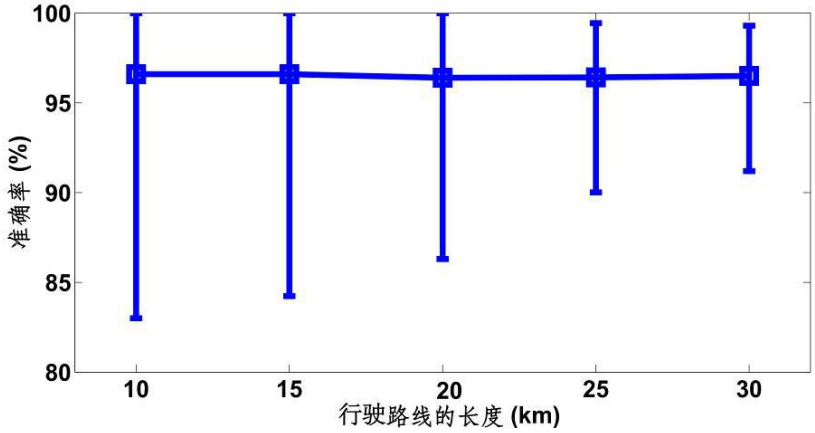
中本地计算中心接收到的 GPS 点的个数和保留的边的个数，这样就可以及时的显示和刷新 *cr* 的值。

为了衡量 VTracer 的效率，我们首先计算了处理每一个原始轨迹片段的总时间成本（即轨迹匹配和轨迹压缩的总时间消耗），然后使用所有轨迹片段时间成本的平均值来量化 VTracer 的整体效率。此外，VTracer 消耗的能量和内存，以及 APP 占用的存储大小也反映了它的效率。

5.4.2 有效性评估

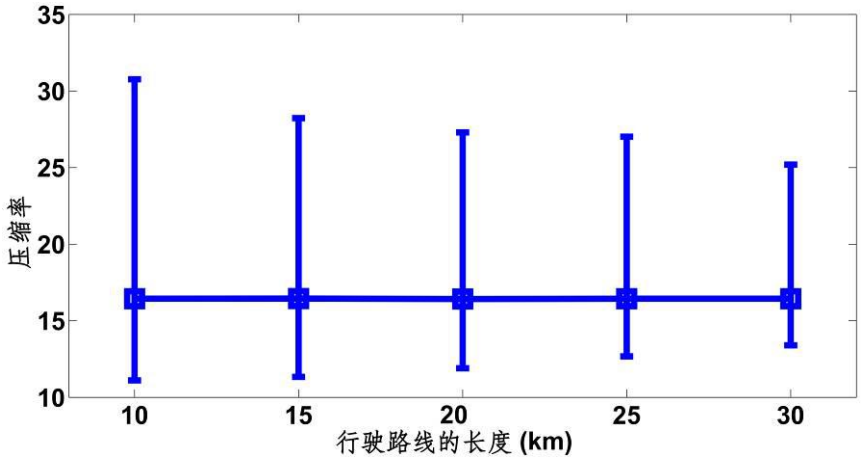
轨迹匹配中的 k 和轨迹压缩中 l 是用户指定的重要参数，会极大影响系统的性能。由于本节主要关注的是实际系统实现与部署，这里我们设定 k　6，l　10。读者可以参考第 2 和第 3 章来了解更多关于算法的细节。

我们感兴趣的问题是，VTracer 系统每次行程的表现是否一致?为了解决这一问题，我们将分别从匹配精度(*acc*)和压缩率(*cr*)两个方面来研究若干轨迹样本。具体来说，先将所有轨迹样本按照行驶距离长短分为五组，即 $0\sim10,10\sim15,15\sim20,20\sim25$， 25 ，然后对属于同一组的轨迹样本，我们统计两个性能指标的平均值、最小值和最大值，结果如图 5.4 所示。从两幅图的结果可以看出，不同行驶距离下的平均性能是相当稳定的。例如，①轨迹匹配精度高，对所有的行程都保持在 95%以上；②压缩率稳定，所有行程压缩率均在 15 以上。此外，我们还可以看到，当行驶距离变长时，两种性能指标的范围都变窄，说明系统运行更加稳定。对这种观察的一种可能的解释是，在短时间内驾驶，车辆很可能仍在同一区域内。在这种情况下，行驶路径的空间环境就非常相似。具体来说，沿线路网或密集或稀疏，导致性能很差或很强。相反，车辆在行驶距离较远时可能会穿越不同的区域，在这种情况下，车辆行驶路线的空间环境很均衡。综上所述，对于距离较小的行驶路径，压缩性能的差异更大，而对于距离较大的行驶路径，压缩性能的差异会更小。



（a）轨迹匹配的准确度

(a) The accuracy of trajectory mapping



（b）轨迹压缩的压缩率

(b) The compression ratio of trajectory compression

图 5.4 不同组轨迹样本的 *acc* 和 *cr* 结果

Figure 5.4 Results of *acc* and *cr* for rides belonging to different groups

我们还比较了 VTracer 和基准算法的压缩率，实验结果如表 5.1 所示。从表中可以看出，VTracer 在三个系统之间实现了压缩率，CCF 与之具有相似的压缩率，而 PRESS 比 VTracer 具有更好的压缩性能。因此，如果嵌入式系统仅考虑有效性能，则可以在 APP 中部署具有不同压缩原理的系统，因为它们具有相似的压缩率。

