## 大型时空接触数据集中的高效可达性查询评估

## 摘要：

随着可靠的定位技术的出现和基于位置的服务的普及，现在可以通过移动对象（例如，个人，移动设备）准确地研究诸如传染病毒，敏感信息块和恶意软件之类的项目的传播。和车辆。在这样的应用场景中，当对象足够接近时（即，当它们是所谓的，接触时），项目在两个对象之间传递，因此一旦项目被启动，它就可以通过不断发展的网络穿透对象群体。对象之间的联系，称为联系网络。在本文中，我们首次在大型（即磁盘驻留）联系人数据集中定义和研究可达性查询，这些数据集记录了在延长的时间段内在空间环境中移动的（可能很大的）一组对象的移动。可达性查询通过由此类联系人数据集表示的不断发展的联系网络验证两个对象是否“可达”。我们提出了两个联系数据集索引，即使联系数据集的大小可能非常大，也可以对这些查询进行有效评估。利用称为ReachGrid的第一索引，在查询时，仅构建和遍历可达性评估所需的联系网络的一小部分必要部分。使用第二种方法，称为ReachGraph，我们预先计算不同规模的可达性，并在查询时利用这些预先计算进行有效的查询处理。我们优化磁盘上两个索引的放置，以便在查询处理期间实现有效的索引遍历。我们通过对真实数据和合成数据进行大量实验来研究我们提出的方法的优缺点。根据我们的实验结果，我们提出的方法在联系网络中的表现优于现有的可达性查询处理技术平均76％。

## 介绍

研究传染性病毒，想法和习惯，恶意软件和广播信息等项目如何通过移动物体（如个人，移动设备或车辆）传播，在包括公共卫生监测，社会在内的广泛应用中具有重要意义。 行为分析，计算机安全和智能交通监控，仅举几例。 在这样的应用场景中，一旦物体处于足够近的距离，即一旦它们被称为接触，物体就会在它们之间传递物品。 因此，一旦物品由物体发起，它就可以穿透称为接触网络的物体之间的不断发展的接触网络。 通过这种分析，人们可以设计公共卫生干预措施，以控制传染病的传播，或找到最初泄露敏感信息或引发恶意软件传播的来源。

可以说，在不断发展的联系网络中进行项目传播分析的主要构建块之一是能够计算可达性查询，该查询评估两个对象是否通过不断发展的联系网络“可达”。以前，缺乏捕获联系网络的准确数据集限制了联系网络中传播分析（尤其是可达性分析）的准确性和适用性，以前的研究不可避免地采用简化的联系网络模型，或者小规模和不准确的联系数据集。然而，随着最近在开发精确定位设备和基于位置的服务的普及方面的进步，可以在大规模和长时间内捕获对象的位置，从而产生非常大的联系数据集，捕获历史记录。物体接触准确，具有高时空分辨率。在本文中，我们专注于定义和有效评估大规模（磁盘驻留）历史联系数据集中的可达性查询，其中主要挑战是减少查询评估的计算时间。

考虑图1中描绘的联系网络，其显示在时间间隔T = [0,3]内的每个时间实例处的一组对象的位置。在该图中，如果两个对象接触，则它们通过链接连接;例如，o 1和o 2在时间0处接触。在[0,1]的时间间隔期间，对象o 4可从o 1到达。原因是如果一个项目在时间0由o 1发起，它可以在时间0从o 1传递到o 2，然后在时间1从o 2传递到o 4.注意在同一图中，o 1不是在[0,1]期间可从o 4到达。请考虑以下示例，了解可达性查询评估如何在上述某些应用场景的上下文中通过联系网络分析项目传播中发挥基础作用。在第一个例子中，假设一组个体O已知携带危险的传染性病毒。通过在O中的每个个体与其他人群之间执行一批可达性查询，可以通过确定在O中可以从O到达的个体集来识别可能在特定时间间隔内被直接或间接污染的个体。同一时间。注意，这种应用需要在成对的个体之间运行可能很多的可达性查询，这可能非常耗时。另一方面，及时治疗可以挽救生命中的大多数病毒性疾病。接下来，想象观察列表上的一组个体O，例如罪犯，并且需要被监视。执法机构可能需要发现那些可能与O中的任何个人有联系的人。同样，这需要执行一批可达性查询，以找到O中任何个人可以到达的所有人。这样的分析可能有助于防止新的犯罪并分析罪犯之间的关系。

近年来，人们广泛研究了图形可达性问题，该问题验证了图形的两个给定顶点之间是否存在路径[18,19]。 我们的问题与图形可达性的现有工作有两种不同。 首先，虽然之前关于图形可达性的工作假定为内存驻留，但我们关注的是非常大的磁盘驻留联系网络。 因此，我们研究如何索引磁盘上的联系人网络以实现高效的查询处理。 其次，我们的问题对象随着时间的推移在环境中移动时与时间和空间信息相关联。 我们表明，我们可以利用这些信息进行有效的可达性查询处理，而图形可达性的现有工作仅关注由抽象图建模而与空间和时间无关的数据集。

