武漢大學

本科毕业论文（设计）

一种直径限制组斯坦纳树算法的设计与实现

姓　　名：　　杨　　宇　　轩

学　　号：　　2020302192031

专　　业：　　计算机科学与技术

学　　院：　　计 算 机 学 院

指导教师：　　董　　文　　永

二〇二四年五月

原创性声明

本人郑重声明：所呈交的论文（设计），是本人在指导教师的指导下，严格按照学校和学院有关规定完成的。除文中已经标明引用的内容外，本论文（设计）不包含任何其他个人或集体已发表及撰写的研究成果。对本论文（设计）做出贡献的个人和集体，均已在文中以明确方式标明。本人承诺在论文（设计）工作过程中没有伪造数据等行为。若在本论文（设计）中有侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

作者签名：杨宇轩　　指导教师签名：

日　　期：　2024　年　5　月　14　日

版权使用授权书

本人完全了解武汉大学有权保留并向有关部门或机构送交本论文（设计）的复印件和电子版，允许本论文（设计）被查阅和借阅。本人授权武汉大学将本论文的全部或部分内容编入有关数据进行检索和传播，可以采用影印、缩印或扫描等复制手段保存和汇编本论文（设计）。

作者签名：　杨宇轩　　指导教师签名：

日　　期：　2024　年　5　月　14　日

摘　　要

知识图谱是一种用于表示知识的图形化结构，它以图的形式呈现实体之间的关系，呈现了丰富的语义信息，通常用于语义网、人工智能和自然语言处理领域。为了探索大型复杂知识图谱，目前的方法通常使用组斯坦纳树进行关键词搜索。然而，这种搜索方式存在一些问题，如难以覆盖全部关键词、子图内聚性不足等。因此，对组斯坦纳树进行直径限制，允许其连接不完整的查询关键子集，可以更好地进行关键词搜索。

相较于传统的组斯坦纳树问题，有直径限制的组斯坦纳树问题仍然缺乏有效的算法实现。针对这一问题，本文提出了 PrunedCBA+ 算法，有效地改进了 Cba 算法的时空复杂度，同时保持了其近似比。具体来说，本文采用了对查询结构的动态聚合、使用哈希表优化空间复杂度、大量采取最优性剪枝减小常数等方法。对若干真实图和合成图的实验结果表明，PrunedCBA+ 算法具有高效性。

关键词：组斯坦纳树；近似算法；知识图谱

ABSTRACT

Knowledge graphs are graphical structures used to represent knowledge, depicting relationships between entities in a graph format, showcasing rich semantic information. They are commonly employed in the fields of the Semantic Web, Artificial Intelligence, and Natural Language Processing. Currently, methods for exploring large and complex knowledge graphs often rely on using Steiner trees for keyword searches. However, this search method has some drawbacks, such as difficulty in covering all keywords and insufficient cohesion within subgraphs. Therefore, imposing a diameter constraint on Steiner trees, allowing them to connect incomplete sets of query keywords, can enhance keyword searches.

Compared to traditional Steiner tree problems, the Steiner tree problem with a diameter constraint still lacks effective algorithmic implementations. Addressing this issue, this paper proposes the PrunedCBA+ algorithm, which effectively improves the time and space complexity of the Cba algorithm while maintaining its approximation ratio. Specifically, this paper employs dynamic aggregation of query structures, space optimization using hash tables, and extensive adoption of optimal pruning to reduce constants. Experimental results on several real and synthetic graphs demonstrate the efficiency of the PrunedCBA+ algorithm.

Key words: Group Steiner Tree; approximation algorithm; knowledge graph

目　　录

1 绪论

1.1 研究背景

随着知识图谱在语义网、人工智能和自然语言处理等领域的广泛应用，对于如何高效地探索和利用大型复杂知识图谱的需求日益增加。知识图谱作为一种图形化结构，以实体之间的关系为基础，呈现了丰富的语义信息。在实际应用中，对知识图谱进行关键词搜索是一项重要的任务，它可以帮助用户快速准确地获取所需信息。

然而，传统的关键词搜索方法在面对大规模复杂知识图谱时往往效率不高，难以覆盖全部关键词，覆盖成本也较高。为了解决这些问题，研究者们提出了使用组斯坦纳树进行关键词搜索的方法。相较于普通的关键词覆盖需要完整覆盖一组特定的顶点组，组斯坦纳树考虑图中多个不同的关键词顶点组，并尝试找到与所有组有交的最小子图，允许资源共享和路径多样性，它被广泛应用于网络设计、通信和路径规划等领域。然而，即使在使用组斯坦纳树进行关键词搜索时，仍然存在着一些挑战，例如存在着难以处理大规模图谱、子图片段内聚性不足等问题。

在这样的背景下，有直径限制的组斯坦纳树问题应运而生。该问题通过限制子图的直径，允许连接不完整的查询关键子集，从而更好地解决了关键词搜索中的一些挑战，相较于组斯坦纳树更加符合潜在用户的需求。然而，目前针对有直径限制的组斯坦纳树问题的有效算法实现仍然相对缺乏。因此，本研究旨在提出一种有效的算法，以改进现有方法并提高关键词搜索在大型复杂知识图谱中的效率和准确性。

1.2 研究现状

本文的研究内容基于传统的组合优化问题——斯坦纳树覆盖，因此将从以下两个方面讨论本文研究内容的研究现状。

1.2.1 有直径限制的组斯坦纳树相关问题

有直径限制的组斯坦纳树问题起源于算法 Core 的提出[1]，这个算法提出并解决了寻找一个知识图谱中覆盖最多查询关键子集的，顶点间最远跳数在一定范围

内的最小可行子图的问题，但是其并没有明确的指出算法距离最优可行子图的近似比，时间效率上也不够高。而基于证书节点的算法 CBA[2] 明确的给出了此问题的近似比，但是其时间效率上依然存在优化空间。而本文的具体研究内容即为对算法 CBA 时间效率上的改进。

对于图中求取斯坦纳子树的相关问题中，加以对直径进行限制是一类比较常见的思路[3–5]。例如，卡普尔和萨尔瓦特提出了动态调整直径的具有双标准近似比的算法[6]，能够对求解可行子图的近似比进行动态调整；马什雷吉和扎雷伊提出了一种基于直径的参数化近似算法[7]，解决了双边权图中的组斯坦纳树问题。

1.2.2 基于斯坦纳树的问题

有直径限制的组斯坦纳树问题基于一般的斯坦纳树问题，其是一个 NP 难的组合优化问题。而近似解决一般图斯坦纳树问题的方法首次在二十世纪八十年代被提出[8]，在了几十年的发展中，贝尔卡等人提出了基于迭代随机舍入的近似算法[9]，改进了斯坦纳树的近似比；程龚等人提出了基于动态标号的查询距离方法，极大的改善了斯坦纳树的求解效率[10]。相应的，斯坦纳树也具有一些基于动态规划的指数级的确定性算法[11, 12]。对于斯坦纳树的一些特殊性研究，孙亚辉等人提出了具有 $g-1$ 近似比的同时具有点权和边权的斯坦纳树问题的解决方案[13]；陈宗建等人也提出了斯坦纳树问题的一个更广义版本的算法[14]，其中权值在顶点对上进行了调整。与这些基于斯坦纳树的问题相比，有直径限制的组斯坦纳树问题进一步对可行答案提出了两个限制条件，即直径和覆盖组数这两个约束条件。现有的斯坦纳树问题算法无法满足这两个条件。

1.3 本文的研究内容

本文的研究内容主要针对于在不劣化算法近似比的同时对前人算法时空复杂度的改进。

主要有以下三点：

- 基于哈希表和动态聚合对图中两点间最短路信息结构有跳数限制的地标标签算法（HBLL）的查询优化。
- 基于最优性剪枝原理，极大的减少了原算法证书节点的查询范围。
- 在算法中动态固化查询信息，减少信息结构的查询调用次数。

