TimeReach：关于演化图的历史可达性查询

## 摘要：

由于大多数图表随着时间的推移而发展，因此能够查询其历史记录非常有用。 我们考虑在过去的某个时间间隔内要求存在路径的历史可达性查询，要么在间隔的整个持续时间内（合取查询），要么在间隔中的至少一个时刻（析取查询）。 我们研究了存储演化图的完全传递闭包和执行在线遍历的两种选择。 然后，我们提出了一个适当的可达性索引，称为TimeReach索引，它利用了大多数真实世界图包含大型强连通组件的事实。 最后，我们针对不同的图表大小，历史查询类型和时间粒度提供了对所有方法的实验评估。

## 介绍

近年来，越来越多的图形结构化数据可从各种来源获得，例如社交，引用，计算机和超链接网络。 随着时间的推移，几乎所有这样的真实网络都会随着节点和边缘的添加或删除而发展。 对其演变的分析发现了大量应用，从社交网络营销到病毒传播和数字取证。

在本文中，我们假设在不同的时刻给出了一组与图形状态相对应的不断发展的图形快照。 我们解决了有效回答涉及此类快照的查询的问题。 特别是，我们关注的是基本查询类型，即可达性查询，它询问节点u是否可以在过去的特定时间间隔内从另一个节点v到达。 我们将此类查询称为历史可达性查询。

尽管如此，人们对处理图形数据有相当大的兴趣，通过各种图形查询（包括可达性，距离和基于模式的查询），查询图形历史的研究要少得多。 构建用于处理历史图形查询的索引的另外两种方法我们都知道考虑历史最短路径查询[9,2]。 具体而言，[9]的作者提出了一种基于排序与最短路径计算相关的节点或边缘的方法，而[2]中提出的动态索引构造不支持节点或边缘删除。

**关于历史查询的所有其他工作主要集中在有效存储和检索处理每个查询所需的图快照[14,13,21,17]。 特别是在[14]中，探索了图形增量和选定的物化快照的组合，而在[13]中，重点是存储，共享和处理增量。 在[21]中，对时间上紧密的快照进行聚类，选择每个聚类的一个代表并用于查询的初始评估。 最后，在[17]中，研究了快照在分布式设置中的放置和复制。 相反，在本文中，我们解决了构建索引以回答历史可达性查询的问题。**

已经广泛研究了静态图上的可达性查询。 该领域的研究遵循两个一般方向，即通过有效存储传递闭包和加速在线遍历。 关于传递闭包，已经提出了各种方法，包括链方法[10,5]，探索生成树的方法，比特矢量压缩[26]和区间[1,28,12]和跳[7,22,6]标签。 在在线遍历的情况下，通常使用区间标记[4,25,30]来修剪搜索空间。 在进化图[1,3,23,31]的情况下，还有一些关于递增地维护可达性索引的工作，但是，可达性仍然考虑单个快照，即图的当前版本。

在本文中，我们探索了图快照的紧凑表示，称为版本图，其中每个节点和边都用一组时间间隔进行注释，在此期间相应的节点和边缘存在于演化图中。 我们将此类设置称为寿命，并通过使用非重叠和非连续间隔来寻求其最小表示。 我们还介绍了一组有效操作路径寿命的基本操作。

为了处理历史可达性查询，我们首先重新审视基本的传递闭包和在线遍历方法。 对于传递闭包，我们计算每对节点的可达性信息的最小表示。 对于在线遍历，我们提出了一种新的基于区间的版本图遍历以及许多修剪步骤。 此外，为了避免与预先计算传递闭包相关的成本和空间开销以及提高在线遍历的处理成本，我们提出了一种称为TimeReach的新方法。

TimeReach利用了大多数图形由强连通组件（SCC）组成的事实[20,15]。 因此，我们不是维护节点对的可达性信息，而是维护发布列表，其中包含有关SCC中节点成员资格的信息。 我们通过为SCC分配适当的标识符来最小化发布列表的大小。 我们表明，SCC最佳分配标识符的问题等同于随后的图快照中SCC之间的最大二分匹配问题。 除了发布之外，我们还保留了一个浓缩版图，它对应于SCC演变的版本图。 为了提高回答历史查询的性能，我们还在浓缩版图上引入了基于修剪的地标标记[2,29]的interval-2hop方法。