在本文中，我们提出了两种用于索引联系网络的索引结构，即ReachGrid和ReachGraph。如果我们仅考虑在给定查询时间间隔（查询间隔）期间发生的联系，则考虑可达性查询，该查询验证对象（查询源）是否可以通过联系网络到达另一个对象（查询目的地）。使用Reach Grid，我们的方法是通过从查询源开始扩展联系网络来即时计算可达性。然而，网络的扩张是非常昂贵的。相反，为了实现“引导”扩展，我们利用以下关于联系网络的简单而有力的观察;只有在相同的空间和时间位置发生的接触才与勘探相关，因此，通过相关的时空地点可以引导接触的探索，并且可以避免其他地方的性能提高。特别是，通过ReachGrid，我们提出了一个时空网格，将联系网络数据集中的所有联系人编入不同的时空地点。在查询时，该索引用于指导联系网络的动态扩展以验证可达性。

另一方面，使用ReachGraph，我们使用预先计算对象之间可达性的替代方法。 对查询源，目标和间隔的所有组合预先计算可达性是不切实际的。 因此，我们建议仅针对精心选择的查询源，目标和间隔组合预先计算可达性查询，并利用这些组合来动态计算所有其他组合的可达性。 反过来，在查询时，这允许递归地将给定的可达性查询分解为一组预先计算的可达性查询，以进行有效的查询处理。

最后，使用ReachGrid和ReachGraph，在磁盘上放置索引会显着影响查询处理的效率。 在随机磁盘块上放置索引（图形节点和网格单元）的简单方法会显着降低查询效率。 因此，在以下两个观察的指导下，我们为ReachGrid和ReachGraph开发了增强的磁盘放置方法。 首先，在查询处理期间按发生时间排序联系人。 其次，在索引遍历期间，如果o可以从o到达，则在o之后遍历对象o'。 我们分别在第4节和第5节中介绍了磁盘上的Reach Grid和Reach Graph的建议磁盘放置方法。

当ReachGrid通过沿空间和时间维度扫描联系人来评估可达性时，ReachGraph通过遍历连接图来计算可达性。 因此，当查询时间间隔较小时，可以预期ReachGrid与ReachGraph相当，反之亦然。 我们在第6节中的实证研究证实了这一预期。此外，我们提出的方法平均优于现有的可达性查询处理算法76％。

本文的其余部分安排如下。 相关工作在第2节中概述。在第3节中，我们分别定义了可达性查询联系网络。我们分别在第4节和第5节中介绍了ReachGrid和ReachGraph索引技术。 第6节介绍了我们的实验结果。 我们在第7节讨论了可达性问题的扩展。最后，我们总结了论文并讨论了第8节中可能的未来工作。

## 相关工作

我们将相关工作分为四类：图形可达性，轨迹索引和轨迹连接，外部图形遍历和图形索引，最后是联系网络分析。

## 2.1、图形可达性

给定有向图G中的两个顶点u和v，图形可达性验证是否存在从u到v的路径[19,18]。 虽然我们通过将联系网络转换为超图来减少我们的问题以图形可达性，但我们的问题与以前关于图形可达性的工作有几点不同。 首先，与以前关注内存驻留图的工作相比，我们考虑磁盘驻留图。 其次，我们关注“时空”图，并相应地利用这些图的空间和时间属性来增强索引构造和图遍历。 特别地，我们的图形顶点可以表示多个对象，而且对象可以与多个顶点相关联。 最后，我们提出的多分辨率图索引和双向图遍历方法是独特和新颖的，允许在现有技术可达性查询处理方法的效率方面进行前所未有的改进。

**2.2、轨迹加入和轨迹索引**

对移动对象数据管理的研究传统上主要集中在范围和最近邻居查询上。最近，还研究了轨迹连接[2,1]。在[1]中提出并研究了近距离接近（CPA）的问题。给定一组轨迹，CPA找到最近距离小于d的一对对象。虽然CPA问题与轨迹连接不同，但可以采用CPA问题的解决方案来解决轨迹连接问题。虽然我们在构建联系网络时使用轨迹连接算法，但我们的重点是索引联系网络以实现高效的可达性查询处理。轨迹处理相关工作的另一个相关主体是轨迹索引[5]，其侧重于有效处理距离查询及其变化的索引轨迹。相比之下，我们的问题是如何索引联系网络以实现高效的可达性查询处理，与范围查询及其变体相比，这种处理要复杂得多。

图[13,7]上的最短路径是另一个相关工作的主体。 给定图G =（V，E），假设每对图顶点之间的行进成本，最短路径找到最优最短路径。 相反，对于可达性查询，我们只对验证两个对象之间是否存在任何联系路径感兴趣。