1.4 本文的组织结构

本文旨在研究优化有直径限制的组斯坦纳树问题。为了系统地探讨这一问题，本文共分为五章，各章节内容安排如下：

第一章，首先介绍了有直径限制的组斯坦纳树问题的研究背景及其在实际应用中的意义。随后，对当前相关研究现状进行了概述，并明确了本文的研究内容和组织结构。

第二章，详细介绍了问题的背景和相关的符号约定。此外，还介绍了本文研究所采用的两点间最短路信息结构有跳数限制的地标标签算法（HBLL）。

第三章，着重阐述了算法的具体实现方式以及算法近似比的相关证明。同时，通过模块化的方式详细介绍了算法的改进细节和原理，为读者提供了深入了解的视角。

第四章，详细描述了实验的设置和结果。进一步通过消融实验对所提出的方法的有效性进行了分析，为读者提供了对算法性能和有效性的全面评估。

第五章，总结了本文的研究内容，并展望了未来可能的研究方向。通过全面概括，读者可以更好地理解本文的贡献和价值，并对未来的研究方向有所启示。

2 问题描述与基础

2.1 准备工作

知识图谱。知识图谱以实体间的关系为基础，被抽象化为一组有连边的关系图。在本文中，它可以被认为是一个有向图 $G = \langle V, E \rangle$，其中 V 代表实体抽象成的节点集合，$E \subset V \times V$ 表示实体间关系抽象成的边集。尽管关系构成了有向关系，但对于图中的路径，本文不关注其中边的方向是否一致，因此，在本文中，图谱可以被视为无向图。

表 2.1 记号约束

$\mathsf{wt}\,(e)$	边 e 的权
$\mathsf{ud}\,(u,v)$	u 与 v 之间的无权距离（跳数）
$\mathsf{wd}\,(u,v,h)$	u 与 v 之间的无权距离不大于 h 的路径中带权距离最小值
$\mathsf{ul}\,(P)$	无权路径 P
$\mathsf{wl}\,(P)$	带权路径 P
$\mathsf{N}\,(v)$	与顶点 v 在图中直接连接的顶点集合
$\mathsf{ud}_{\,T}(u,v)$	在树 T 上计算的无权距离 $\mathsf{ud}\,(u,v)$
$\mathsf{wt}\,(T)$	树 T 的边权和
$\mathsf{diam}\,(T)$	树 T 的直径
$\mathsf{cov}\,(T)$	树 T 的覆盖的查询集合数量

记号约束。表 2.1 记录了图中的约定记号。具体的，边 $e = (u,v) \in E$ 会随着实体间关系的重要程度抽象出权值 $\mathsf{wt}\,(e) \in \mathbb{R}$；$\mathsf{ud}\,(u,v)$ 表示连接点 u, v 之间最少需要的边数；$\mathsf{wd}\,(u,v,h)$ 表示带权边边权和最小的一条边数不超过 h 的路径边权和；$\mathsf{N}\,(v)$ 表示集合 $\{w|(v,w) \in E\}$；$P = \{v_1, v_2, \ldots, v_{|P|}\}$ 表示一条无权或带权路径；$T = \langle V_T, E_T \rangle$ 表示图中的一颗生成树，其被定义为一个边数最小的连通子图；$\mathsf{ud}_{\,T}(u,v)$ 被定义为特殊子图 T 中连接点 u, v 的最少边数；$\mathsf{wt}\,(T)$ 被定义为特殊子图 T 的边权和；$\mathsf{diam}\,(T)$ 被定义为树的直径，表示 $\max_{(u,v)\in V_T \times V_T} \mathsf{ud}_{\,T}(u,v)$。

关键词查询。一次关键词查询可以被抽象为 $\mathbb{K} = \{K_1, K_2, \ldots, K_g\}$，其中 $K_i \subseteq V$。关键词查询 $\mathbb{K}$ 的答案是图 G 的生成树 T。本文定义生成树 T 覆盖的集合数量为 $\mathsf{cov}\,(T) = |\{K \in \mathbb{K} : K \cap V_T \neq \varnothing\}|$

2.2 问题描述

有直径限制的组覆盖斯坦纳树问题（DCGST）的目标是找到图中的一颗生成树，使得它覆盖尽可能多查询集合的同时，满足生成树的直径限制，同时尽可能使树的边权小。形式化的，令 $G = \langle V, E \rangle$ 表示图，$\mathbb{K} = \{K_1, K_2, \ldots, K_g\}$ 表示一次查询，D 表示生成树的直径限制，那么问题的目标是寻找一颗依次如下描述的生成树：

- 生成树的直径不大于 D：

$$T \in T_1 = \{T : \mathsf{diam}(T) \leq D\}$$

- 生成树覆盖的集合最多：

$$T \in T_2 = \{T \in T_1 : \mathsf{cov}(T) = \max_{T' \in T_1} \mathsf{cov}(T')\}$$

- 生成树的边权最小：

$$T \in T_3 = \{T \in T_2 : \mathsf{wt}(T) = \min_{T' \in T_2} \mathsf{wt}(T')\}$$

2.3 问题工具

在与图论有关算法中，两个节点之间、节点与点集之间的最短路径查询经常被使用。虽然许多传统算法诸如非负权图上的单源最短路径算法 Dijkstra，任意图上的单源最短路算法 Bellmon-Ford, 全局多源最短路算法 Floyd-Warshell 都很好的解决了这个问题，但实际应用中的图节点数量的数量级比较大，硬件内存无法支持对任意两点间的最短路径算法进行存储。

因此，研究者们想出了许多保留更少信息以满足最短路查询的相关结构，其中一种最常见的做法是基于将一条最短路划分为两段，被称为 2-hop 覆盖类型[15]。中心标签（Hub-Labeling）和地标标签（Landmark Labeling）是其中的典型例子。

由于本文需要在满足条数限制内寻找图中的最短路径，因此对一种剪枝地标标签算法（Pruned Landmark Labeling）[16] 进行了改进，以适应算法的需要，此算法被命名为有跳数限制的地标标签算法（Hop-Bounded Landmark Labeling）。

2.3.1 有跳数限制的地标标签算法的构建

有跳数限制的地标标签算法（HBLL）的基本思路与普通的地标标签算法类似，对于图 $G = \langle V, E \rangle$ 中的每个节点 v，其都拥有一个三元组地标集合，记为 $L(v)$。对于每个属于 $L(v)$ 的三元组 $\langle l, h, d \rangle$，都代表一条跳数不大于 h 端点为 v 与 l 的路径，其拥有跳数范围内的距离 d，即 $\mathsf{wd}(v, l, h) = d$。而对于整个图来说，这些地标标签在算法中具有两个重要的性质保证他们的正确性与有效性：

- （完整性）任取满足 $\mathsf{ud}(u, v) \leq h$ 的 h 与 $u, v \in V$，$L(u)$ 与 $L(v)$ 存在一个共同的地标 l，满足 l 在跳数不超过 h 的 u, v 之间的最短路径上，即：

$$\langle l, h_u, d_u \rangle \in L(u), \langle l, h_v, d_v \rangle \in L(v) : h_u + h_v \leq h \text{ and } d_u + d_v = \mathsf{wd}(u, v, h) \tag{2.1}$$

- （简洁性）对于每个节点的标签三元组，不存在一个三元组较于其他三元组，同时拥有更大的跳数和更大的距离，即：

$$\forall \langle l, h_i, d_i \rangle, \langle l, h_j, d_j \rangle \in L(v) : h_j > h_i \rightarrow d_j < d_i \tag{2.2}$$

Construction of HBLL。算法 1 详细描述了有跳数限制的地标标签算法（HBLL）构建方法，具体的内容由剪枝地标标签算法（Pruned Landmark Labeling）[16] 改进而来。其基本的思想是依次枚举每个节点向外扩张得到当前节点作为地标的结果，直到这个地标作为中间点的结果可以被之前的地标所覆盖，它的本质是每个点为起点的最短路有向无环图的并构成的结果。其中调用的函数 GetWD(v, u, h) based on L_{i-1} 表示基于前 $i-1$ 个点扩展结果获得的地标标签集合中，能够得到的节点 v 与 u 之间在 h 跳数内的最小距离。

在实际存储中，每个节点的 $L(v)$ 可以按照简洁性依次排布在链表中，按照多元组大小顺序进行排布，即若三元组 $\langle l_i, h_i, d_i \rangle$ 若排在三元组 $\langle l_j, h_j, d_j \rangle$ 之前，当且仅当 $l_i < l_j$ 或 $l_i = l_j$ and $h_i < h_j$ 或 $l_i = l_j$ and $h_i = h_j$ and $d_i < d_j$。