我们用三个真实的社交网络数据集对我们的方法进行了广泛的评估。 我们的实验结果表明，TimeReach是节省空间的，特别是对于由大型SCC组成的图，就像社交网络一样。 它的增量结构很快; 索引新的快照图只需几秒钟。 最后，使用TimeReach处理历史查询比版本图的在线遍历快几个数量级。

本文的其余部分结构如下。 在第2节中，我们介绍了相关的工作，而在第3节中，我们正式定义了历史可达性查询。 在第4节中，我们介绍了生命周期的版本图和操作，并介绍了两种基线方法，即传递闭包和在线遍历。 在第5节中，我们介绍了TimeReach 索引方法，而在第6节中，我们介绍了实验结果。 第7节总结了论文。

## 相关工作

尽管图形数据管理一直是当前许多研究的焦点，但处理历史查询的工作相当有限。研究进化图的主要重点是有效地存储和检索图快照。在本文中，我们将重点放在处理查询的索引上。为此，我们假设以版本图形式的图快照序列的紧凑表示。或者，可以在序列中仅存储图快照的一些子集以及适当的增量，使得可以通过在所选快照上应用增量来重建任何其他快照[14,13]。已经提出了用于减少存储和快照重构开销的各种优化，诸如增量的分层索引和用于快照的重叠存储的存储器池[13]。还提出了对时间上紧密的快照进行聚类并计算每个聚类的代表[21]，为每个聚类存储来自代表的Deltas以实现高压缩。在G \*图数据库中，通过利用它们之间的共性来有效地存储快照[16]。每个节点的不同版本仅存储一次，而不管它所属的快照数量，并由紧凑的内存中索引编制索引。对于负载平衡和可用性，快照数据在许多工作者之间复制。

这些方法中的历史查询处理需要作为重建相关快照的第一个且成本高昂的步骤。 然后，通过对每个查询的在线遍历处理查询。 通过最小化所应用的增量数量[14,13]，避免重建所有快照[21]，或通过并行查询执行和正确的快照放置来尽量减少需要重建的快照数量，从而解决查询性能问题。 分配[17]。 在这项工作中，我们解决了一个不同的问题，即历史可达性查询的索引。

历史最短路径距离查询在[9]中得到了解决。 作者提出了一种基于排序与最短路径计算相关的节点或边缘的方法。 最后，[2]最近的工作还提出了一种历史距离查询的动态索引方案。 但是，作者只考虑插入。 这个假设简化了问题，因为两个可到达的节点仍然可以到达。 作者提出了一种动态2hop索引构造，它不适用于节点或边缘删除的情况。

静态图上的可达性查询已经从两个方面进行了彻底调查：传递闭包压缩和改进在线搜索。

传递闭包压缩。相关研究旨在通过为每个节点仅存储它可以到达的节点的子集来压缩传递闭包。第一个想法是分解k个节点不相交链中的图，并且每个节点只存储它可以在每个链中到达的第一个节点[10,5]。另一行研究提取了图的生成树，并使用它来压缩传递闭包。树的每个节点用整数间隔标记，使得如果节点u是v的祖先，则u的间隔包含v的间隔。通过树边缘的可达性可以通过标签包含检查容易地确定。为了通过非边缘边缘合并可达性，每个节点在图形[1]中继承其后继的间隔，或者构造非树边缘的部分传递闭包[28]。在区间标记的基础上，在[12]中使用了一个树，其顶点是从原始图中提取的成对不相交路径。压缩传递闭包的另一种方法是2跳标记[7,22,6]。每个节点存储两组中间节点：一组可以到达的节点Lout和一组可以到达它的节点Lin。节点u只有在（v）6 =∅中L out（u）∩L时才能到达节点v

加速在线遍历。 这些方法使用区间标记来通过修剪搜索空间来辅助在线遍历。 在[4]和[25]中，构建了图形的树覆盖，然后，对于树标记无法回答的查询，使用标记来执行非树边缘的在线搜索以指导 搜索。 在[30]中，多个区间用于标记。 如果标签包含检查未产生否定答案，则使用修剪搜索的间隔在线遍历图表。