表 5.1 不同轨迹压缩系统的压缩率

Table 5.1 Compression ratio of different systems

系统	VTracer	PRESS	CCF
cr	15.85	16.67	15.69

5.4.3 效率评估

时间成本 图 5.5 为采用或放弃对第三阶段的树进行修剪策略时，压缩每个轨迹片段的时间成本的累积分布函数结果。可以看出，应用剪枝策略后，时间开销变小，系统效率提高。因此，引入剪枝策略在降低时间成本方面效果显著。更具体地说，使用修剪策略后，VTracer 系统最多消耗不到 200 毫秒(约占所有轨迹批的 80%)，对于在线应用步骤来说，这已经足够快了。而没有修剪策略，这一比例不到 65%。从图中还可以看出，修剪策略前后 VTracer 系统的最大时间开销分别为 600 毫秒和 400 毫秒。由于采样时间间隔为 6 秒，无论是否采用修剪策略，系统都可以及时响应(即，在接收到新生成的数据点之前完成对轨迹

片段的压缩)。因此，对于更密集的 GPS 轨迹（采样间隔更小），VTracer 系统仍可以对其进行在线压缩。

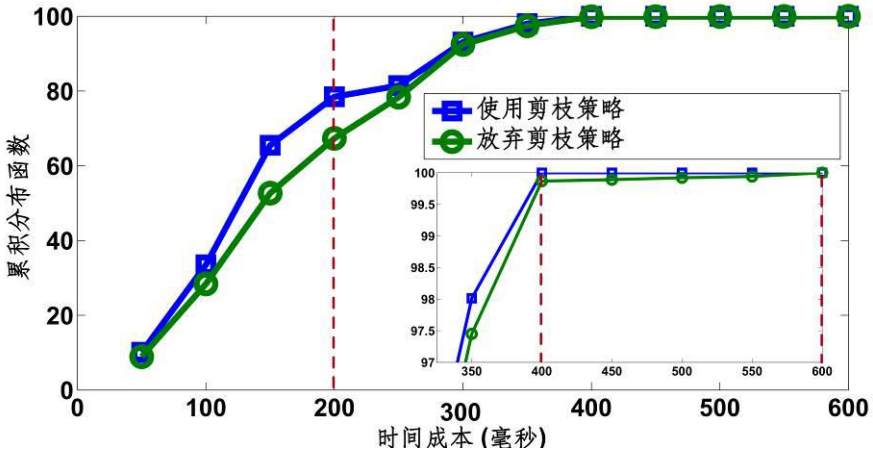


图 5.5 有无剪枝策略的时间消耗对比

Figure 5.5 Comparison results on the time cost with/without the pruning strategy

表 5.2 是三个轨迹压缩系统压缩轨迹片段时平均时间成本和最大时间成本的对比。可以看出，VTracer 系统比 CCF 消耗更少的计算时间。原因是 CCF 需要额外的操作，即从包含数千个的预定义数据库中查找出边对应的编码。与其他两个系统相比，PRESS 需要的计算时间最长，这是因为 PRESS 需要在线计算多个最短路径，该过程是非常耗时的。PRESS 的最大时间成本约为 9 秒，这大于采样时间（即 6 秒）。因此我们可以得出结论，VTracer 系统更适合在线轨迹压缩系统，因为它在压缩率和计算时间成本之间进行了很好的权衡。

表 5.2 不同轨迹压缩系统的时间成本

Table 5.2 Time cost of different systems

系统	VTracer	PRESS	CCF
平均时间消耗（毫秒）	146	1483	359
最大时间消耗（毫秒）	400	9348	671

能耗 我们采用电量(单位 mAh)来表示手机的能耗。为了准确的测量能耗，我们只需要关掉其他所有不必要的 APP，只使用华为 Android 操作系统中的嵌入式 APP，每小时监测一次 TrajCompressor APP 的能耗。图 5.6 显示了工作时间为 12 小时的 TrajCompressor APP 的两条能耗曲线。一条曲线对应于启用可视化工具时的情况，另一条曲线对应于禁用可视化工具时的情况。很容易看出，

如果启用前端可视化器，会消耗更多的能量。因此，默认情况下禁用可视化工具可以节省手机电量。同时，无论启用还是禁用前端可视化器，随着工作时间的增加，能耗几乎呈线性增长。华为 P10 智能手机的电池容量为 3000 毫安，TrajCompressor APP 可以连续工作约 140 小时。而如果禁用了可视化器，它可以工作长达 280 小时。综上所述，TrajCompressor APP 的能耗在实际应用场景中是可以接受的。

我们还通过计算 10 小时运行不同轨迹压缩应用时的总耗电量来比较它们的能耗。实验结果如表 5.3 所示。与效率研究类似，TrajCompressor APP 比 PRESS 和 CCF 应用更节能，这是因为 PRESS 和 CCF 应用需要消耗更多的电力来完成计算任务。

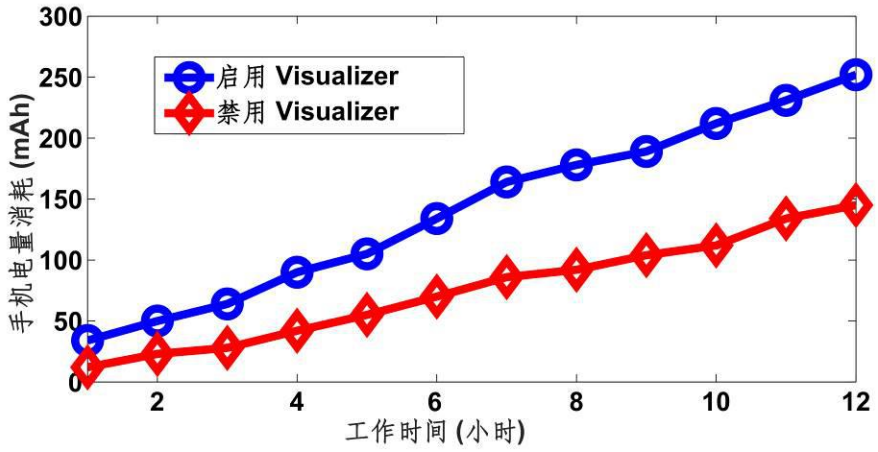


图 5.6 当可视化功能被禁用和启用时，TrajCompressor 的工作时间与手机电量消耗结果

Figure 5.7 Energy consumption comparison results of TrajCompressor app with the working time when the visualizer is disabled and enabled