2.3.2 有跳数限制的地标标签算法的方法与实现

根据记号约束和算法需要，有跳数限制的地标标签算法（HBLL）需要提供三个方法。为了方便分析，本文假设对于某个图，其顶点 $|L(v)|$ 的平均值为 $|L|$，其大小属于 $O(n)$ 数量级，但实际上要小得多。

GetUD。根据 HBLL 的性质，方法 GetUD(u, v) 表示在 HBLL 所表示的图中，

Input: Graph $G = \langle V, E \rangle$

Output: HBLL

for $u \in V$ **do**
  $L_0(u) \leftarrow \varnothing$;
end

for $i \leftarrow 1$ ***to*** $|V|$ **do**
  $L_i \leftarrow L_{i-1}$;
  $V_0 \leftarrow \{v_i\}$;
  for $j \leftarrow 1$ ***to*** $|V|$ **do**
    $d_0[v_j] \leftarrow 0$ **if** $j = i$ **else** ∞;
  end
  for $h \leftarrow 0$ ***to*** ∞ **do**
    if $V_h = \varnothing$ **then**
      break the inner for loop;
    end
    $V_{h+1} \leftarrow \varnothing$;
    $d_{h+1} \leftarrow d_h$;
    for $u \in V_h$ **do**
      if $d_h[u] \geq$ GETWD(v_i, u, h) *based on* L_{i-1} **then**
        continue the for loop;
      end
      $L_i(u) \leftarrow L_i(u) \cup \{\langle v_i, h, d_h[u] \rangle\}$;
      for $u' \in N(u)$ **do**
        if $d_h[u] + \mathrm{wt}\,(u, u') < d_{h+1}[u']$ **then**
          $V_{h+1} \leftarrow V_{h+1} \cup \{u'\}$;
          $d_{h+1}[u'] \leftarrow d_h[u] + \mathrm{wt}\,(u, u')$;
        end
      end
    end
  end
end

return $L_{|V|}$;

算法 1　CONSTRUCTION OF HBLL

节点 u 和节点 v 的最短跳数，其实现如下：

$$\textsc{GetUD}(u,v) = \min_{\substack{\langle l,h_u,d_u\rangle \in L(u) \\ \langle l,h_v,d_v\rangle \in L(v)}} h_u + h_v \tag{2.3}$$

在实际的方法实现中，由于地标标签的存储结构有序，可以采取二路归并维护双指针的方法。因此方法单次调用的平均复杂度为 $O(|L|)$。

GetWD。根据 HBLL 的性质，方法 $\textsc{GetWD}(u,v,h)$ 表示在 HBLL 所表示的图中，节点 u 和节点 v 的在跳数不大于 h 的路径中，具有的最小距离，其实现如下：

$$\textsc{GetWD}(u,v) = \min_{\substack{\langle l,h_u,d_u\rangle \in L(u) \\ \langle l,h_v,d_v\rangle \in L(v) \\ h_u+h_v \le h}} d_u + d_v \tag{2.4}$$

在实际的方法实现中，同样可以采取二路归并维护双指针的方法，只是层次上复杂一层，这里不过多赘述，单次调用的平均复杂度不变，仍为 $O(|L|)$。

GetMWP。方法 $\textsc{GetMWP}(u,v,h)$ 返回 $\textsc{GetWD}(u,v,h)$ 中最短路径的具体内容，即一列路径的节点与节点之间的连边。完成这个方法需要额外对每个三元组存储前置三元组位置，并递归的依次寻找，直到两侧节点均达到 l 处，由于路径长度不大于 h，因此这个方法的时间复杂度为 $O(h)$。

3 基于证书节点算法的具体改进

为了简化问题，本文假设带有限制的直径 D 均为偶数，为奇数的情况可以比较简单的扩展为偶数的情况，具体可以参考论文[1]。

3.1 基于证书节点的原始算法介绍

3.1.1 基本思想

引理 3.1.1：对于一颗直径不超过 D 的树 T，一定存在节点 $c \in V_T$，满足 $\forall v \in V_T$，s.t. $\mathsf{ud}_T(c, v) \leq \frac{D}{2}$。

证明：反证。对直径恰好为 D 的树，选取直径中点 v，其到直径两端距离长度为 $\frac{D}{2}$，假设其不满足上述限制，那么存在一点 w，使得 $\mathsf{ud}_T(v, v) > \frac{D}{2}$，则选取这条链作为直径的一半会使直径扩大，与直径定义矛盾。对于直径小于 D 的情况，证明类似。 □

考虑到DCGST问题首先需要满足的限制为直径 D 的所有生成树，由引理3.1.1可得，对于图中所有满足直径限制的生成树，其一定存在点 c 和若干条以 c 为起点，长度不超过 $\frac{D}{2}$ 的路径的并构成。

因此，证书节点算法（*Certifcate-Based Algorithm*）的基本思想分为两步，第一步从原图中寻找证书节点和以证书节点为起点的若干条路径，第二步将它们合并为一颗树并输出。

3.1.2 具体内容

Input: Graph G, Query $\mathbb{K}$, diameter bound D
Output: feasible answer Tree T
$\langle M, c \rangle \leftarrow$ PrunedOptMC(M, c);
$T \leftarrow empty$;
for *each* $v \in M$ **do**
  $T \leftarrow$ Merge(T, GetMWP($c, v, \frac{D}{2}$));
end
return T;

算法 2 CBA

Cba 总览。对于单次查询的证书节点算法 2，它调用了两个子模块 PrunedOptMC 与 Merge，将于后文逐一介绍。在算法中，本文首先调用 PrunedOptMC 获取证书节点 c 和一个终端节点多重集合 M。其中多重集合 M 与询问集合的某个子集 $\hat{\mathbb{K}} \subset \mathbb{K}$ 对应，其每个元素都属于某个查询元素 K_i，因此有 $|M| = \mathrm{cov}(T)$，这将得到一个可行的解。接着，将初始为空的树 T 和 c 与 M 中每个节点构成的跳数不大于 $\frac{D}{2}$ 的最短路径进行合并，得到最终的可行解树输出。

PrunedOptMC。原始 Cba 算法使用普通广度优先搜索从查询集合进行扩展，这里不过多介绍，改进的剪枝证书节点算法使用优先队列维护可能的证书节点，并辅以相应的剪枝。

形式化的，算法 3 模块 PrunedOptMC 首先初始化 $checked$ 数组表示图中每个节点是否被尝试扩展为证书节点过，hop 表示图中每个节点距离最近的查询集合的跳数。接着，算法将查询集合中的所有点插入大根优先队列并尝试进行扩展。优先队列每次取出其中 $\mathrm{pri}(v,h)$ 最大的节点 v，其距离最近的查询集合的跳数为 h。在进行可能的剪枝后，算法调用 MaxM 模块计算以某个节点 v 为证书节点能够获得的终端可重集 M_v，并最终选择一个最优的二元组 $\langle |M_v|, \sum_{u \in M_v} \mathrm{wd}(v,u,\frac{D}{2}) \rangle$。具体的，二元组 $\langle x_1, y_1 \rangle$ 比 $\langle x_2, y_2 \rangle$ 更优当且仅当 $x_1 > x_2$ 或 $x_1 = x_2$ AND $y_1 < y_2$，即终端可重集的大小尽可能大的同时，证书节点距离每个终端节点在 $\frac{D}{2}$ 跳数内的距离之和尽可能小。

PrunedOptMC 使用了以下三类剪枝方法：

1. 使用优先队列扩展，其中 pri(v,h) 定义为：

$$\mathrm{pri}(v,h) = |\{K \in \mathbb{K} : u \in K, h + \mathrm{ud}(v,u) \le D\}| \tag{3.1}$$

2. 基于跳数的剪枝

   具体的，对于每个节点，仅在它距离查询节点集合更近时，才被允许进入有限队列待更新，且最大的跳数距离不大于 $\frac{D}{2}$。正确性比较显然，这里不过多赘述。

3. 基于权值剪枝

   具体的，若当前证书节点对应的终端可重集大小与最优相等时，以当前终端节点到终端集最近的节点距离乘最优可重集大小进行二元组第二项的估计，即 $|M| \cdot \min_{u \in \bigcup_{k \in \mathbb{K}} K} \mathrm{wd}(v,u,\frac{D}{2})$，易得此估计比真实值更小，因此具有正确性。