一些工作讨论了在演化图[1,3,23,31]的情况下索引的增量维护。 但是，更新的索引仅包含有关图形当前版本的可访问性信息，不能用于回答历史查询。

所提出的方法与我们的方法正交，因为它们可以被调整以使得它们可以用于加速或避免浓缩图的在线遍历。 我们通过调整其中之一，即2hop标签来证明这一点。

1. **问题定义**

随着新节点或边缘的添加，或者删除现有节点或边缘，大多数真实世界图形会随着时间的推移而发展。 我们假设时间是离散的，并使用连续的整数来表示连续的时间瞬间。 时间瞬间有两种直观的解释。 一种解释是实际时间，例如时刻t可以对应于2014年10月20日，PDT上午5:00。 另一种观点是可行的。 在这种情况下，每次发生图形操作，更新或删除时，时间都会提前。 对时间的两种解释都与我们的表述一致。

设G =（V，E）为有向图，其中V是节点集，E是边集。 我们使用G t =（V t，E t）来表示时刻t的图快照，即在时刻t存在的节点和边的集合。

定义1（演化图）。 时间间隔[t i，t j]中的演化图G [t i，t j]是图快照的序列{G t i，G t i + 1，...，G t j}。

图1（a）中示出了一个示例，其描绘了由四个图快照{G t 0，G t 1，G t 2，G t 3}组成的演化图G [t 0，t 3]。 为简洁起见，我们将时刻t i + 1表示为t i + 1，并且当从上下文中清楚地表达含义时，可互换地使用t i和i。

我们使用术语时间粒度来指代创建新时刻和相应图快照的频率。 在实际时间的情况下，粒度可以例如从几毫秒到几年，而在操作时间的情况下，粒度可以在一个或多个操作的水平。 细粒度的时间粒度需要维护大量的历史信息，但支持精确的历史查询。

给定静态有向图G =（V，E）和两个节点u，v∈V，可达性查询询问在G中是否存在从u到v的路径。对于演化图，我们引入以下两种类型的历史可达性查询。

定义2（历史可达性查询）。 令G [ti，tj] = {G ti，G ti +1，... G tj}，是演化图，IQ = [tk，tl]⊆[ti，tj]时间间隔和v，ua对 节点：

（i）联合历史可达性查询uIQ∧; v返回true，如果在所有图形快照中存在从u到v的路径G t m，tk≤tm≤tl G [t i，t j]。

（ii）析取历史可达性查询u IQ∨; 如果在G [t i，t j]的至少一个图快照G t m，tk≤tm≤t1中存在从u到v的路径，则v返回true。

我们的目标是推导出有效回答可达性查询的方法。 一个简单的解决方案是为每个图快照构建一个不同的索引，然后在每个图快照中构建一个可达性查询。 但是，这种解决方案会产生很大的空间开销。 此外，它需要额外的处理来组合每个查询的结果。 相反，我们建议为间隔建立索引。

1. **版本图**

在本节中，我们将介绍版本图，这是一个演化图的自然具体表示。 首先，让我们来定义生命的概念。 对于节点u（或边e），其寿命表示在进化图中存在u（相应e）的时间间隔集。 更正式地说，给定一个演化图GI = {G ti，G ti +1，...，G tj}，节点u的寿命L（u），（分别为L（e））（相应边缘） e）是一组间隔，使得区间[ti，tj]⊆I属于L（u），（分别为L（e）），当且仅当对于所有ti≤tm≤tj，u∈V tm（相应的e∈Etm）。

我们将生命跨度建模为一组时间间隔，以捕获图形演化的一般情况，其中节点和边缘可以被删除，然后在后续快照中重新插入。 时间间隔集也称为时间元素[11]。 如果我们不允许重新插入已删除的节点或边缘，则生命跨度只是间隔。 此外，如果没有删除，则所有寿命都是[t i，t curr]形式的间隔，其中t i是节点或边缘首次出现的时刻，t curr是当前快照的时刻。 因此，在这种情况下，寿命可以简单地用时刻t i表示。 在下文中，我们使用I来表示时间间隔，我使用I来表示时间间隔的集合。 为了表示演化图G I，我们使用版本图V G I。 版本图是标记的有向图，它以简洁的方式捕获图的演变。

定义3（版本图）。 给定演化图GI = {G ti，G ti +1，...，G tj}，其版本图是边和节点标记的，有向图VGI =（VI，EI，L u，L e）其中： VI = Stm∈IVtm，EI = Stm∈IEtm，L u：VI→I在VI中为每个节点u分配其寿命L u（u）和L e：EI→I指向EI中的每个边e 寿命L e（e）。

图1（b）中显示了一个示例，其中描绘了图1（a）中演化图的版本图。

**4.1寿命操作**

让我们定义一些关于寿命的操作，即一组间隔。 对于两组I和I 0的时间间隔，我们说I覆盖I 0，表示I w I 0，如果对于I 0的区间I 0中的每个时刻t，则I中的区间I使得t 属于I.我们也使用I w I I和I wt时间t。 我们说两组I和I 0的时间间隔相等，I≈I0，如果我是0和I 0 w I.