**2.3、外部图形遍历和图形索引**

通过外部存储器图遍历[12,17]，研究扩展了经典的图遍历方法，如深度优先搜索（DFS）和广度优先搜索（BFS）。 如前所述，可以利用DFS和BFS来回答可达性查询。 但是，通过我们的工作，我们尝试通过设计有效的多分辨率索引结构和遍历方法来避免不必要的图节点扩展。

另一类工作侧重于索引时态图。 时间扩展网络（TEN）和时间聚合网络（TAN）[14]是表示时变网络的两种模型。 TEN通过在每个时间实例处实例化网络快照来表示时间依赖性。 TAN扩展了TEN，其中时变属性在边和顶点上进一步聚合。 我们利用TEN对接触网络进行初始建模，然后将其转换为更复杂的索引结构，如第5节所述。最近，[8]研究了由TEN表示的时空网络的有效索引。 然而，在本文中，重点是索引技术，以实现路由评估和检索查询的有效处理，而不是我们关注复杂可达性查询处理的工作。

**2.4、联系网络分析**

最近的研究[16,10]集中于分析接触网络的特征，例如两个物体之间的平均接触路径长度，或者最近研究了两个物体再次相互接触的持续时间。 这个工作领域与我们的工作正交，因为我们专注于索引联系网络以实现高效的可达性查询处理。

在缺乏连续网络连接的延迟容忍网络（DTN）中的路由是另一个相关工作[9]。 这项工作与我们的工作之间的区别有两个。 首先，在DTN中路由的目标是基于诸如消息传递比率之类的成本度量来找到从源节点到存在节点的最佳路径。 接下来，我们的可达性查询与时间间隔参数相关联，该时间间隔参数在索引构建和查询处理期间被利用以实现有效的可达性查询处理。

1. **问题定义**

在本节中，我们首先定义联系人网络，然后在联系人网络中形式化可达性查询。

**3.1、联系网络**

考虑在环境中移动的一组对象E.Wesay在两个对象oi，oj∈O之间发生接触c = {oi，oj}，当它们在足够近的距离内发送项目时，即，当它们的时候 距离小于阈值d T. d T的值取决于感兴趣的应用。 例如，对于通过人类群体的疾病传播，d T是米的数量级，而通过一组移动设备传输蓝牙数据，d T大约为几百米。 在c期间我们和接触对象，我们定义时间间隔T c，其中接触持续c的有效性间隔。

考虑一个时间间隔T，在此期间O中的对象在环境E中移动，并随着时间的推移进行各种接触。 每个对象的移动o∈O可以通过o的轨迹来建模，其在每个时刻t∈T捕获o的位置。我们将在时间间隔T期间O中的对象对之间的接触的集合称为接触网络。 在T期间O表示并用C表示。例如图1中，c 1 = {o 1，o 2}，c 2 = {o 2，o 4}，c 3 = {o 3，o 4}和c 4 = {o 1，o 2}是在T = [0,3]期间发生的接触，其有效间隔T c 1 = [0,0]，T c 2 = [1,1]，T c 3 = [1 ，2]和T c 4 = [2,3]。 请注意，我们区分c 1和c 4，尽管它们具有相同的接触对象，因为根据定义，有效间隔需要是连续的。

**3.2、可达性查询**

考虑联系网络C，其基于在时间间隔T期间在环境E中的对象O的移动历史构造。给定一对对象（oi，oj），oi，oj∈O和时间间隔T p。 ⊆T，可达性查询q在时间间隔T p期间验证是否存在来自oi的连接路径p ij。 直观地，两个对象o i和o j之间的接触路径由接触网络C中的一系列接触组成，通过该接触网络，任何虚拟物品i可以从网络行进到o i到o j。 我们将从对象oi到对象oj的接触路径定义为C中的一系列接触（c 1，c 2，...，cn），其中T ci与T p（1≤i≤n）重叠，并且对于每对 联系人ci和c i + 1（1≤i≤n - 1）我们有1）联系人共享一个对象，即，如果c 1 = {o 1，o 2}并且c 2 = {o 3，o 4 然后o 2 = o 3，和2）T ci在T c i + 1之前开始。

我们分别调用o i，o j和T p，查询源，查询目的地和查询间隔，并用q表示这样的查询：o t Tp❀oj。

1. **REACHGRID**

**空两段落**

受上述观察的启发，我们引入了一种有效的查询处理方法，该方法给出了可达性查询q试图仅构造处理q所必需的C'部分。 为此，首先在离线阶段我们构建一个称为ReachGrid的时空索引结构。 ReachGrid可以修剪与查询q无关的大多数联系人。 在在线处理阶段，我们在扫描查询间隔时，按照从查询源可访问的顺序递增地查找可从查询源到达的对象。 如果发现查询源可以从查询源访问，或者在查询间隔期间以及从查询源可访问的对象之间发生的所有联系都被处理，我们就会停止该过程。