Input: Graph $G = \langle V, E \rangle$, Query $\mathbb{K}$, diameter bound D

Output: multiset of terminals M and certificate c

$checked \leftarrow$ an array of length $|V|$ with initial 0;

$hop \leftarrow$ an array of length $|V|$ with initial ∞;

$Q \leftarrow$ an empty priority queue;

for $u \in \bigcup_{K \in \mathbb{K}} K$ **do**
    ENQUEUE($Q, \langle u, 0, \text{pri}(u, 0) \rangle$);
    $hop[u] \leftarrow 0$;
end

$\langle M, c \rangle \leftarrow \langle \emptyset, \text{null} \rangle$;

while Q *is not empty* **do**
    $\langle v, h, \_ \rangle \leftarrow$ DEQUEUE(Q);
    if pri(v,h) $< |M|$ **then**
        break the while loop;
    end
    if pri(v,h) $= |M|$ **then**
        if $|M| \cdot \min_{u \in \bigcup_{k \in \mathbb{K}} K} \text{wd}(v, u, \frac{D}{2}) \geq \sum_{u \in M} \text{wd}(c, u, \frac{D}{2})$ **then**
            continue the while loop;
        end
    end
    if *checked[v]=false* **then**
        $M_v \leftarrow$ MAXM($G, \mathbb{K}, D, v$);
        $checked[v] \leftarrow$ true;
        if ($|M_v| > |M|$) *or* ($|M_v| = |M|$ *and*
          $\sum_{u \in M_v} \text{wd}(v, u, \frac{D}{2}) < \sum_{u \in M} \text{wd}(c, u, \frac{D}{2})$) **then**
            $\langle M, c \rangle \leftarrow \langle M_v, v \rangle$;
        end
    end
    if $h < \frac{D}{2}$ **then**
        for $v' \in N(v)$ **do**
            if $h + 1 < hop[v']$ **then**
                ENQUEUE($Q, \langle v', h + 1, \text{pri}(v', h + 1) \rangle$);
                $hop[v'] \leftarrow h + 1$;
            end
        end
    end
end

return $\langle M, c \rangle$;

算法 3 PRUNEDOPTMC

公式 3.1 的正确性可以由引理 3.1.1 直接说明。

引理 3.1.2：pri(v,h) 是 v 及其后续可搜索节点作为证书节点能够得到的生成树覆盖的集合数量上界。

证明：对 ud 数组使用三角形不等式可证明。具体可参见原始 CBA 论文[2]。 □

Input: Graph G, Query $\mathbb{K}$, diameter bound D, certificate v
Output: multiset of terminals M_v
$M_v \leftarrow \emptyset$;
for $K \in \mathbb{K}$ **do**
  $u \leftarrow \texttt{argmin}_{u' \in K}$ GETWD $(v, u', \frac{D}{2})$;
  if wd $(v, u, \frac{D}{2}) < \infty$ **then**
    $M_v \leftarrow M_v \cup \{u\}$;
  end
end
return M_v

算法 4　MAXM

MaxM。本模块为算法 4，计算在给定证书节点 v 时，终端可重集的具体内容。具体的，对于当前的证书节点，枚举每个 $K \in \mathbb{K}$，使用 GETWD 查询 $u \in K$ 中距离节点 v 在跳数 $\frac{D}{2}$ 内的拥有最短距离的点，并把其加入终端可重集 M。

Input: Tree T, Path P
Output: Merged Tree T
for each $Edge(u, v) \in P$ **do**
  if $Edge(u, v) \notin$ **then**
    $T \leftarrow T \cup$ Edge(u, v);
  end
end
return T

算法 5　MERGE

Merge。此模块为算法 5，将一条路径合并至一棵树上。本文定义被合并出的树树根为 c，也即证书节点，而每个被合并的路径都具有端点 c，因此路径一定是前半部分边存在于树上，后半部分边不存在于树上，可以比较简单的进行合并。

3.1.3 算法分析与证明

引理 3.1.3：CBA 算法给出了一个可行解。

引理 3.1.4：CBA 算法给出的解 T 满足：

$$\mathsf{wt}(T) \leq \sum_{u \in M} \mathsf{wd}(c, u, \frac{D}{2}) \tag{3.2}$$

引理 3.1.5：CBA 算法可行解的近似比为一次查询中关键词集合的数量，即 $|\mathbb{K}|$。

以上三个引理的证明可以参见 CBA 算法的原始论文[2]。其具体对于近似比的放缩思路是将最优可行解放缩至 PRUNEDOPTMC 中找到的每条 c 到终端可重集的路径权值和，得到比例 $|\mathbb{K}|$ 。

时间复杂度。由于本文的重点在于对同一个图中多次的关键词搜索查询，因此对每个图构建 HBLL 的相关复杂度不做计算，仅使用 $O(L)$ 表示 HBLL 中标签的平均大小作为后续计算的依据。对于 $O(L)$ 在实际中的情况，具体可参见小节 4.4。

为了方便后文的讨论，本文规定 n, m 分别表示当前图点数和边数；g 表示一次查询中关键词集合的数量，即 $|\mathbb{K}|$；S 表示每个平均的关键词查询集合的大小，即 $\frac{\sum_{K_i \in \mathbb{K}} |K_i|}{|\mathbb{K}|}$；$L$ 表示 HBLL 中平均标签的大小；D 表示直径限制。其中由于实际应用中图的性质和有直径限制的组斯坦纳树问题的内聚性要求，g, D 都是很小的值，一般不超过 10，对于 D，本文仅对其为 $2, 4, 6$ 的情况进行了实验，实践证明，这已经可以达到实际应用的要求。而参数 L, S 根据实际情况，一般为中等大小的值，约为 n, m 的根号级别。

在 CBA 算法的两步中，第二步合并了 $O(g)$ 条从证书节点到终端节点的路径，每次合并首先通过调用 GETMWP 找到证书节点到当前终端节点的具体路径，单次时间复杂度为 $O(L)$，同时，由于路径跳数不超过 D，因此合并一条路径的时间复杂度为 $O(D)$，总复杂度 $O(g(D + L))$，不在主要的瓶颈上。

而在寻找证书节点 c 和可重终端集合 M 的过程中，时间瓶颈在于调用 HBLL 进行信息查询，而这一部分的所有查询都是点到集合的查询，由于查询集合大小都是 $O(S)$ 级别，因此单次查询时间复杂度为 $O(gSL)$。使用这些信息的部分包括 MAXM，剪枝中查询候选点到询问集合所有点的距离和计算 pri，对于前两者，每个节点最多只会调用一次，对于后者，由于每个节点只会在距离最近集合的最小跳数更新时进入队列，只会入队最多 D 次，因此每个节点最多调用 D 次。再加上算法在寻找节点过程中使用了堆进行维护，整个模块 PRUNEDOPTMC 的时间复杂

度为 $O(ngSLD + nD\log(nD))$。

因此，算法的整体时间复杂度为 $O(ngSLD + nD\log(nD) + g(D + L))$。

3.2 对有跳数限制的地标标签算法的查询优化

3.2.1 基本思想

原始 CBA 算法大量的进行了两类对于 HBLL 方法的调用，第一类是 GETUD(u, v)，主要使用在计算 pri(v, h) 中，每次计算需要查询图中某个节点 v 距离每一个查询集合 $K_i \in \mathbb{K}$ 的跳数，共进行了平均 S 次的调用；第二类是 GETWD(u, v, h)，主要使用在方法 MAXM 以及后续尝试更新最优二元组的过程中，每次计算需要查询图中某个节点 v 距离每一个查询集合 $K_i \in \mathbb{K}$ 的在某个跳数内的最短距离，同样进行了平均 S 次的调用。

可以发现，对于同一个关键词查询 $\mathbb{K} = \{K_1, K_2, \ldots, K_g\}$，算法发生了多次对某个节点 v 和某个查询集合 K_i 的相关距离信息查询，这启发本文将简单的查询 S 次计算最小值进行相应的优化。