我们希望在等效的间隔集中保持最小。 我们称这样的集最小集。 我们首先定义一些时间间隔的简单属性。 当I∩I 0 =∅并且否则重叠时，两个时间间隔I = [t i，t j]和I 0 = [t 0 i，t 0 j]被称为不相交。 当t 0 i = t j + 1时它们被称为连续，否则称为非连续。 很容易看出以下命题成立。

命题1。

（i）一组间隔是最小的，当且仅当它由不相交和不连续的间隔组成时。

（ii）对于每组时间间隔，存在唯一的等效最小间隔集。

接下来我们在区间集上定义两个有用的操作，即连接和合并。 给定两组间隔，join返回两者共有的时刻，而merge返回其中至少一个存在的时刻。

定义4（间隔集的连接和合并）。 设I = {I 1，... I k}和I 0 = {I 0 1，... I 0 l}是两组时间间隔。

（i）加入I⊗I 0和I 0是相当于{I1∩I0 1，... I1∩I0 l，...，Ik∩I0 1，...的最小集合。 我∩我0 l}。

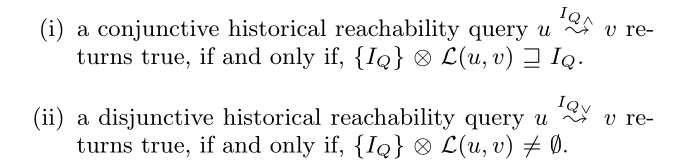
（ii）合并I⊕I 0和I 0是相当于I∪I 0的最小值。

注意，如果I和I 0是最小的，那么集合{I1∩I0 1，... I1∩I0 l ...，Ik∩I0 1}是最小集合，而集合{ 我1∪I 0 1，...我1∪I 0 l，...，我k∪I 0 1 ...我k∪I 0 l}可能不是最小的。

路径p的寿命L（p）包括其所有边缘共存的时间间隔。 显然，对于路径p = e 1 ... em，它认为L（p）= L e（e 1）⊗...⊗L e（em），其中L e（ei），1≤i≤ m，是ei的生命。 例如，对于图1（b）中的路径p =（（u 4，u 3），（u 3，u 7），（u 7，u 6）），L（p）= {[2,3] } {{1,3]}⊗{[0,1]，[3,3]} = {[3,3]}，而路径p 0 =（（u 1，u 3），（u 3） ，u 7），（u 7，u 4）），L（p 0）= {[0,1]}⊗{[1,3]}⊗{[0,0]，[2,3]} =∅。

我们现在可以定义两个节点u和v之间的可达性的寿命L（u，v）。设P（u，v）= {p 1，... pl}是从u到u的所有路径的集合 v.L（u，v）取决于VGI中从u到v的所有可能路径的寿命，特别是L（u，v）= L（p 1）⊕...⊕L（pl）。 例如，对于图1（b）中的节点u 4和u 6，P（u 4，u 6）= {p 1，p 2，p 3，p 4，p 5，p 6}其中p 1 = u 4 u 3 u 6，p 2 = u 4 u 3 u 7 u 6，p 3 = u 4 u 1 u 3 u 6，p 4 = u 4 u 1 u 3 u 7 u 6，p 5 = u 4 u 1 u 2 u 3 u 6，p 6 = u 4 u 1 u 2 u 3 u 7 u 6（注意，为了符号简洁，路径由参与节点而不是边缘表示）。 然后，L（u 4，u 6）= {[2,3]}⊕{[3,3]}⊕{[0,1]}⊕{[1,1]}⊕{[1,1]} ⊕{[1,1]} = {[0,3]}。