表 5.3 不同轨迹压缩应用的能量消耗

Table 5.3 Engery cost of different systems

应用程序	TrajCompressor	PRESS	CCF
能量消耗 （mAh）	210	920	245

内存消耗 与能耗研究类似，我们也依赖于另一个嵌入式 APP 来监测 TrajCompressor APP 的内存消耗(单位为 MB)。与能耗相比，内存消耗对工作时间更为敏感，因此我们每 10 分钟监测一次该参数。图 5.7 为工作 12 小时的手机内存消耗结果。可以看出，内存消耗随着工作时间的增加而波动。最大内存

消耗约为 76 MB，仅占整个内存的 2.0%（整个内存大小为 4 GB）。综上所述，内存消耗也是可以接受的，TrajCompressor APP 几乎不影响其他移动应用。

我们记录了三个应用步骤的内存消耗，结果（即平均和最大内存消耗）如表 5.4 所示。与效率研究类似，TrajCompressor APP 比其他两个应用 PRESS 和 CCF 占用更少的内存，这是因为查找边的代码（CCF）和计算最短路径（PRESS）的额外操作将消耗更多的手机计算资源。

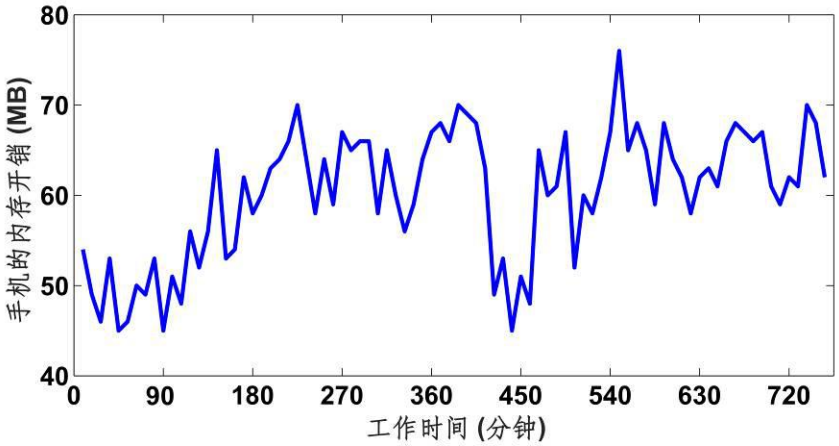


图 5.7 TrajCompressor APP 的内存消耗

Figure 5.7 Memory consumption of TrajCompressor app

表 5.4 不同轨迹压缩应用的存储空间消耗

Table 5.4 Storag cost of different systems

应用程序	TrajCompressor	PRESS	CCF
平均存储空间 （MB）	60	97	86
最大存储空间（MB）	76	161	121

应用程序大小 因为用户更容易接受占用存储空间小的应用，因此我们也对 TrajCompressor APP 的大小感兴趣。在本研究中，我们调查了中国大陆五个不同城市(即重庆、北京、上海、深圳、广州)TrajCompressor APP 的大小。图 5.8 为五个城市的 APP 的大小统计。可以看出，重庆市的 APP 占用的存储空间最小，仅占 2.39 MB 的空间。北京市的 APP 占用的存储空间最大，接近 12 MB。不同城市的 APP 占用的存储空间不同是因为不同城市的路网规模不同。路网是轨迹匹配和压缩中必不可少的数据，应该在系统启动之前下载。如果你想添加更多的城市，就像一些商业 GPS 导航应用一样，APP 的大小也会增加。

我们还比较了在重庆市部署的三个应用的大小，比较结果显示在表 5.5 中。可以看出，应用 TrajCompressor APP 和 PRESS 的大小几乎相同，但 CCF 要大得多，这是因为与其他两个应用步骤相比，CCF 存储了预定义的数据库。具体来说，每个交叉口的每个边都按顺时针顺序编码并提前保存在数据库中以确保及时响应。

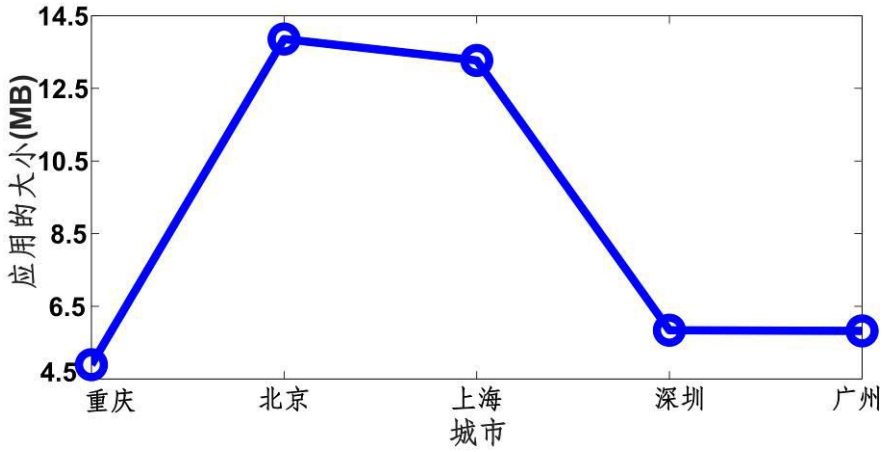


图 5.8 不同城市的 TrajCompressor APP 的大小

Figure 5.8 App size of TrajCompressor in different cities

表 5.5 不同轨迹压缩应用的 APP 大小

Table 5.5 Size of trajectory compression apps

应用程序	TrajCompressor	PRESS	CCF
APP 大小（MB）	4.85	4.83	7.89

5.4.4 案例研究

我们专门选取了三个区域内三次出行产生的 GPS 轨迹数据样本进行案例研究。图 5.9 给出了在 AMap 上的三次出行的具体路径和对应区域，并且不同地区的路网密度有很大差异。例如，区域 III(即重庆市中心区域)的路网密度分别是第区域 I 和区域 II 的 2.8 倍和 1.5 倍。表 5.6 还提供了关于这三个区域更多的统计数据。注意，一个区域的边密度是由该区域的总边数与区域面积之比得到的，面积的单位是平方公里。