注意到 HBLL 对于每个点维护的地标标签集合 $L(v)$ 在内容上是等价的，因此，对于每一个固定的关键词查询，本文利用聚合与缩点的思想，将每个查询集合 K_i 视作图中的一个新的点 vk_i，并提前预处理出它的地标标签集合 $L(vk_i)$，并施加以地标标签集合的简洁性性质，删去多余的三元组。从理论上来讲，如果本文仍然在查询时使用双指针的方法，某个节点 v 与合并后的查询集合 $L(vk_i)$ 发生一次查询产生的时间复杂度为 $O(L + LS)$，与之前的 $O(S(L + L))$ 区别不大，但实际上经过简洁性删除后，合并的新标签的大小是远小于 LS 的，因此实际效率改善不小。

更进一步的，可以观察到算法实际上在两个不均等大小的列表中进行了归并操作，其中一个大小为 $O(L)$，但另一个为 $O(LS)$。容易想到，可以利用二分思想、平衡树等数据结构，将其复杂度改为 $O(L\log(LS))$，但这个方法实际运行起来常数较大，效率一般。为了对这一步进行优化，本文提出了两类方法，一类是基于数组固化，另一类基于哈希表，其中前者使用的内存空间较大，但是常数更小，在实际实现时可以综合考虑选择方法。这两类方法均可以通过预处理将单次节点查询集合的复杂度优化至 $O(L)$，具体的：

假设当前查询限制跳数为 D'，注意到对于 $L(v)$ 中某三元组 $\langle l, h, d\rangle$，本质上需要查询一个在 $L(vk_i)$ 中的另一个三元组 $\langle l', h', d'\rangle$ 满足 $l = l', h + h' \le D'$ 的最

小 $d+d'$，即 $\min\limits_{\substack{\langle l',h',d'\rangle\in L(vk_i)\\ l=l',h+h'\le D'}} d+d'$。

- 基于数组固化的方法：对于每个 $L(vk_i)$ 预处理一个表 $fd_{l,h}$ 表示 $\min\limits_{\substack{\langle l',h',d'\rangle\in L(vk_i)\\ l'=l,h'\le h}} d'$。这个表使用了 $O(nD)$ 的空间，但对于查询三元组 $\langle l,h,d\rangle$，可以直接通过 $fd_{l,D'-h}$ $O(1)$ 获取对应的值。此预处理可以使用动态规划等方法快速完成。
- 基于哈希表的方法：基于哈希表的方法本质上是对于数组固化方法的空间优化，每个询问维护一个哈希函数 $f:\langle l,h\rangle\to\mathbb{N}^+$ 和最优解表 hd_v，对于每个 $\langle l',h',d'\rangle\in L(vk_i)$，在表中维护 $h''\in\{h',h'+1,\dots,D-1,D\}$：$hd_{f(\langle l',h''\rangle)}\leftarrow \min(hd_{f(\langle l',h''\rangle)},d')$。类似的，查询三元组 $\langle l,h,d\rangle$ 仍然可以通过 $hd_{f(\langle l,D'-h\rangle)}$ 进行快速查询。对与设计较好的哈希函数，此方法可以在平均 $O(1)$ 的时间复杂度内完成。

3.2.2 具体内容

Input: a Landmark Labeling set $L=\{L(v_1),L(v_2),\dots,L(v_S)\}$
Output: an aggregated Landmark Labeling $L(vk)$
$L(vk)\leftarrow$ an empty set of triple $\langle l,h,d\rangle$;
for *each* $L(v_i)\in L$ **do**
  $L(vk)\leftarrow L(vk)\cup L(v_i)$;
end
Sort $L(vk)$ by triple $\langle l,h,d\rangle$;
$\langle l',h',d'\rangle\leftarrow\langle 0,0,0\rangle$;
for *each triple* $\langle l,h,d\rangle\in L(vk)$ **do**
  if $l=l'$ *and* $d\ge d'$ *and* $h\ge h'$ **then**
    $L(vk)\leftarrow L(vk)-\{\langle l,h,d\rangle\}$;
  else
    $\langle l',h',d'\rangle\leftarrow\langle l,h,d\rangle$
  end
end
return $L(vk)$;

算法 6　AggregHBLL

AggregHBLL。详见算法 6。本模块将某个查询集合的地标标签标签进行合并，因为每个集合的具体三元组在形式上是相同的，因此仅需要将它们简单的合并在一起，并按照原始 HBLL 的简洁性要求有序的排布并删去多余的三元组，时间复

Input: a Landmark Labeling $L(v)$,G's node number n, diameter bound D
Output: an $n \times D$ array fd
$fd \leftarrow$ an array of $n \times D$ with initial ∞;
for *each triple* $\langle l, h, d \rangle \in L(v)$ **do**
  for $D' \leftarrow h$ **to** D **do**
    $fd_{l,D'} \leftarrow \min(fd_{l,D'}, d)$;
  end
end
return fd;

算法 7 ConstructFD

杂度为 $O(LS\log(LS))$。

ConstructFD。详见算法 7。本模块将某个被合并后的地标标签集合进行数组固化，采用了类似动态规划的算法进行实现。时间复杂度为 $O(LD)$。由于数组可以在外部类每个对象中提前开好，在使用完毕后收回对应内存，所以不计入时间复杂度的计算。

Input: a Landmark Labeling $L(v)$, diameter bound D
Output: an approximate $L \times D$ array hd
$f \leftarrow$ an empty HashTable whose max $f(\langle x, y \rangle) \geq L \times D$;
$hd \leftarrow$ an array of length max $f(\langle x, y \rangle)$ with initial ∞;
for *each triple* $\langle l, h, d \rangle \in L(v)$ **do**
  for $D' \leftarrow h$ **to** D **do**
    $hd_{f(\langle l,D' \rangle)} \leftarrow \min(hd_{f(\langle l,D' \rangle)}, d)$;
  end
end
return fd;

算法 8 ConstructHD

ConstructHD。详见算法 8。本模块与上一个模块类似。所不同的是使用了哈希表先对二元组 $\langle l, h \rangle$ 做了一次映射，每次根据需要做二元组的个数 $O(LD)$ 级别建立哈希表，具体大小可以取同级复杂度的常数级别倍数，具体实现方法很多，也可以借用现有的实现。接着，将做了一次映射后的结果放入表 hd 中，其时间复杂度与 ConstructFD 模块相同，但是空间上有较明显的改善。

3.3 最优性剪枝的使用

原始 CBA 算法中已经使用了若干剪枝，本文再补充三类。

基于虚假最短距离的剪枝。考虑到 CBA 算法使用优先队列的过程中非常类似使用 dijkstra 求解非负权图的多源最短路，但是与 dijkstra 所不同的是，优先队列中维护的不是到源点的最短路，而是本文在公式 3.1 中定义的 pri 数组。虽然维护的内容不同，但是初始被塞入优先队列中的节点都是源点，它们的最终在计算 GETUD 时得到的结果都为 0，同时，在算法中逐点扩散时，可以比较清晰的维护出每个点在经过优先队列中节点后，到达源点的最小距离。显然，由于本文使用了真实的非负边权，这个结果相对于真正的最短距离而言，只可能是一个偏大的数值，但是其仍然可以用作在原始 PRUNEDOPTMC 中计算每个节点到查询集合所有节点距离的估计值，剪去不必要的 GETUD 的调用。

除了这一部分，当某个点的 pri 值已经到达查询集合个数即 $\mathbb{K}$ 时，在其即将入队时，可以提前采用 $|M| \cdot dis[v'] \geq Msum$ 剪枝拒绝其入队。显然，由于最优可重终端节点集合 M 已经无法继续增大，节点 v' 已经不再可能更新答案，而其在跳数范围内的后继结点，若能更新答案一定会被其他的节点加入队列。

具体的实现详见算法 9 PRUNEDOPTMC+，首先初始化一个类似 $hop[u]$ 的数组 $dis[v]$，表示对于每个节点对查询集合的距离估计，然后在尝试 $\text{pri}(v,h) = |M|$ 情况时，首先使用 $dis[u]$ 作为 $\min_{u \in \bigcup_{k \in \mathbb{K}} K} \text{wd}(v,u,\frac{D}{2})$ 的估计进行一次筛选。最后是在最优可重终端节点集合 M 已经无法继续增大时，通过同样的不等式拒绝当前节点进入优先队列。

基于直径上界的剪枝。这方面的剪枝思路比较简单，但是十分有效。考虑最终实现有条数限制的地标标签算法时，在直径上界为 D 时，调用标签计算最短跳数或最短距离时的限制为 $\frac{D}{2}$，因此对于满足 $h > \frac{D}{2}$ 的三元组 $\langle l,h,d \rangle$，算法并没有遍历它的需要。为了快速跳过这些位置，可以预处理出原始 HBLL 标签存储结构中对应的下标集合，也可以维护出每个位置在限制不大于某值时的下一个下标位置。因为内聚性的要求，这些限制的个数有限，对整体的空间复杂度影响不大。具体剪枝效果可以参见小节 4.4。

基于 HBLL 退化的剪枝。在整体的 CBA 算法中，由于优先队列的权值是基于跳数的，因此最终 GETUD 的调用一定是严格多余 GETWD 的。同时，可以注意到在计算 GETUD 时，本质上对于三元组 $\langle l,h,d \rangle$ 来说，第三个参数 d 没有意义，同

Input: Graph $G = \langle V, E \rangle$, Query $\mathbb{K}$, diameter bound D

Output: multiset of terminals M and certificate c

...