**4.1、构建索引**

ReachGrid利用空间和时间上的对象的位置来避免遍历可访问性查询的不相关联系人。一旦在遍历按其出现时间排序的联系人时发现查询源和目标之间的联系路径，它就利用时间局部性来停止查询处理。为此，基于对象轨迹中的位置 - 矢量对的时间戳对对象轨迹段进行分组。当两个物体非常接近时，它们之间会发生接触。因此，基于空间局部性对对象进行分组倾向于聚合在一起并且在同一组中随时间接触的对象。这允许遍历组的子集，其仅包括在处理查询时可从查询源到达的对象。 ReachGrid通过在物体轨迹上施加两个网格来实现时间和空间局部性。第一个网格划分时间间隔T（T是C中所有接触发生的时间间隔）。第二个网格在T中的每个时间间隔内对轨迹段进行空间划分。

我们按如下方式构建ReachGrid。 首先，我们将时间间隔T划分为一组不相交的时间间隔，即T =（T 1，...，T n）。 接下来，我们基于局部性在每个T i期间对轨迹段进行空间划分，轨迹段在R（T i）中。 为此，对于每个时间间隔T i，我们在环境E上施加网格C i，其随后在R（T i）中划分轨迹段。 以这种方式，C i中的网格单元c包括跨越由c表示的区域的轨迹段。 注意，轨迹段r i（t i）∈R（T i）可以跨越C i的多个单元。 时间和空间网格的分辨率取决于输入联系网络和查询工作负载，我们在第6节中根据经验选择它们。

构造索引的示例如图2所示，其中T被划分为六个时间间隔。 此外，施加在环境上的4×4网格在T 0和T 1期间对轨迹段进行空间划分。 T 0和T 1分别具有三个和两个时间实例。 示出了前两个时间间隔的网格单元，即C 0和C 1中的网格，而其余部分未示出用于说明目的。 三个不同的物体在O中，随着时间的推移由圆形，正方形和三角形表示。

当通过探索空间网格单元中的轨迹段来进行查询处理时，我们建议将轨迹放置在盘上的连续块上的C i中的单元c中，以使得能够在查询处理期间有效地检索必要的轨迹段。 此外，c中轨迹段的位置矢量和时间戳对（ - →v，t）被放置在按时间戳排序的盘上。 这样，只要发现查询源和目标之间的联系路径，就可以避免处理c内的所有轨迹段。 因此，应该确定将单元放置在盘上的不同时间网格中，即C i中的单元相对于C j中的单元，其中i <j。 基于早期查询处理终止的相同目标，我们将单元放置在磁盘中C j中的单元之前的C i中。

**4.2、查询处理**

查询处理旨在通过扫描查询间隔来逐步查找可从查询源访问的对象。为此，在查询处理开始时，通过施加在前一部分中构造的时间网格将查询间隔分成时间间隔的子集，即，T p =（T j，...，T k） 。之后，定位包括查询间隔开始处的查询源的C j中的网格单元c，即​​，单元c包括查询间隔开始时的查询源位置。这可以在恒定数量的IO中执行，假设外部哈希表随着时间的推移将每个对象映射到其轨迹。假设我们在查询处理期间调用可从查询源访问的对象集，即种子集。最初，种子集仅包括查询源。为了处理可达性查询，该算法迭代T p中的每个T i并发现新的种子。为此，在T i的开始处定位包括当前种子的网格单元。随后，找到在T i期间可从至少一个种子到达的对象并将其添加到种子集中。请注意，只要发现可从查询源访问的新对象，就会将其添加到种子集中，因此该过程将继续更新的种子集。发现新种子的顺序基于它们可从任何当前种子到达的时间顺序。在一些情况下，T j可以是其起点与查询间隔起始点不同的间隔。在这些情况下，我们从查询间隔起始点开始处理T j。如果将查询目标添加到种子集或处理整个查询间隔，我们将停止查询处理。

查询处理的主要步骤是在每个T i期间发现新种子，j≤i≤k。假设在T i开始处的当前种子集是S i。目标是发现S i + 1，即在T i + 1的开始处的种子集，其与T i的结尾的种子集相同。假定其中S i中的种子所在的网格单元集合由C S i表示。我们首先发现在T i期间可能包含与种子接触的物体o的所有其他细胞。我们将这种细胞称为潜在的种子细胞，并用N i表示它们。通过在S i中创建对象的轨迹段的最小边界区域（MBR）并且因此找到并过滤与这些MBR在最大d T的距离处的单元，可以有效地找到N i内的单元。在查询处理期间，每当更新N i时，发现N i中的第一个对象o'，其不在S i中，但是可以从S i中的任何种子到达。直观地，我们从T i开头的种子集中的对象传播虚拟项目i，并找到接收i的第一个对象。这可以通过执行时间连接来完成，该时空连接通过在连接间隔期间的扫描时间来工作。因此，我们向S i添加o'并因此找到N i。假设在[t 1，t']期间发现o'从种子可到达（T i = [t 1，t 2]）。我们使用更新的集递归地继续该过程，但是在[t'，t 2期间]。请注意，在T i期间，检索的单元被缓冲以防止将来从磁盘中进行不必要的检索，并在T i结束时被丢弃。