表 5.6 关于三个地区的统计信息

Table 5.6 Statistic information of the three regions

区域	I	II	III
面积	7.23	7.12	4.63
#边	1578	2339	2834
#节点	1504	2056	2687
边密度	218.26	336.94	612.10

VTracer 对这三种路径的压缩性能结果如图 5.9 所示。图中黑点为原始 GPS 点；绿色实线是指所匹配的轨迹；蓝色实线段是指轨迹压缩后保留的边。在每个图的左下角还显示了一些额外的信息，包括当前压缩率、车辆位置、速度和车辆航向。

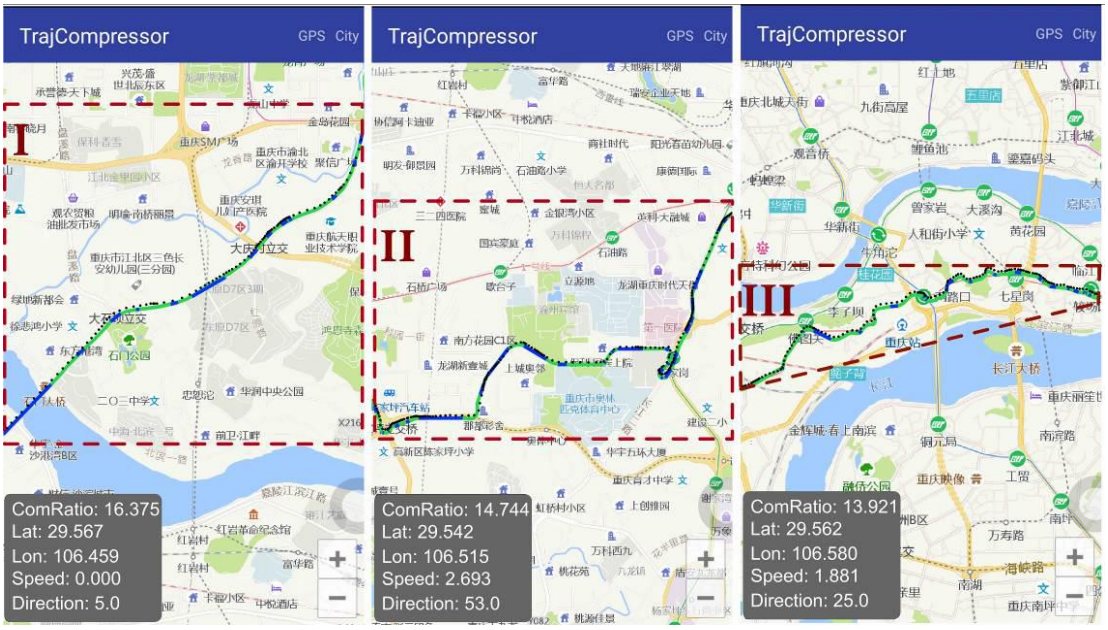


图 5.9 三个出行轨迹的压缩结果

Figure 5.9 Performance results of three trajectories

我们还在表 5.7 中展示了三种情况下的详细性能指标，包括路径长度、轨迹匹配的准确性、轨迹压缩的压缩率以及平均时间成本。可以看出，在路网稀疏的区域，VTracer 的性能稍好，而且消耗的时间较少。例如，在案例 1 中，轨迹匹配的精度完全等于 1，压缩率也是最高的，而且时间成本在所有案例中都是最小的。这可能是因为①路网结构简单，使得地图匹配更容易、更准确；②在路网稀疏的地区，因为十字路口较少，所以转弯次数较少，因此压缩率更高。

表 5.7 不同路网密度的结果

Table 5.7 Result under different road network densities

案例	距离（km）	准确度（%）	压缩率	平均时间
1	4.6	100	16.375	63.23
2	7.5	96.12	14.744	179.42
3	8.1	94.17	13.921	263.05

5.5 章节小结

在本章中，我们设计了一款名为 VTracer 的轨迹压缩系统，目的是向数据中心发送更少的数据。与先前对 GPS 噪声敏感且轨迹数据频率较低的系统相比，我们用 VTracer 系统能在 GPS 设备端收集密集的轨迹数据，发送更小更完整但信息量更多的压缩轨迹数据。受移动边缘计算的启发，我们将繁重的在线轨迹压缩任务迁移到附近的移动电话设备上。实际部署中的良好效果证明了我们开发的 VTracer 系统在地图匹配、压缩率、内存、能耗和应用程序大小方面的卓越性能。

6 总结与展望

车辆时空轨迹数据是实现众多城市智慧交通服务的数据基石，随着车载GPS设备的普及和移动互联网的成熟，嵌入车辆中海量GPS设备源源不断地向云端数据中心发送实时轨迹数据。但是，如此庞大规模的轨迹数据引起了一系列问题，如过大的存储和通信开销，以及冗余的数据严重阻碍数据的挖掘与计算任务。更糟糕的是，未来持续增长的公共交通和私家车会安装GPS设备，持续上传数据加剧了问题的严重性。为解决以上问题，本文设计了轨迹匹配与压缩的算法，本文的主要贡献体现在以下几个方面：