$M\,sum \leftarrow 0$;

for $u \in \bigcup_{K \in \mathbb{K}} K$ **do**

  ENQUEUE($Q, \langle u, 0, \text{pri}(u, 0) \rangle$);

  $hop[u] \leftarrow 0$; $dis[u] \leftarrow \infty$;

  Qsta(u) $\leftarrow$ Qsta(u) $\cup$ id(u);

end

while Q *is not empty* **do**

  ...

  if pri(v, h) $= |M|$ **then**

    if $|M| \cdot dis[v] \geq M\,sum$ *or* $|M| \cdot \min_{u \in \bigcup_{k \in \mathbb{K}} K} \text{wd}(v, u, \frac{D}{2}) \geq M\,sum$ **then**

      continue the while loop;

    end

  end

  if *checked[v]=false* **then**

    ...

    if M_v *better than* M **then**

      $\langle M, c \rangle \leftarrow \langle M_v, v \rangle$;

      $M\,sum \leftarrow \sum_{u \in M} \text{wd}(v, u, \frac{D}{2})$;

    end

  end

  if $h < \frac{D}{2}$ **then**

    for $v' \in N(v)$ **do**

      $dis[v'] \leftarrow \min(dis[v'], dis[u] + \text{wt}(v, v'))$;

      if $h + 1 < hop[v']$ **then**

        if $|M| = |\mathbb{K}|$ *and* $|M| \cdot dis[v'] \geq M\,sum$ **then**

          continue the for loop;

        end

        ENQUEUE($Q, \langle v', h + 1, \text{pri}(v', h + 1) \rangle$);

        $hop[u] \leftarrow h + 1$;

        Qsta(v') $\leftarrow$ Qsta(v') $\cup$ Qsta(u);

      end

    end

  end

end

return $\langle M, c \rangle$;

算法 9 PRUNEDOPTMC+

时，在两个三元组 l 相等时，也仅需要 h 更小的三元组。因此，这本质上是 HBLL 向 PLL 的一个退化，但没有必要再重新建立 PLL 对此进行维护，只需要类似与基于直径上界的剪枝中那样，维护下一个下标位置快速进行跳转即可。具体剪枝效果可以参见小节 4.4。

3.4 减少函数调用

原始的 CBA 算法产生对于同一信息的多次调用，最典型的即为 GETUD 与 GETWD 的重复调用，这会在某些节点重复进入优先队列时产生。同时，也产生了一些可以直接计算答案的不必要调用。

固化重复的调用。使用表 $resud[u][k], reswd[u][k]$ 记录当前询问已经查询过的 GETUD (u,k), GETWD (u,k)，在再次查询时直接 $O(1)$ 查表。表的空间复杂度为 $O(ng)$，在实现时可以提前开好，每次询问使用，询问结束时逐一清空表项，不产生额外的时空复杂度。

减少不必要调用。显然，在将若干源点加入队列时，这些节点均来自某个查询集合，其对于所属的查询集合的跳数不再需要查询，它们扩展出的节点也同样不需要查询。因此，使用 $Qsta(u)$ 集合表示 u 点一定距离哪些查询集合在 $\frac{D}{2}$ 范围内，$id(u)$ 表示源点中节点 u 属于哪个查询集合，其具体的实现在算法 9 PRUNEDOPTMC+ 中。代码中可以使用二进制表示集合来优化过程。

3.5 总体分析

基于原始 CBA 算法的效率优化中，本文主要进行了以下三个方面：一是对有跳数限制的地标标签算法中针对于图中节点对特定点集的跳数、跳数限制的距离做了相应的优化，在基本不影响空间的情况下，将相关查询中关于单个询问点集的量级去掉；二是针对原算法的剪枝进行了进一步的加强，提出了基于虚假最短距离的剪枝，极大的减少了对 GETWD 的调用；三是针对重复的查询和显然没有必要的查询进行了信息的存储和固化，减少算法的常数。

具体的，如小节 3.1.3 中约定的那样，在算法的第一部分调用 HBLL 进行单次查询的时间复杂度被改进为 $O(gL)$，整个 PRUNEDOPTMC 的时间复杂度被改进为 $O(ngLD+nD\log(nD)$, 整体的时间复杂度也变为 $O(ngLD+nD\log(nD)+g(D+L))$。实际上，从算法的表现来看，它的效率比理论复杂度要快得多。

同时，本文并没有改变保证原始 CBA 算法中近似比为 $\mathbb{K}$ 的相关操作，即仍然

选择最优的证书节点 c 和其对应的路径和最小的终端节点集合 $\boldsymbol{M}$。

4 实验

4.1 实验设置

表 4.1 图与询问

图	点数 (n)	边数 (m)	查询数量	集合组数 (g)
MND	37,303	111,740	40	1-4
CYC	61,382	157,683	50	1-6
YAGO	632,308	922,627	32	2-6
LMDB	840,520	2,132,799	200	1-10
DBP	5,900,558	18,743,178	438	1-10
LUBM-50K	55,664	166,682	250	2-6
LUBM-500K	539,313	1,643,567	250	2-6
LUBM-5M	4,824,109	14,703,746	250	2-6

本文将在英特尔至强黄金 5220R（2.20GHz）中央处理器中进行实验，为 C++ 实验代码提供 256GB 实验内存。

图。关于无向图的选取，本文采取了五个不同大小的真实图和基于 LUBM① 生成的合成图。其中，真实图分别为小图 MONDIAL②(MND) 和 OpenCyc③(CYC)，中等图 YAGO④ 和 LinkedMDB⑤(LMDB)，大图 DBpedia⑥(DBP)；虚拟图本文基于 LUBM 生成了三种大小的图，分别为 LUBM-50K, LUBM-500K 和 LUBM-5M。

表 4.1 展示了每个图的大小。

查询。针对真实图，本文收集了真实的关键词查询，并将每个关键词映射到一个包含该关键词的顶点组（即知识图谱中的实体），这些实体的名称包含该关键词。具体而言，对于 MND，本文重新使用了评估工作[1] 中的查询。对于 CYC，本文构建了包含 $1-6$ 个关键词的 50 个查询，这些关键词是随机从实体名称中抽样得到的。对于 YAGO，本文重新使用了基准测试[17] 中的查询。对于 LMDB，本文

①http://swat.cse.lehigh.edu/projects/lubm/
②https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.rdf
③https://master.dl.sourceforge.net/project/texai/open-cyc-rdf/1.1/open-cyc.rdf.ZIP? viasf=1
④https://yago-knowledge.org/data/yago1/yago-1.0.0-turtle.7z
⑤https://www.cs.toronto.edu/ oktie/linkedmdb/linkedmdb-latest-dump.zip
⑥https://downloads.dbpedia.org/2016-10/core/mappingbased_objects_en.ttl.bz2