图3中示出了用于在T i期间的查询处理的示例。在T i期间的时刻t 0，t 1和t 2，t 0 <t 1 <t 2处的对象o 1，o 2，o 3和o 4位置被突出显示。每个对象的轨迹由连接上述时间标记处的位置的链接示出。假设查询源和目的地分别为o 1和o 2，查询间隔为[t 0，t 2]。 o i的轨迹段周围的阴影区域表示具有d T宽度的轨迹段的MBR。该MBR表示o在种子集S i中，并且其轨迹在o的轨迹段的MBR内的任何其他对象将与o进行接触并被添加到S i。在t 0，S i包含o 1。在t 1，o 1和o 3形成接触，因此将o 3加到S i上。在[t 1，t 2]期间，o 1和o 3都在S i中。最后，在t 2处，分别将o 2和o 4所在的单元c 1和c 2添加到N i中，随后将o 2添加到S i中。因此，在[t 0，t 2]期间，可以从查询源访问查询目标。出于说明目的，我们仅讨论了在该示例中N i如何在t 2处改变。

整个在线处理步骤总结在算法1中。该算法获取查询源，目的地，间隔和在离线过程中构建的索引。 首先，将查询间隔量化为从T开始的时间间隔。然后，在第2-5行中执行初始化。 该算法在T p中迭代T i，并且对于每个T i，它在第9行中执行连接以找到在间隔w期间可从种子到达的第一对象。 R C S j（w）表示在w期间跨越C S j中的单元的对象轨迹段的集合。 我们采用[1]中的连接方法来扫描时间间隔w，并且每当发现一个新对象（而不是在种子集中并且从查询源可到达）时终止。 因此，在第10行更新集合。最后，当o j被添加到种子集或者处理T p中的所有间隔时，算法终止。

假设C j的每个单元平均包括n c个不同对象的轨迹，并且每个磁盘块平均包含C j的b c个单元。 最后，假设T'p = [t 1，t]⊆Tp = [t 1，t 2]是查询目的地可从源到达的最小时间间隔。 如果在T p期间无法从查询源访问查询目标，我们假设T'p = T p。 以下定理证明了ReachGrid查询处理和索引构造的复杂性。

T HEOREM 4.1。 可以使用O（| O || T |）IO构造ReachGrid。 查询处理的IO复杂度为O（| O || T'p | n c×b c）。 由于空间不足，我们跳过了证明的细节。

1. **REACHGRAPH**

在本节中，我们首先介绍ReachGraph索引构建步骤，然后讨论ReachGraph查询处理。

**5.1、构建索引**

为了构建给定联系网络C的ReachGraph，我们从C开始并对C应用一系列转换，最终将其转换为ReachGraph超图H N.转换分两个阶段进行，即还原阶段和增强阶段。首先，我们观察到在联系网络C中，可以识别节点的不相交子集，其中子集中的所有节点都等效地可达到或来自C中的任何其他节点v。因此，在还原阶段，我们预先计算这些子集。并将每个子集中的所有节点（以及它们的连接）减少到单个超节点。我们将得到的超图D N称为C大小的显着缩减版本。接下来，在增强阶段，为了进一步改进ReachGraph，我们以预定义的时间间隔预先计算D N中节点对之间的可达性。我们在几个时间分辨率下执行该预计算，并因此用额外链接的层次来增加D N以生成ReachGraph超图H N.利用H N，可以将可达性查询有效地分解为用于实时查询应答的一组预先计算的可达性查询

H N顶点的磁盘放置有两个原则，可以改善查询处理。 首先，有效放置应该放置在同一磁盘块上彼此可达的顶点。 以这种方式，在查询处理期间检索顶点时，还将读取和缓冲将在未来检索的一组顶点。 其次，在查询处理过程中如何遍历H N的顶点继承了一个顺序，当在磁盘上存储H N时应该利用该顺序。 通过H N顶点中的触点发生的时间顺序强制执行该排序。 我们解释了如何在磁盘上存储H N时考虑这两个原则，以实现高效的查询处理。

在本节的其余部分，我们首先将C模型呈现为所谓的时间扩展网络。 接下来，我们详细解释上述转换C Reduction - →D N和D N Augmentation - →H N. 最后，我们将讨论如何在磁盘上存储H N.