① 在轨迹匹配方面，我们探索车辆航向在地图匹配中的可用性，创新地利用该信息加速地图匹配的速度（效率）与保证地图匹配的质量（准确性）。另外，为了保证在移动环境下系统的效率，我们进一步地针对SD-Matching算法做了工程方面的效率优化，即使用索引技术和剪枝技术等。

② 在空间轨迹压缩方面，我们创新性地探索车辆航向变化在轨迹压缩上的可用性，并利用有趣的发现来实现高质量的轨迹压缩，我们设计的HCC算法能够在效率和压缩率上取得良好的平衡。

③ 在时间轨迹压缩方面，我们提出一种时间维度上全新轨迹表示，包含三个部分，距离序列（D）、加速度和瞬时速度序列（AV）和时间序列（T）。我们针对这三个部分分别开发了高质量压缩器。另外，我们还是保存和压缩时变速度信息的先驱，时变速度信息对理解驾驶风格和掌握城市交通状态至关重要。

④ 我们不满足于开发出新算法，还将算法部署在真实的移动环境下。因为移动环境下缺乏充足算力，以及严苛环境对系统效率要求更高，我们受移动边缘计算的原理启发，创新地将繁重的计算任务从计算能力有限的GPS设备迁移到附近的手机端，这一举措不仅能弥补移动环境下算力的不足，还能释放对GPS设备采样频率的限制，允许更密集的采样频率，提高轨迹匹配准确度。

⑤ 无论是轨迹匹配还是压缩，还是真实移动环境下设计的系统，我们都对它们进行了广泛的实验评估，实验结果证明SD-Matching算法、HCC算法和D压缩器、AV压缩器、T压缩器均优于当前基准算法。另外，VTracer系统在真实环境下的性能评估结果也远优于同类系统。

虽然以上算法能够在轨迹匹配和压缩中取得较好的效果，但还可以从以下几个方面进行优化。

① 在地图匹配研究中，我们还可以通优化路网来加速地图匹配算法。举个例子，路网中存在大量的出度和入度皆为1的边，这意味着经过这些边的车辆

只有唯一的行驶路径，因此无需路径搜索。基于此观察，若将此类相邻的边连起来，则能有效减少路径搜索时间。除此之外，还有许多类似的有趣观察，我们在未来将针对这些现象进行分析和总结，通过优化路网来加速匹配。

② 另外，地图匹配算法的准确度在很多情况下并不是 100%，这意味着存在失效的轨迹匹配案例，我们在未来希望找到并分析这些失效案例，研究为什么 SD-Matching 算法对它们无效，借此进一步改进 SD-Matching 算法，提高算法的鲁棒性和准确度。

③ 我们计划拓宽和深化在时间维度上的轨迹压缩研究。首先，我们打算研究时间效应（如一天中的不同时刻）对压缩性能的影响。其次，在现阶段设计 AV 压缩器时，只考虑利用空间相似度建立加速度的霍夫曼森林。而未来我们计划探索不同驾驶风格的可用性，并检查它是否能提高压缩率。

④ 最后，现在的 VTracer 系统并没有移植时间维度上轨迹压缩算法框架，我们计划下一阶段将 DVAT 算法移植到真实系统中，并在真实的移动环境下测试其性能。

参考文献

[1] 李德仁，姚远，邵振峰．智慧城市中的大数据[J]．武汉大学学报，2014, 39(6): 631-640.

[2] 李德仁，邵振峰，杨小敏．从数字城市到智慧城市的理论与实践[J]．地理空间信息，2011, 6: 1-5.

[3] 王静远，李超，熊璋，et al．以数据为中心的智慧城市研究综述[J]．计算机研究与发展，2014, 51(2): 239-259.

[4] 陈柳钦．智慧城市：全球城市发展新热点[D]. 2011.

[5] Ding Y, Chen C, Zhang S, et al. GreenPlanner: Planning personalized fuel-efficient driving routes using multi-sourced urban data[C]. 2017 IEEE International Conference on Pervasive Computing and Communications (PerCom), 2017: 207-216.

[6] Chen C, Wang Z, Guo B. The road to the Chinese smart city: progress, challenges, and future directions[J]. IT Professional, 2016, 18(1): 14-17.

[7]齐观德，潘纲，李万坚．当出租车轨迹挖掘遇见智能交通 [J][D]. 2013.

[8] 左一萌，林学练，马帅，et al．路网感知的在线轨迹压缩方法[J]．软件学报, 2018, 29(3): 734-755.

[9] 马连韬，王亚沙，彭广举，et al．基于公交车轨迹数据的道路 GPS 环境友好性评估[J]．计算机研究与发展, 2016, 53(12): 2694-2707.

[10] 赵婧，张渊，李兴华, et al．基于轨迹频率抑制的轨迹隐私保护方法[J]．计算机学报，2014, 37(10): 2096-2106.

[11] Castro P S, Zhang D, Chen C, et al. From taxi GPS traces to social and community dynamics: A survey[J]. ACM Computing Surveys, 2013, 46(2): 17.

[12] Chen C, Zhang D, Castro P S, et al. iBOAT: Isolation-based online anomalous trajectory detection[J]. IEEE Transactions on Intelligent Transportation Systems, 2013, 14(2): 806-818.

[13] Chen C, Jiao S, Zhang S, et al. TripImputor: real-time imputing taxi trip purpose leveraging multi-sourced urban data[J]. IEEE Transactions on Intelligent Transportation Systems, 2018, (99): 1-13.