表 4.2 对于单位权图（UW）每个询问的平均运行时间（秒）和标准差

图	算法	$D = 2$	$D = 4$	$D = 6$
OpenCyc	CORE	0.002(0.002)	0.058(0.144)	0.019(0.042)
	PRUNEDCBA	0.000(0.000)	**0.001(0.002)**	0.013(0.021)
	PRUNEDCBA+	**0.000(0.000)**	0.002(0.002)	**0.008(0.012)**
Mondial	CORE	0.003(0.004)	0.004(0.007)	0.002(0.002)
	PRUNEDCBA	0.000(0.000)	0.001(0.002)	0.004(0.009)
	PRUNEDCBA+	**0.000(0.000)**	**0.001(0.001)**	**0.001(0.001)**
YAGO	CORE	0.048(0.146)	1.688(4.095)	1.701(8.280)
	PRUNEDCBA	0.003(0.001)	0.017(0.025)	0.089(0.131)
	PRUNEDCBA+	**0.003(0.001)**	**0.015(0.022)**	**0.043(0.061)**
LinkedMDB	CORE	0.092(0.172)	5.606(12.481)	3.237(9.466)
	PRUNEDCBA	0.006(0.002)	0.028(0.034)	0.271(0.378)
	PRUNEDCBA+	**0.006(0.002)**	**0.017(0.019)**	**0.101(0.173)**
Dbpedia	CORE	2.025(8.005)	13.947(21.558)	1.632(5.871)
	PRUNEDCBA	0.029(0.023)	0.583(1.231)	6.444(10.664)
	PRUNEDCBA+	**0.031(0.015)**	**0.236(0.383)**	**1.238(2.129)**
LUBM-50K	CORE	1.659(3.023)	0.120(0.122)	0.020(0.031)
	PRUNEDCBA	0.008(0.014)	0.020(0.026)	0.046(0.048)
	PRUNEDCBA+	**0.003(0.003)**	**0.006(0.007)**	**0.009(0.009)**
LUBM-500K	CORE	10.807(20.490)	1.335(1.473)	0.167(0.235)
	PRUNEDCBA	0.012(0.018)	0.059(0.090)	0.346(0.324)
	PRUNEDCBA+	**0.005(0.004)**	**0.012(0.011)**	**0.039(0.032)**
LUBM-5M	CORE	5.643(11.884)	6.964(13.874)	0.292(0.313)
	PRUNEDCBA	0.025(0.021)	0.224(0.282)	1.205(1.027)
	PRUNEDCBA+	**0.019(0.006)**	**0.082(0.083)**	**0.410(0.684)**

从[18]中抽样了 200 个问题，并通过去除停用词和标点符号将它们转换为关键词查询。对于 DBP，本文重新使用了由 DBpedia-Entity v2 测试集提供的查询[19]。

对于合成图，本文遵循了[11]的方法，直接通过改变组数（用 g 表示）和每个组中顶点数（用 f 表示）来构建组。对于每个设置 $\langle g, f\rangle \in \{\langle 2,100\rangle, \langle 4,100\rangle, \langle 6,100\rangle, \langle 4,10\rangle, \langle 4,1000\rangle\}$，本文按照 g, f 的大小随机采样 50 组顶点作为询问。

表 4.1 展示了每个询问的细节。

边权设置。本文使用两种边权形式。第一种是单位权（**UW**），为图中每条边赋予相同的权值，这与跳数计算的结果一致；第二种是基于信息的边权（**IW**）[20–22]，通过给每条边在知识图谱中同类型边的数量的对数赋予实数权值，表示同类型的关系出现的次数的重要性。

基准模型。本文将 PRUNEDCBA+ 算法与两个基准模型进行比较：CORE[1]，第一类提出解决有直径限制的斯坦纳树问题的算法，其仅仅给出一个可行解但不保证近似比；PRUNEDCBA[2]，即剪枝优化的原始 CBA 算法，其给出了明确的近似比，

表 4.3 对于基于信息的边权图（IW）每个询问的平均运行时间（秒）和标准差

图	算法	$D = 2$	$D = 4$	$D = 6$
OpenCyc	Core	0.002(0.002)	0.077(0.190)	0.027(0.055)
	PrunedCBA	0.000(0.000)	**0.001(0.002)**	0.013(0.017)
	PrunedCBA+	**0.000(0.000)**	0.002(0.002)	**0.007(0.010)**
Mondial	Core	0.003(0.004)	0.004(0.005)	0.003(0.003)
	PrunedCBA	0.000(0.000)	0.002(0.002)	0.004(0.006)
	PrunedCBA+	**0.000(0.000)**	**0.001(0.001)**	**0.001(0.001)**
YAGO	Core	0.059(0.182)	2.230(5.509)	1.840(8.731)
	PrunedCBA	0.003(0.001)	0.018(0.026)	0.110(0.139)
	PrunedCBA+	**0.003(0.001)**	**0.016(0.022)**	**0.045(0.056)**
LinkedMDB	Core	0.159(0.297)	6.999(14.257)	4.023(10.689)
	PrunedCBA	0.007(0.003)	0.027(0.032)	0.335(0.395)
	PrunedCBA+	**0.006(0.002)**	**0.018(0.020)**	**0.113(0.171)**
Dbpedia	Core	3.120(14.348)	17.144(26.170)	2.135(6.688)
	PrunedCBA	**0.026(0.019)**	0.647(1.319)	8.503(12.254)
	PrunedCBA+	0.031(0.016)	**0.273(0.420)**	**1.073(2.287)**
LUBM-50K	Core	1.980(3.610)	0.140(0.144)	0.023(0.035)
	PrunedCBA	0.008(0.015)	0.027(0.036)	0.061(0.056)
	PrunedCBA+	**0.002(0.003)**	**0.004(0.004)**	**0.006(0.005)**
LUBM-500K	Core	10.849(20.408)	1.444(1.582)	0.181(0.254)
	PrunedCBA	0.013(0.018)	0.067(0.104)	0.323(0.274)
	PrunedCBA+	**0.004(0.003)**	**0.008(0.006)**	**0.023(0.014)**
LUBM-5M	Core	5.821(12.301)	7.321(14.167)	0.304(0.336)
	PrunedCBA	**0.025(0.021)**	0.223(0.253)	1.793(1.988)
	PrunedCBA+	0.029(0.024)	**0.079(0.093)**	**0.177(0.201)**

运行效率较快。这两类代码的 C++ 实现均参考了它们的开源代码。

直径限制。本文使用三个设置：$D \in \{2, 4, 6\}$。

4.2 效率实验

本节比较 Core，PrunedCBA，PrunedCBA+ 的找到可行解的具体运行时间。

指标。每个算法在每个图中的每个独立的询问的运行时间。

结果。表 4.2 和表 4.3 展示了不同的算法对于每个询问在不同的图之间（包括真实图和合成图），分别基于单位权和基于信息的边权得到的运行时间的平均值和标准差。在 MND, CYC, YAGO, LMDB, LUBM-50K, LUBM-500K 这些顶点数在数万以及数十万之间的图上，PrunedCBA 和 PrunedCBA+ 都能快速的得到结果，实际情况比较接近，但是 Core 算法就表现的比较一般。在 DBP 和 LUNM-5M 这些顶点数为数百万的图上，PrunedCBA+ 与 PrunedCBA 的表现差距就更加明显，前者比后者平均快了 3 – 4 倍。这展现了 PrunedCBA+ 对于算法效率的有效优化。

4.3 消融实验

表 4.4 消融实验：在图 YAGO 和 LUBM-500K 上每个询问的平均运行时间（毫秒）和标准差

图-边权	算法	$D=2$	$D=4$	$D=6$
YAGO-UW	w/o HBLL+	**2.812(1.073)**	15.875(24.451)	52.406(80.271)
	w/o prun.	3.000(1.031)	21.094(34.733)	47.000(71.824)
	w/o prev.	3.156(1.439)	16.719(24.835)	45.219(61.673)
	PrunedCBA+	3.000(1.392)	**15.375(22.452)**	**43.031(61.050)**
LUBM-500K-UW	w/o HBLL+	12.376(18.716)	57.064(90.380)	147.064(149.092)
	w/o prun.	**4.380(2.926)**	49.568(61.803)	60.128(46.611)
	w/o prev.	4.856(4.172)	13.108(14.108)	46.104(36.944)
	PrunedCBA+	4.864(3.514)	**11.920(10.723)**	**38.620(32.230)**
YAGO-IW	w/o HBLL+	3.094(1.422)	16.906(26.143)	57.719(85.276)
	w/o prun.	**2.688(1.014)**	24.344(42.556)	55.094(86.981)
	w/o prev.	3.094(1.071)	18.219(25.346)	51.688(70.029)
	PrunedCBA+	3.219(1.340)	**15.812(22.210)**	**45.000(56.332)**
LUBM-500K-IW	w/o HBLL+	11.996(17.774)	40.908(61.809)	98.336(87.154)
	w/o prun.	5.132(3.527)	33.884(28.417)	53.440(33.464)
	w/o prev.	4.676(3.481)	9.592(7.853)	29.816(18.148)
	PrunedCBA+	**4.272(2.976)**	**8.040(5.694)**	**23.080(13.847)**