显然，历史可达性查询可以用寿命表示。 具体地，给定版本图V G I，时间间隔I Q = [t k，t l]⊆[t i，t j]和两个节点v，u，



为了表示lifespans，我们使用位数组。 假设不失一般性，最大时刻即图快照的数量是T.然后，寿命，即一组间隔，I由大小为T的位数组B表示，使得B 如果我是，则[i] = 1，否则为0。 例如，取I = {[2,4]，[9,10]，[13，curr]}和T = 16.I的位数组表示为00111000011001111.这导致连接⊗和 合并⊕。 特别地，让I和I 0是两组间隔，B和B 0是它们的位阵列。 然后，I⊗I 0被计算为B逻辑AND B 0并且I⊕I 0被计算为B logical-OR B 0。 另一种表示方式是使用有序的间隔列表。 然后使用合并排序的变化执行寿命操作，导致O（T）复杂度。 列表在计算可达性时通常会产生大量计算开销。

**4.2基线方法**

在静态图上回答可达性查询有两种基线方法，即图传递闭包的预计算和图的在线遍历。 在本节中，我们将重新审视版本图上历史可达性查询的这些基线方法。

**4.2.1历史传递闭包**

我们不是为演化图G I的每个图快照保持不同的传递闭包，而是为版本图V G I维护单个传递闭包CL I。 传递闭包包括每对节点u，v，它们的可达性寿命L（u，v）。 为了构造传递闭包，我们使用Floyd Warshall算法的变体，其考虑了寿命，如算法1所示。如果存在路径pu，则从节点u到节点w和路径pw，v从节点w到 节点v然后存在路径pu，v =（pu，w，pw，v）从u到v与L（pu，v）= L（pu，w）⊗L（pw，v）和L（pu， v）与目前计算的L（u，v）合并。

在最坏的情况下，算法1的时间复杂度为O（| V I | 3 T），并且需要以| V I |的顺序存储。 2。 为了回答可达性查询，我问你∨; v或你我Q∧; v，最初定位CL I中的条目L（u，v），然后与查询间隔I Q连接，因此需要恒定的时间复杂度。

**4.2.2版本图的在线遍历**

处理区间I Q的可达性查询的直接方法是在所有图快照G t，t∈IQ上执行在线遍历。 当使用版本图表示时，这对应于仅遍历边e，使得L e（e）w t，对于每个t∈IQ一次。 我们称这种方法为基于即时的遍历。

为了避免多次遍历，即I Q中的每个快照一次，我们考虑基于区间的遍历版本图。 解析历史查询的基于BFS的间隔遍历在算法2中示出，并且在算法3中示出了联合历史查询。

特别是，对于联合查询，由于节点v可以通过不同图形快照的不同路径从u到达，我们维持一个区间集R，其中L（u，v）⊗IQ部分到目前为止（第9行，算法）3）。 当R覆盖整个查询时间间隔I Q时，遍历结束（第10行，算法3）。

为了加速遍历，我们执行了一些修剪测试。 当我们到达寿命超出查询间隔的节点时，遍历停止。 此外，当{IQ}⊗Le（n，w）6 =∅时，遍历在节点n的邻居w处停止，因为节点v不能通过在至少一个t内不存在的边缘到达。 查询间隔（第6行，算法2和3）。

如果它参与从源到目标的多个路径，则可以多次遍历边缘。 为了减少这种遍历的次数，我们通过为每个节点w记录一个间隔设置IN（w）以及已经遍历的查询间隔的部分来提供额外的修剪。 如果查询到达w再次查找间隔I0⊆IQ和IN（w）w I 0，则遍历被修剪（算法2的第11行，算法3的第15行）。

例如，考虑图1（b）和中的版本图

查询u 1 [0,3]∧; 你5。 路径p 1 = u 1 u 3 u 6 u 5，p 2 = u 1 u 3 u 7 u 6 u 5，p 3 = u 1 u 2 u 3 u 6 u 5，p 4 = u 1 u 2 u 3 u 7 u 6 u 5，p 5 = u 1 u 4 u 3 u 6 u 5和p 6 = u 1 u 4 u 3 u 7 u 6 u 5 with L（p 1）= {[0,1]} ，L（p 2）= {[1,1]}，L（p 3）= {[1,1]}，L（p 4）= {[1,1]}，L（p 5）= { 需要遍历[2,3]}和L（p 6）= {[3,3]}以正确推断出查询结果为真。 因此，对于不同的时间间隔I 0 i I I Q，需要多次遍历一些边缘，例如（u 3，u 6），（u 6，u 5）。 然而，当查询通过路径p 3再次到达u 3时，它被修剪并且它不会遍历边缘（u 3，u 6），因为IN（u 3）等于{[0,1]}，它涵盖了 当前查询间隔I 0 = {[1,1]}。