**5.1.1、联系网络模型**

我们用时间扩展网络（TEN）模型代表一个联系网络C [14]。 TEN通过在每个时间实例中包含单独的网络实例来捕获网络的时间依赖性。因此，在时刻t∈T的每个对象o i与单独的顶点o i（t）相关联。为了捕获接触，如果它们在时间t处接触，则在o i（t）和o j（t）之间引入双向边e =（o i（t），o j（t））。这样的边缘捕获了物品可以在t处从o i转移到o j的事实。注意，我们假设传输延迟可以忽略不计，因此，e是双向的。此外，在连续时间实例处在对应于相同对象的顶点之间引入边缘，即，在oi（t）和oi（t +）之间创建边e'=（oi（t），oi（t + 1））。 1）每次t。在这种情况下，e'是一个有向边，表示我可以在[t，t + 1]期间保持一个项目。我们在时间t定义所有顶点和边的图G t，即G t =（V，E），其中V = {o i（t）| oi⊆O}，作为C的t的快照。

图4（a）示出了对应于图1中的接触网络的示例C.对于图4（a）中的G 0，V = {o 1（0），o 2（0），o 3（0），o 4 （0）}和E =（o 1（0），o 2（0））。 当且仅当存在从o i（t 1）到o j（t 2）的路径时，很容易观察到在t p = [t 1，t 2]期间o j可从o i到达。 该路径表示在T p期间从o i到o j的接触路径。 例如，在图4（a）中，给定路径（o 1（0），o 2（0），o 2（1），o 4，在T p = [0,1]期间，o 4可从o 1到达（1））。

**5.1.2、转换联系网络**

**5.1.2.1、通过简化转换。**

在还原阶段，我们执行两个不同的步骤，将C转换为具有明显更小尺寸的超图D N. 减少C使得在查询处理期间遍历以查找可能的联系路径更有效。 请注意，这些缩减步骤是无损的，并保持查询处理的准确性。 我们首先陈述两个用于还原的属性。

PROPERTY 5.1。 [快照对称性]如果在时间实例t期间可以从o i到达o j，即查询间隔T p = [t，t]，则可以以相同的间隔从o j到达o i。

PROPERTY 5.2。 [传递性]假设在T p = [t 1，t 2]期间可以从o i到达o j，并且在T'p = [t'1，t'2]期间可以从o j到达o k。 如果t2≤t'2则在T''p = [t 1，t'2]期间可以从o i到达o k。

在缩减阶段的第一步，想法是在每个时刻t预先计算和实现对象之间的可达性。根据属性5.1和5.2，C的连通组件捕获在t处彼此可到达的对象集。例如，在图4（b）中，c 4 = {o 2（1），o 3（1），o 4（1）}，它捕获了所有对象o 2，o 3和o 4都可以从在时间实例t = 1时彼此相对。此外，如果在T p = [t，t']期间，来自连通分量c∈Gt的一个对象可以从连通的主管c'∈Gt'中的另一个对象到达，那么很容易从属性5.1和5.2中推导出来。在T p期间，c中的所有对象都可以从c'中的所有其他对象到达。因此，在缩小阶段的第一步，我们将C变换为图DN，其顶点是C的连通分量。为此，首先在每个Gt∈C中，我们将同一连通分量c中的所有顶点替换为由c表示的单个顶点。假设G t的连通分量的集合由C t表示。接下来，我们从每个c∈Ct到每个其他c'∈Ct+ 1创建一个边，如果在C中我们找到从c中的顶点到c'中的顶点的至少一个边。与C相比，这将C转换为有向无环图（DAG）D N，其具有明显更少数量的顶点和边缘，同时保持对象之间的可达性。对于D N，如果o j（t 2）的连通分量可以从o i（t 1）的连通分量到达，则在t p = [t 1，t 2]期间可以从o i到达o j。因此，为了回答可达性查询，我们需要在查询时找到给定o i（t 1）和o j（t 2）的o i（t 1）和o j（t 2）的相应连通分量。正如我们解释的那样，我们为每个时间实例t∈T生成并使用外部哈希表H t来定位对应于每个顶点o i（t）的连通分量。

缩减阶段的第二步随着时间的推移在连续的G t s中合并相同的连通分量。如果在时间间隔T'⊆T期间一组对象O'⊆O彼此可达（并且仅彼此相邻），则在D N中它们都属于T'期间相同连接分量的快照。因此，为了进一步减小D N的大小，我们可以在T'期间保留这种连接组件的一个副本，并在整个T'期间将其视为O'中对象的连接组件。例如，在图4（b）中，c 5和c 7是在T'= [3,4]期间相同连通分量的快照，并且可以合并。概括地说，假设一组连通分量ct∈Ct，c t +1∈Ct + 1，...，c t +n∈Ct + n都具有相同的成员O'，并且T'= [t ，t + n]。在这种情况下，我们删除ct，...，c t + n-1并通过加权边e将DN中的ct的父级（比如用d表示的G t-1中的连通分量）连接到c t + n （N）。我们将e（n）称为聚合边，其中权重捕获这样的事实：对于接下来的n个时间实例，d只能到达O'中的对象。图5显示了该还原步骤后图4（b）中的D N.除去c 5，c 4和c 3通过聚集边e（2）和e'（2）连接到c 7。这种减少可以显着缩小D N，特别是当物体位置的采样率与物体移动速度高度相关时。

**5.1.2.2、通过增加转换**。

为了找到两个连通分量ci∈Ct和cj∈Ct'之间的路径，我们可以简单地从c i开始扩展D N并检查我们是否能找到到达c j的路径。 尽管D N远小于C，但这种扩展仍然需要很长时间才能终止。 因此，我们建议预先计算D N的某些顶点之间的可达性，以便能够快速遍历D N.