[14] Zhang M, Chen C, Wo T, et al. SafeDrive: online driving anomaly detection from large-scale vehicle data[J]. IEEE Transactions on Industrial Informatics, 2017, 13(4): 2087-2096.

[15] Gudmundsson J, Katajainen J, Merrick D, et al. Compressing spatio-temporal trajectories[J]. Computational geometry, 2009, 42(9): 825-841.

[16] Han Y, Sun W, Zheng B. COMPRESS: A comprehensive framework of trajectory compression in road networks[J]. ACM Transactions on Database Systems, 2017, 42(2): 11.

[17] Ji Y, Zang Y, Luo W, et al. Clockwise compression for trajectory data under road network constraints[C]. 2016 IEEE International Conference on Big Data (Big Data), 2016: 472-481.

[18] Muckell J, Olsen P W, Hwang J-H, et al. Compression of trajectory data: a comprehensive evaluation and new approach[J]. GeoInformatica, 2014, 18(3): 435-460.

[19] Song R, Sun W, Zheng B, et al. PRESS: A novel framework of trajectory compression in road networks[J]. Proceedings of the VLDB Endowment, 2014, 7(9): 661-672.

[20] Sun P, Xia S, Yuan G, et al. An overview of moving object trajectory compression algorithms[J]. Mathematical Problems in Engineering, 2016, 2016.

[21] Chen Y, Jiang K, Zheng Y, et al. Trajectory simplification method for location-based social networking services[C]. Proceedings of the 2009 International Workshop on Location Based Social Networks, 2009: 33-40.

[22] Long C, Wong R C-W, Jagadish H. Trajectory simplification: On minimizing the direction-based error[J]. Proceedings of the VLDB Endowment, 2014, 8(1): 49-60.

[23] Zheng K, Zheng Y, Xie X, et al. Reducing uncertainty of low-sampling-rate trajectories[C]. 2012 IEEE 28th International Conference on Data Engineering, 2012: 1144-1155.

[24] Goh C Y, Dauwels J, Mitrovic N, et al. Online map-matching based on hidden markov model for real-time traffic sensing applications[C]. 2012 15th International IEEE Conference on Intelligent Transportation Systems, 2012: 776-781.

[25] Wang L, Zhang D, Wang Y, et al. Sparse mobile crowdsensing: challenges and opportunities[J]. IEEE Communications Magazine, 2016, 54(7): 161-167.

[26] Cho W, Choi E. A basis of spatial big data analysis with map-matching system[J]. Cluster Computing, 2017, 20(3): 2177-2192.

[27] Pink O, Hummel B. A statistical approach to map matching using road network geometry, topology and vehicular motion constraints[C]. 2008 11th International IEEE Conference on Intelligent Transportation Systems, 2008: 862-867.

[28] Quddus M A, Ochieng W Y, Noland R B. Current map-matching algorithms for transport applications: State-of-the art and future research directions[J]. Transportation research part c: Emerging technologies, 2007, 15(5): 312-328.

[29] Lou Y, Zhang C, Zheng Y, et al. Map-matching for low-sampling-rate GPS trajectories[C]. Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems, 2009: 352-361.

[30] Ding Y, Chen C, Zhang S, et al. TrajCompressor: An Online Map-matching-based Trajectory Compression Framework Leveraging Vehicle Heading Direction and Change[J]. IEEE Transactions on Intelligent Transportation Systems, 2019, accepted.

[31] 李清泉，胡波，乐阳．一种基于约束的最短路径低频浮动车数据地图匹配算法[J]．武汉大学学报• 信息科学版, 2013, 38(7): 805.

[32] Castro P S, Zhang D, Li S. Urban traffic modelling and prediction using large scale taxi GPS traces[C]. International Conference on Pervasive Computing, 2012: 57-72.

[33] Li Y, Huang Q, Kerber M, et al. Large-scale joint map matching of GPS traces[C]. Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2013: 214-223.

[34] Qu Q, Liu S, Zhu F, et al. Efficient online summarization of large-scale dynamic networks[J]. J IEEE Transactions on Knowledge

Data Engineering, 2016, 28(12): 3231-3245.

[35] Bentley J L, Maurer H A. Efficient worst-case data structures for range searching[J]. Acta Informatica, 1980, 13(2): 155-168.

[36] Bernstein D, A K. An Introduction to Map Matching for Personal Navigation Assistants. [J], 1996.

[37] Greenfeld J S. Matching GPS observations to locations on a digital map[C]. 81th annual meeting of the transportation research board, 2002: 164-173.

[38] White C E, Bernstein D, Kornhauser A L. Some map matching algorithms for personal navigation assistants[J]. Transportation research part c: emerging technologies, 2000, 8(1-6): 91-108.

[39] Velaga N R, Quddus M A, Bristow A L. Developing an enhanced weight-based topological map-matching algorithm for intelligent transport systems[J]. Transportation Research Part C: Emerging Technologies, 2009, 17(6): 672-683.

[40] Kang W, Li S, Chen W, et al. Online map-matching algorithm using object motion laws[C]. 2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart computing (hpsc), and ieee international conference on intelligent data and security (ids), 2017: 249-254.

[41] Yuan J, Zheng Y, Zhang C, et al. An interactive-voting based map matching algorithm[C]. Proceedings of the 2010 Eleventh International Conference on Mobile Data Management, 2010: 43-52.

[42] Bierlaire M, Chen J, Newman J. A probabilistic map matching method for smartphone GPS data[J]. Transportation Research Part C: Emerging Technologies, 2013, 26: 78-98.

[43] Koller H, Widhalm P, Dragaschnig M, et al. Fast hidden Markov model map-matching for sparse and noisy trajectories[C]. 2015 IEEE 18th International Conference on Intelligent Transportation Systems, 2015: 2557-2561.