由于在基于即时和基于间隔的遍历的最坏情况下，每个边缘可以遍历| I Q | 两次遍历的复杂度为O（（| V I | + | E I |）| I Q |）。 然而，实际上，基于间隔的遍历优于基于时间的遍历，因为每个边缘遍历覆盖查询间隔的大部分而不是单个时刻。 此外，修剪可确保在相同间隔内不会遍历两次边。

1. **TIMEREACH索引**

我们的方法利用了这样一个事实，即许多真实世界的社交图以大型强连通组件（SCC）为特征[20,15]。 因此，我们维护有关每个节点所属的SCC的信息，而不是维护节点对的可达性信息。 如果两个节点属于同一组件，则它们是可访问的。 但是，随着图表随着时间的推移而发展，其强关联组件也会发生变化。 图1（c）中显示了一个示例，其描述了图1（b）中的图表随着时间的推移而演变的SCC。

给定一个演化图GI = {G ti，G ti +1，...，G tj}，我们在每个图快照Gtk∈GG中调用一个算法，例如Tarjan的算法[24]，以识别相应的一组强连通分量。 在每个快照上为每个SCC分配唯一的ID。

对于每个节点u，我们维护一个列表P（u），其包含（C，t）对，指定节点u在时刻t所属的强连接组件C. P（u）被称为发布列表，列表中的每对都是发布。 存储复杂度为Ω（| V I || I |），因为每个节点在每个时刻最多参与一个SCC。 如果我们使用Tarjan的算法[24]，构造列表的时间复杂度是O（（| VI | + | EI |）| I |），因为Tarjan算法的每次运行都有一个O（| VI | + | EI） |）复杂性。

为了清楚起见，我们假设单个节点形成单独的SCC，其id是相应节点的ID。 但是，为了提高空间效率，我们不会在这种情况下保留帖子。

我们执行额外的优化。 许多节点具有强连接，即，即使面对组件拆分和连接，它们仍保持在相同的组件中。 我们利用这一事实来减少发布所需的存储空间，方法是观察这些节点的发布列表由相同的元素组成。 我们通过仅存储此类列表一次并使用指向公共列表的指针替换相关节点的发布列表来避免冗余。 我们将此方法称为发布共享。

图2（a）中示出了一个示例，其中，例如，第一个发布列表指示具有ID 1到50的节点属于强连接组件，其在时间t 0具有id C 1，在t 1处具有C 6并且 C 9 at t 2。

另外，对于每个图快照G tk，我们构造SCC图快照GS tk =（VS tk，ES tk），使得在G tk中每个SCC在VS tk中存在节点U，并且存在边（U ，V）在ES tk中，当且仅当，在G tk中存在从属于对应于U的SCC的节点u到属于对应于V的SCC的节点v的边（u，v）。 对于时间间隔I = [t i，t j]，这导致演进的SCC图G S I = {G S t i，G S t i +1，...，G S t j}。 随着SCC的创建，我们逐步构建SCC图。 每个SCC图的大小取决于原始快照图的大小，在最坏的情况下等于它。

我们称这种方法为简单的TimeReach（TR）。 为了回答可达性查询，我问∧; v，（或者，我是Q∨; v），我们检查每个t∈IQ是否u和v属于同一个组件。 如果不是这种情况，我们遍历相应的G S t。 接下来，我们提出了一种更节省空间的方法来利用强连接组件进行历史查询。

**5.1简明TimeReach**

在TR方法中，我们每时刻维护信息，我们希望汇总此类信息以在时间间隔内表达SCC参与。 在这种情况下，如果你在I 0的所有时刻参与具有id C的SCC，则发布（C，I 0），I0⊆I，属于P（u）。 我们的目标是尽量减少此类帖子的总数。

问题1（最佳SCC-id分配）。 给定时间间隔I和每个t∈I的一组SCC，找到SCC的id分配，这导致最小的过帐数量。

每次节点参与不同的SCC时，都会创建一个新的发布。 因此，应重新分配SCC ID，以便最大限度地减少此类新发布的数量。 我们使用加权图来形式化SCC的id的最佳分配。

特别地，我们使用加权图GC（VC，EC，W）在时间间隔I上对SCC演化进行建模，其中每个节点U∈VC对应于在某个时刻t∈I存在的SCC，并且边e =（ U，V）∈EC，当且仅当SCC U在时间tk存在时，SCC V在时间tk + 1存在并且存在至少一个属于U和V的节点。 W向每个边e =（U，V）分配权重W（e），其对应于属于U和V的节点的数量。