特别是，我们建议在不同的预定义时间间隔内预先计算可达性。为此，我们将T分解成一组具有相等长度L的不相交间隔I 1，I 2，...，I n，并且对于每个I i = [ta，tb，C ta和C tb中的顶点之间的预计算可达性]。因此，如果存在从c到c'的长度L的路径，则利用从每个连通分量c∈Cta到每个其他连通分量c'∈Ctb的新有向边来增强D N.我们将这些边称为长边，并将它们加权L，这表示包含的时间实例的数量。得到的增广超图H N可以被认为是D N与由长边组成的新图的并集，每个长边都具有权重L.我们将后一图形接触网络称为第L个分辨率并且用D N L表示它。因此，D N可以被认为是第一分辨率或D N 1的接触网络。可以扩展这个想法并在其他时间间隔预先计算可达性以生成多分辨率图H N = DN∪DN L1∪DN L2∪...∪DN L n。但是，如果过度，这会显着增加边数，从而不利地降低查询扩展的效率。在第6节中，我们通过实验选择H N的最佳分辨率。图6描绘了D N 3，其中D N如图5所示。

**5.1.3、磁盘放置**

我们在H N的遍历中区分两种情况，H N是磁盘居住的超图。对于第一种情况，内部存储器可以保持c×| V（H N）|其中V（H N）是H N和c中的顶点集合的值是一个小常数（c≈12.375）[15]。在这种情况下，可以构造图的DFS树并将其保存在内部存储器中并在查询处理期间遍历它以验证可达性。对于第二种情况，上述关于顶点数量的假设不满足。对于这种情况，我们采用[12]中提出的外部BFS的思想，以便在查询处理期间有效地检索顶点。与[12]类似，我们对H N进行分区，并将顶点放在连续磁盘块上的同一分区中。然而，我们采用有向图的技术，因为H N是DAG。为此，我们首先按照拓扑顺序对H N的顶点进行排序，这与在可达性查询处理期间遍历H N的顺序相同。请注意，找到这样的顺序是微不足道的，因为在拓扑顺序上在T上创建H N个顶点（C i中的顶点在C i + 1之前生成）。然后，从每个顶点v，我们找到距离v最多dp的所有顶点U，即距v最多dp深度的顶点.U∪v中的顶点集可从v到达形成一个分区pv。我们称v是p v的根。如果u尚未分配给分区，我们迭代顶点并创建以顶点u为根的分区。请注意，在创建分区时仅考虑D N中的边，因此在分区过程中忽略长边以保留同一分区内图顶点的时间局部性。分区按生成顺序放在磁盘上。

我们运行示例的最终索引如图7所示，其中长边由e i（3），i = 1，...，4表示，聚合边由e（2）和e'（2）表示。超图H N和将对象与H N的分区相关联的哈希表位于盘上。每个哈希表H t定位包含o j（t）给定对象o j和时间实例t的分区。在该示例中，生成五个分区p 0，p 1，...，p 4，其中它们的连通分量成员是{c 0，c 3，c 4}，{c 1}，{c 2}，{c 6 ，c 8，c 9}和{c 7}。连接组件的成员放在H N的顶点内，我们将在下一节中讨论。尽管未在图中示出，但我们也将D N 1的反向图存储在盘上，即，如果e =（u，v）∈DN 1，则我们将e =（v，u）加到H N.这可以实现H N的有效双向遍历，我们将在下一节中讨论。最后，哈希表存储在主存储器中，以便能够快速查找给定t的H t，并随后查找H N的分区，其中包括磁盘上查询间隔开始（结束）的查询源（目标）。

**5.2、查询处理**

考虑可达性查询q：其中T p = [t 1，t 2]。为了处理q，可以首先在H N中找到顶点v 1和v 2，其分别表示在t 1和t 2处的o i和o j的连通分量。之后，从v 1开始，可以通过BFS或DFS技术遍历HN，以访问最多| t 2 -t 1 |从v 1开始的所有顶点.oj可以在T p期间从oi到达，如果且仅如果v 2是访问顶点之一。不幸的是，这种方法可能会在t 1«t 2时特别访问大量顶点。在本节中，我们提出了两个强大的思想，它们在H N遍历期间显着减少了访问顶点的数量。首先，我们利用多分辨率索引来遍历H N.因此，只要有可能，在遍历期间获取具有最大权重的长边（首先在较高分辨率上执行遍历）以实现H N的快速遍历。其次，在传递性属性5.2的推动下，我们从两个方向遍历H N，以更快地找到查询源和目标之间的可能联系路径。特别地，H N从查询源开始向前遍历并且并行地从查询目的地开始在D N的反向上向后遍历。遍历在两种情况下终止。或者，找到可从查询源到达并且可到达查询目的地的对象，或者在两个方向上遍历H N，直到双向遍历在查询间隔的中间停止。