[44] Newson P, Krumm J. Hidden Markov map matching through noise and sparseness[C]. Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems, 2009: 336-343.

[45] Zhou X, Ding Y, Tan H, et al. HIMM: An HMM-based interactive map-matching system[C]. International Conference on Database Systems for Advanced Applications, 2017: 3-18.

[46] Wei H, Wang Y, Forman G, et al. Fast Viterbi map matching with tunable weight functions[C]. Proceedings of the 20th International Conference on Advances in Geographic Information Systems, 2012: 613-616.

[47] Uthayakumar J, Vengattaraman T, Dhavachelvan P. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications[J]. Journal of King Saud University-Computer Information Sciences, 2018.

[48] Douglas D H P T K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature[J]. Cartographica: the international journal for geographic information and geovisualization, 1973, 10(2): 112-122.

[49] Keogh E, Chu S, Hart D, et al. An online algorithm for segmenting time series[C]. Proceedings 2001 IEEE International Conference on Data Mining, 2001: 289-296.

[50] Li B, Zhang D, Sun L, et al. Hunting or waiting? Discovering passenger-finding strategies from a large-scale real-world taxi dataset[C]. 2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011: 63-68.

[51] Liu J, Zhao K, Sommer P, et al. Bounded quadrant system: Error-bounded trajectory compression on the go[C]. 2015 IEEE 31st International Conference on Data Engineering, 2015: 987-998.

[52] Lv C, Chen F, Xu Y, et al. A trajectory compression algorithm based on non-uniform quantization[C]. 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), 2015: 2469-2474.

[53] Meratnia N, Rolf A. Spatiotemporal compression techniques for moving point objects[C]. International Conference on Extending Database Technology, 2004: 765-782.

[54] Ji Y, Liu H, Liu X, et al. A comparison of road-network-constrained trajectory compression methods[C]. 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), 2016: 256-263.

[55] Kellaris G, Pelekis N, Theodoridis Y. Map-matched trajectory compression[J]. Journal of Systems Software, 2013, 86(6): 1566-1579.

[56] Kellaris G, Pelekis N, Theodoridis Y. Trajectory compression under network constraints[C]. International Symposium on Spatial and Temporal Databases, 2009: 392-398.

[57] Liu J, Zhao K, Sommer P, et al. A novel framework for online amnesic trajectory compression in resource-constrained environments[J]. IEEE Transactions on Knowledge Data Engineering, 2016, 28(11): 2827-2841.

[58] Lin X, Ma S, Zhang H, et al. One-pass error bounded trajectory simplification[J]. Proceedings of the VLDB Endowment, 2017, 10(7): 841-852.

[59] Knuth D E. Dynamic huffman coding[J]. Journal of algorithms, 1985, 6(2): 163-180.

[60] Yang X, Wang B, Yang K, et al. A novel representation and compression for queries on trajectories in road networks[J]. IEEE Transactions on Knowledge

Data Engineering, 2018, 30(4): 613-629.

[61] Capon J. A probabilistic model for run-length coding of pictures[J]. IRE Transactions on Information Theory, 1959, 5(4): 157-163.

[62] Rana R, Yang M, Wark T, et al. Simpletrack: Adaptive trajectory compression with deterministic projection matrix for mobile sensor networks[J]. IEEE Sensors Journal, 2015, 15(1): 365-373.

[63] Ziv J, Lempel A. A universal algorithm for sequential data compression[J]. IEEE Transactions on information theory, 1977, 23(3): 337-343.

[64] Schmid F, Richter K-F, Laube P. Semantic trajectory compression[C]. International Symposium on Spatial and Temporal Databases, 2009: 411-416.

[65] Richter K-F, Schmid F, Laube P. Semantic trajectory compression: Representing urban movement in a nutshell[J]. Journal of Spatial Information Science, 2012, 2012(4): 3-30.

[66] Chen C, Ding Y, Xie X, et al. A three-stage online map-matching algorithm by fully using vehicle heading direction[J]. Journal of Ambient Intelligence Humanized Computing, 2018, 9(5): 1623-1633.

[67] Song R, Lu W, Sun W, et al. Quick map matching using multi-core cpus[C]. Proceedings of the 20th International Conference on Advances in Geographic Information Systems, 2012: 605-608.

[68] Tarjan R. Depth-first search and linear graph algorithms[J]. SIAM journal on computing, 1971, 2: I46-I60.

[69] Haklay M, Weber P. Openstreetmap: User-generated street maps[J]. IEEE Pervasive Computing, 2008, 7(4): 12-18.

[70] Yu Z, Xu H, Yang Z, et al. Personalized travel package with multi-point-of-interest recommendation based on crowdsourced user footprints[J]. IEEE Transactions on Human-Machine Systems, 2016, 46(1): 151-158.

[71] Guo B, Chen H, Han Q, et al. Worker-contributed data utility measurement for visual crowdsensing systems[J]. IEEE Transactions on Mobile Computing, 2017, 16(8): 2379-2391.

[72] Chen C, Zhang D, Ma X, et al. Crowddeliver: planning city-wide package delivery paths leveraging the crowd of taxis[J]. IEEE Transactions on Intelligent Transportation Systems, 2017, 18(6): 1478-1496.

[73] Han W, Deng Z, Chu J, et al. A parallel online trajectory compression approach for supporting big data workflow[J]. Computing, 2018, 100(1): 3-20.

[74] Zhang D, Sun L, Li B, et al. Understanding taxi service strategies from taxi GPS traces[J]. IEEE Transactions on Intelligent Transportation Systems, 2015, 16(1): 123-135.

[75] Zhao Y, Shang S, Wang Y, et al. Rest: A reference-based framework for spatio-temporal trajectory compression[C]. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018: 2797-2806.