加权图的一个例子如图2（b）所示，它描绘了图的演变，其发布列表如图2（a）所示。 例如，在时刻t 1创建的组件C 7由来自组件C 4的100个节点和来自C 5的150个节点组成。

设GC [tk，tk +1]（VC [tk，tk + 1]，EC [tk，tk + 1]，W）为GC（VC，EC，w）的子图，由节点U∈组成 VC [tk，tk +1]对应于以时间间隔[tk，tk + 1]存在的SCC。 G C [t k，t k +1]表示SCC演变中的一个步骤。 注意，根据G C的定义，G C [t k，t k +1]是二分图。

我们做了以下观察。 在时刻t k + 1，精确地为在t k + 1而不是在t k参与不同SCC的那些节点创建新的发布。 这些新发布的数量等于G C [t k，t k +1]中从节点U到V的权重之和，其中U具有与V不同的id。 因此，为了最小化新发布的数量，我们必须最大化具有相同id的节点对之间的边缘的权重。 这对应于找到G C [t k，t k +1]的最大二分匹配。

定理1.最优SCC-id分配问题可以减少到找到每个G C [t k，t k +1]的最大权重二分匹配（MWM）M k的问题。

证明。 如上所示，求解每个二分图G C [t k，t k +1]的MWM最小化了在t k + 1处创建的新发布的数量。我们将证明这种逐步分配在G C中是最优的。 出于矛盾的目的，假设最优分配是边N的集合N，EC，并且N不同于通过最大二分匹配获得的边集，即Pe∈Nw（e ）> P k Pe∈Mkw（e）。 因此，对于一些m，对于N m =N∩EC[tm，tm +1]，它认为Pe∈Nmw（e）> Pe∈Mmw（e），这意味着M m不是MWM 这是一个矛盾。

图2（c）显示了通过二分匹配分配新ID之后的加权图，而图2（d）显示了新的发布列表。

图2：（a）共享发布列表，（b）SCC演变的加权图，（c）二分匹配后的加权图，以及（d）压缩共享发布列表

最大权重二分匹配问题得到了很好的研究（例如，参见[8]的调查）。 在图G（V，E）上解决这个问题最广泛使用的算法是匈牙利算法，其运行时间范围从O（| V | 3）到O（| E || V | + | V | 2 loglog | V |）取决于实现。 另一类算法取决于边缘权重，最快的算法在O（| E | p | V | logW）时间内运行，其中W是最大边缘权重。 另外，已经提出了许多快速近似算法。 最简单的算法是贪婪算法，它按重量对边缘进行排序，并重复选择具有最大权重的边缘。 该算法可以用O（| E |）时间复杂度实现，并产生1/2最差情况近似。

用于构造SCC过帐的增量算法在算法4中给出。它将当前快照和计算到前一个快照的过帐作为输入，并构造当前过帐。它首先使用具有复杂度O（| V t | + | E t |）的Tarjan算法计算SCC（第2行）。然后，它构造具有复杂度O（| E C [t-1，t] |）的图G C [t，t + 1]（第5行）。接下来，计算MWM并将新id分配给新SCC（第6-9行）。此步骤的复杂性取决于使用哪种算法来计算MWM。我们使用具有复杂度O（| E S [t-1，t] |）的贪婪算法。最后，为当前快照的每个节点创建/更新SCC发布，仅为参与与时刻t-1中的SCC（具有不同id）的不同SCC的节点创建新条目（第11-22行） ）。这些步骤的复杂性为O（| V t |），因为循环中的每个操作都具有恒定的时间复杂度。因此，总计算法的运行时间为O（| V t | + | E t |）。

与简单的TR方法一样，我们还构建了演化的SCC图，在这种情况下，由于通过二分匹配实现的强连接组件数量的减少，节点数量更少。

最后，我们构造了演化SCC图的版本图V G S I =（V S I，E S I，L u，L e），我们称之为压缩版图。 我们按如下方式逐步构建压缩版本图。 对于每个快照G ti∈GI，对于每个边（u，v）∈Et i，我们查找条目（U，I 0），（V，I 00）s.t的发布P（u），P（v）。 ti∈I0和ti∈I00。 如果U 6 = V并且边缘（U，V）6∈ES I，则边缘被添加寿命{[t i，t i]}，否则边缘的寿命被扩展为包括t i。 我们将上述方法称为精简TimeReach（TRC）。