对于存储器驻留图的遍历算法，外部图遍历算法也在文献中进行了研究[12,17]。我们分别用E-DFS和E-BFS表示外部BFS和DFS。尽管可以采用E-DFS和E-BFS来遍历H N，但我们采用E-BFS来实现H N的双向遍历。因此，我们的ReachGraph查询处理通过从v 1和v 2并行执行E-BFS来工作，其中来自v 2的搜索遍历D N 1的反向图。假设在前向遍历期间访问的顶点中的对象集合，即源自v 1的遍历，由O F表示。因此，我们在O B向后遍历期间访问的顶点中注释了一组对象。当OBSOF变为非空或者在[t 1，（t 1 + t 2）2]期间所有顶点都可从oi到达并且在[（t 1 + t 2）2，t 2期间可到达oj时，遍历终止）遍历。在第一种情况下，查询目的地可以从源到达，而对于后一种情况则不是这样。在遍历期间检索并缓冲分区以启用对一些未来顶点的内存中查找。当没有足够的空间用于新分区时，可以丢弃内存中较旧的分区。在向前遍历期间，如果顶点连接到长边，则遍历具有最大权重的边，并忽略其他边。我们认为这种方法是双向多分辨率BFS或BM-BFS。

BM-BFS技术的伪代码在算法2中给出。该算法首先在第2-3行中找到顶点v 1，v2∈HN.函数FindVertex（p，o，t）得到一个分区p，一个对象o和一个时间实例t，并返回包含o（t）的H N的顶点。然后，初始化两个队列用于第4行中输入图的前向和后向遍历.O F和O B也在第5行初始化。我们表示其实例包含在v中的对象集合。该算法通过运行ProcessQueue过程并行向前（第7行）和向后遍历（第8行），直到Q F和Q B都变空或验证可达性。使用ProcessQueue过程，在第2行中提取任一队列头部中的顶点v h。检查O v h中的每个对象以检查它是否已在反向遍历中被访问（第5-8行）。如果是这种情况，则报告查询目标从查询源可访问。之后，将v h的每个子v添加到遍历队列以启用下一步遍历。子（v，方向）过程返回源自v的最高分辨率的边，其端点是表示时间实例t∈[t 1，t 1 + t 2 2]和t∈[t 1 + t 2 2]的顶点， t 2]分别用于前向和后向遍历方向。以下证明了BM-BFS的正确性。

T HEOREM 5.3。 BM-BFS在查询间隔期间验证从查询源到目标的可达性。

证明。首先，假设H N仅包括一个分辨率，即H N = D N 1。 H N是DAG，其顶点在拓扑上排序并带有时间戳。前向遍历访问表示具有[t 1，t 1 + t 2 2]的有效区间子集并且可从查询源到达的联系的所有顶点。因此，后向遍历访问表示具有[t 1 + t 2 2，t 2]的有效性区间子集的联系的所有顶点。因此，如果存在从v 1到v 2的路径p，则在H N的前向和后向遍历之后发现p中的顶点。另外，p中的顶点是带时间戳的，因此，在H N的遍历期间保持p中的顶点的顺序。当我们在遍历期间考虑长边时，可能不会访问表示特定时间实例的H N的一些顶点。然而，图的一般连通性在所有分辨率下都被保留，因此通过采用长边，可以正确地验证查询。此外，基于敏感性5.2，提前终止条件准确地终止遍历。这样就完成了证明。

假设每个分区包括n个不同对象的实例，并且每个磁盘块平均保持b p个分区。 以下定理证明了ReachGraph查询处理和索引构造的复杂性（| T'p |在定理4.1中定义）。

定理5.4。 ReachGraph索引可以用O（| O || T |）IO构造。 查询处理IO复杂度为O（| O ||| T'p | n p×b p）。

由于空间不足，我们跳过了证明的细节。

GRAIL [18]是内存驻留图最有效的图形可达性方法之一。 它基于图顶点的随机区间标记的思想。 表1比较了在磁盘主体D N上采用ReachGrid和ReachGraph与GRAIL的索引构造和查询时间复杂度以处理可达性查询。 我们的方法明显优于GRAIL，因为有效的磁盘放置和提前终止查询（| T'p |≤| T p |）。 对于GRAIL，d是一个小常量，它是分配给每个图顶点的间隔数。 n r是在每个时刻t∈T从任何对象可到达的平均对象数o∈O。