[76] Gu Y, Qian Z S, Chen F. From Twitter to detector: Real-time traffic incident detection using social media data[J]. Transportation research part C: emerging technologies, 2016, 67: 321-342.

[77] Lee C F, Öberg P. Classification of road type and driving style using obd data[R]. SAE Technical Paper, 2015.

[78] Lee W-H, Tseng S-S, Shieh J-L, et al. Discovering traffic bottlenecks in an urban network by spatiotemporal data mining on location-based services[J]. IEEE Transactions on Intelligent Transportation Systems, 2011, 12(4): 1047-1056.

[79] Chen C, Chen X, Wang L, et al. MA-SSR: A memetic algorithm for skyline scenic routes planning leveraging heterogeneous user-generated digital footprints[J]. IEEE Transactions on Vehicular Technology, 2017, 66(7): 5723-5736.

[80] Zhang D, Li N, Zhou Z-H, et al. iBAT: detecting anomalous taxi trajectories from GPS traces[C]. Proceedings of the 13th international conference on Ubiquitous computing, 2011: 99-108.

[81] Zhao D, Stefanakis E. Integrated compression of vehicle spatio‐temporal trajectories under the road stroke network constraint[J]. Transactions in GIS, 2018, 22(4): 991-1007.

[82] Etienne L, Devogele T, Buchin M, et al. Trajectory Box Plot: a new pattern to summarize movements[J]. International Journal of Geographical Information Science, 2016, 30(5): 835-853.

[83] Dong J-X, Hicks C, Li D. A heuristics based global navigation satellite system data reduction algorithm integrated with map-matching[J]. Annals of Operations Research, 2019: 1-16.

[84] Shi W, Cao J, Zhang Q, et al. Edge computing: Vision and challenges[J]. IEEE Internet of Things Journal, 2016, 3(5): 637-646.

[85] Cao H, Wolfson O, Trajcevski G. Spatio-temporal data reduction with deterministic error bounds[J]. The VLDB Journal—The International Journal on Very Large Data Bases, 2006, 15(3): 211-228.

[86] Stone J V. Information theory: a tutorial introduction[M]. Sebtel Press, 2015.

[87] Mohamed R, Aly H, Youssef M. Accurate real-time map matching for challenging environments[J]. IEEE Transactions on Intelligent Transportation Systems, 2017, 18(4): 847-857.

[88] Satyanarayanan M. The emergence of edge computing[J]. Computer, 2017, 50(1): 30-39.

[89] Yi S, Li C, Li Q. A survey of fog computing: concepts, applications and issues[C]. Proceedings of the 2015 workshop on mobile big data, 2015: 37-42.

[90] Wang M, Shan H, Lu R, et al. Real-time path planning based on hybrid-VANET-enhanced transportation system[J]. IEEE Transactions on Vehicular Technology, 2015, 64(5): 1664-1678.

[91] Tran T X, Hajisami A, Pandey P, et al. Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges[J]. arXiv preprint arXiv:.03184, 2016.

[92] Taleb T, Dutta S, Ksentini A, et al. Mobile edge computing potential in making cities smarter[J]. IEEE Communications Magazine, 2017, 55(3).

[93] Abbas N, Zhang Y, Taherkordi A, et al. Mobile edge computing: A survey[J]. IEEE Internet of Things Journal, 2018, 5(1): 450-465.

[94] Abid H, Chung T C, Lee S, et al. Performance analysis of LTE smartphones-based vehicle-to-infrastrcuture communication[C]. 2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing, 2012: 72-78.

[95] Araniti G, Campolo C, Condoluci M, et al. LTE for vehicular networking: a survey[J]. IEEE communications magazine, 2013, 51(5): 148-157.

附　录

A. 攻读硕士学位期间的主要研究成果（#共同一作）

[1] 作者***, 导师***, Zhang S, et al. GreenPlanner: Planning personalized fuel-efficient driving routes using multi-sourced urban data[C]//2017 IEEE International Conference on Pervasive Computing and Communications (PerCom). IEEE, 2017: 207-216. (EI 检索，CCF-B)

[2] 作者***, 导师***, Xie X, et al. Fuel Consumption Estimation of Potential Driving Paths by Leveraging Online Route APIs[C]//International Conference on Green, Pervasive, and Cloud Computing. Springer, Cham, 2018: 92-106. (EI 检索)

[3] 作者***, 导师***, Xie X, et al. An Online Trajectory Compression System Applied to Resource-Constrained GPS Devices in Vehicles[C]//2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON). IEEE, 2018: 1-3. (EI 检索，CCF-B)

[4] 导师***, 作者***#, Xie X, et al. A three-stage online map-matching algorithm by fully using vehicle heading direction[J]. Journal of Ambient Intelligence and Humanized Computing, 2018, 9(5): 1623-1633. (CCF-C, JCR 三区，中科院 SCI 四区)

[5] 导师***, 作者***#, Xie, X, et al. An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change[J]. IEEE Transaction on Intelligent Transpotation Systems. (接收，JCR 一区，中科院 SCI 二区)

[6] 导师***, 作者***#, Xie, X, et al. VTracer: When Online Trajectory Compression Meets Edge Computing[J]. IEEE Transaction on Industrial Informatics. (大修，JCR 一区，中科院 SCI 一区)

[7] 作者***, (2018). Eco-Route: Recommending Economical Driving Routes For Plug-in Hybrid Electric Vehicles[J]. arXiv preprint arXiv:1806.09458, 2018.

[8] 导师***, 作者***#, Sui, G, et al. DVAT: An Error-bounded Trajectory Data Representation and Compression Framework[J]. IEEE Transaction on Knowledge Data Engineering. (提交，JCR 一区，中科院 SCI 二区)

B. 攻读硕士学位期间的主要科研项目

致　　谢