**5.2查询处理**

查询处理（析取或连接）可达性查询u I Q; v分两步进行。 在第一步中，检索节点u和v的适当发布。 如果两个节点在整个查询间隔期间属于同一个强连接组件以进行联合查询，或者一次属于析取查询，则答案为真。 否则，让I 0 Q为节点u和v属于不同组件的间隔集。 该查询被重写为一组U k I Q i形式的可达性子查询; V m，其中u属于SCC U k，v属于SCC V m，对于某些公共时间间隔IQ i，I 0 Q w IQ i，集合IQ = Si IQ i由不相交的间隔组成，并且IQ≈I0 Q。 子查询的结果被组合以通过AND（OR）为连接（析取）查询产生查询的答案。

例如，考虑查询u [1,15]∧; v在图3中，u和v的发布列表分别为P（u）=（C 6 [4,7]，C 5 [8,11]，C 4 [11，curr]和P（v） =（C 6 [1,8]，C 4 [11,15]）。查询分为三个子查询：u IQ1∧; C 6，u IQ2∧; C 6，v IQ3∧; C 5。

在最坏的情况下，两个节点在IQ的每个时刻都属于不同的SCC，因此我们需要遍历每个t的压缩版本图，成本为O（| IQ |（| VSI | + | ESI |） 影响性能的两个因素是每个节点的过帐数量和压缩版本图的大小。 发布次数越少，第二步中所需的子查询越少。 压缩版图越小，遍历越快。 因此，SCC ID的最佳分配对于查询处理性能至关重要，因为它使发布列表保持较短并且压缩版本图的大小较小。

**5.3区间2Hop**

通过维护其他信息，可以提高版本图表的可访问性。 在本文中，我们使用基于修剪的地标2hop标记的方法[2,29]。 这个想法是，对于给定图的每个节点u，我们在（u）和L out（u）中维护两个标签L，其包括可以到达u并且可以分别到达u的节点。 计算标签使得节点u达到v，如果只有，则在（v）∩Lout（u）6 =∅。 现在可以使用标签来回答可访问性查询，而不是遍历图形。

对于历史可达性查询，我们还保持L中的每个节点w（v）可达性寿命L（w，v）以及L out（u）中的每个节点w可达性寿命L（u，w）。 在存在2hop标签的情况下，回答一个问题，我是Q∧; v（uIQ∨; v），我们计算（v）∩L out（u）中的集合L，然后对于（v）∩L out中的每个w，我们在L中加入w的生命周期（v ）L out（u）中w的寿命。 为了回答该问题，将（v）∩Lout中的L中的节点w的连接寿命L（w）与查询间隔L连接以查看它们是否覆盖I Q（或者至少具有共同的时刻）。

我们以递增方式计算压缩版图的节点的标签。 对于间隔I = [t i，t j]，我们从t i开始计算I中每个时间t的SCC图快照的标签。 对于每个时间tk，tk> ti，我们合并在时间tk为节点C计算的标签，以及在前一时间tk-1为C计算的标签。用于构造每个SCC图快照的L in和L out 在时刻tk，我们使用INOUT策略处理图的节点，该策略从具有最大（indegree（u）+1）×（outdegree（u）+1）的节点开始BFS遍历[29]。 图4中给出了版本图中每个SCC节点的最终2hop标签的示例。

1. **实验**
2. **结论**

大多数现实生活中的图表随着时间而演变。 在本文中，我们解决了在这些图上有效回答历史可达性查询的问题。 这样的查询询问在过去的时间间隔期间是否可以从另一节点v到达节点u。 我们已经提出了一种称为TimeReach的方法，该方法利用了大多数图由强连接组件（SCC）组成的事实。 TimeReach维护有关每个节点的SCC成员资格的信息，以及表示强连接组件之间链接的图表。 我们还维护了一个浓缩版图，它对应于SCC演变的版本图。 我们对三个真实社交网络数据集的广泛实验表明，TimeReach具有存储效率，并且可以以较小的开销逐步构建。 即使涉及大的时间间隔，也可以有效地处理历史查询。

未来的工作有很多可能的方向。 其中一个方向是利用TimeReach来回答其他类型的历史查询，例如最短路径查询。 另一个方向涉及TimeReach的分布。 分布可以基于时间或通过将属于相同SCC的节点放在一起来利用SCC演进。