本节进行消融实验，比较去掉对有跳数限制的地标标签算法的优化（记为 w/o HBLL+）、去掉增加的最优性剪枝（记为 w/o prun.）、去掉重复调用（记为 w/o prev.）三种优化与完整的 PrunedCBA+ 算法在时间上的差异。

指标。选取一个真实图 YAGO 和一个合成图 LUBM-500K，进行在两类边权上的实验，比较每个独立的询问的运行时间。

结果。表 4.4 展示了每个算法在真实图 YAGO 和合成图 LUBM-500K 上的平均运行时间（毫秒）和标准差。对于 $D=2$ 时，由于图的搜索距离很短，空间非常小，因此本文的几类优化不总是十分有效。但在 $D=4,6$ 时，由于搜索空间急剧变大，完整的 PrunedCBA+ 算法的在效率上表现的就远超过不适用任何一个剪枝得到的结果了。

4.4 地标标签算法参数相关实验

本节比较基于直径上界的剪枝和基于 HBLL 退化的剪枝对于单个顶点平均标签大小的具体影响。

指标。选取每个图中基于单位权和基于信息的边权的 HBLL 情况，假设 HBLL 中每个顶点存储了若干标签三元组 $\langle l,h,d\rangle$，本节对比不经过任何剪枝的三元组

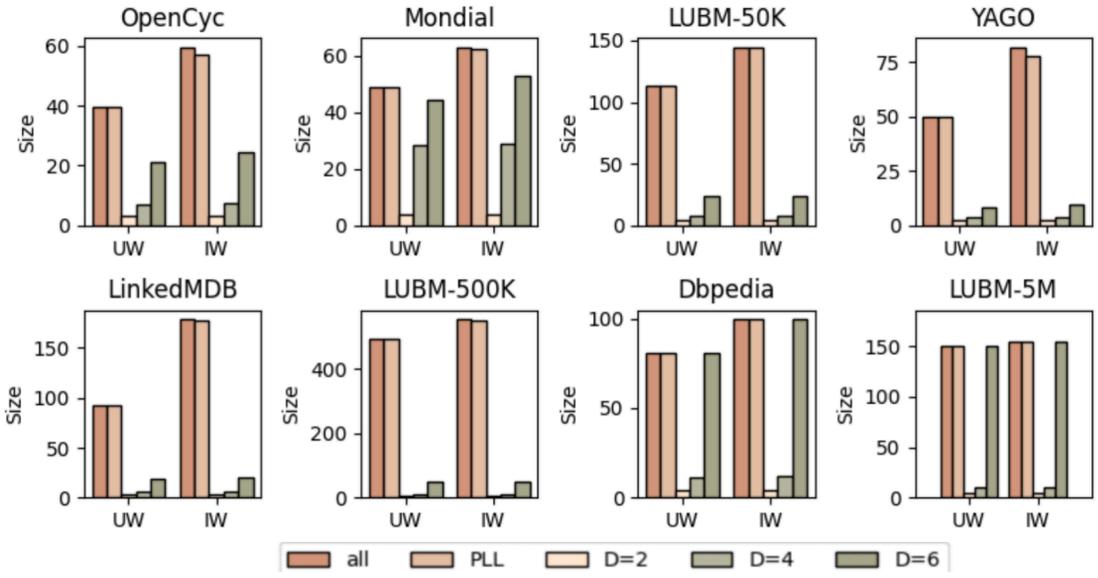

图 4.1　不同图中不同边权形式中使用不同剪枝的顶点地标标签的平均大小

（标签为 all），只保留 $h \leq 1,2,3$ 的三元组（标签分别为 $D=2$、$D=4$、$D=6$），对每个 l 只保留最小的 h 的三元组（标签为 PLL）这五类情况中，每个顶点的平均标签三元组大小。

结果。图 4.1 直观地展示了每个图中五类情况的具体大小。可以观察出，无论哪个图，对于基于 HBLL 退化的剪枝，即对每个 l 只保留最小的 h 的三元组的优化对于整体的平均大小影响不大；而对于基于直径上界的剪枝中，$D=2,4$ 的情况影响都很大，其极大的减少了平均标签的大小。但在 $D=6$ 的情况中，小图的剪枝效果仍然优秀，但在顶点数为数百万的大图 Dbpedia 和 LUBM-5M 中，其剪枝效果就不甚理想了。总的来说，这两类基于地标标签算法的剪枝整体上对算法的时间效率的提升还是较为明显的。

5 总结与展望

本文聚焦于优化具有直径限制的组斯坦纳树问题，以应对大型复杂知识图谱中关键词搜索的挑战。在当前研究基础上，通过引入哈希表和动态聚合优化地标标签算法（HBLL）的查询过程，利用最优性剪枝原理减少候选点查询范围，并动态固化查询信息以减少查询调用次数，从而提高了算法的时空效率，不仅在实验中取得了良好的效果，而且在不降低近似比的前提下，改进了现有算法的时空复杂度。

具体而言，本文首先系统地介绍了具有直径限制的组斯坦纳树问题的背景及其在知识图谱中的重要性。针对现有算法在处理大规模图谱、覆盖不完整关键词集等方面存在的挑战，提出了一种优化算法。该算法通过对地标标签算法的查询过程进行优化，并结合最优性剪枝原理和动态固化查询信息的策略，有效提高了算法的效率和准确性。然后本文详细介绍了算法的具体实现方式，包括算法的原理、模块化设计以及近似比的证明。最后，通过对实验设置和结果的描述，本文验证了所提出方法的有效性和可行性。

尽管本研究在优化有直径限制的组斯坦纳树问题上取得了显著进展，但仍存在一些可以改进和深入探讨的方向，例如进一步探索更加高效的查询优化策略，结合机器学习或深度学习方法，根据图谱的特性和查询模式动态调整算法参数，以进一步提高算法的性能和适用性；考虑拓展研究对象，探索适用于不同类型图谱或具有特定约束条件的组斯坦纳树问题的解决方案，如考虑节点属性信息、多图谱融合等问题。此外，可以进一步研究算法在实际应用场景中的可扩展性和可部署性，如在大规模分布式系统中的并行计算和实时查询支持等方面进行深入探讨。而随着人工智能和知识图谱领域的不断发展，未来的研究方向可能还包括与其他前沿技术的融合，如图神经网络、自然语言处理等，以进一步拓展知识图谱应用的边界和深度。

综上所述，本研究在优化有直径限制的组斯坦纳树问题上取得了一定进展，并提出了一种有效的算法解决方案。未来的研究可以在此基础上进一步深入探讨，以应对知识图谱领域中的挑战和需求，推动相关技术的发展与应用。

参考文献

[1] CHENG G, LI S, ZHANG K, et al. Generating compact and relaxable answers to keyword queries over knowledge graphs[C/OL]//The Semantic Web –ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part I. Berlin, Heidelberg: Springer-Verlag, 2020: 110–127. https://doi.org/10.1007/978-3-030-62419-4_7.

[2] ZHANG K, WANG X, CHENG G. Efficient approximation algorithms for the diameter-bounded max-coverage group steiner tree problem[C/OL]//WWW '23: Proceedings of the ACM Web Conference 2023. New York, NY, USA: Association for Computing Machinery, 2023: 199–209. https://doi.org/10.1145/3543507.3583257.

[3] CHENG G, LIU D, QU Y. Efficient algorithms for association finding and frequent association pattern mining[C]//GROTH P, SIMPERL E, GRAY A, et al. The Semantic Web – ISWC 2016. Cham: Springer International Publishing, 2016: 119-134.

[4] CHENG G, LIU D, QU Y. Fast algorithms for semantic association search and pattern mining[J]. IEEE Transactions on Knowledge and Data Engineering, 2021, 33(4): 1490-1502.

[5] LI S, HUANG Z, CHENG G, et al. Enriching documents with compact, representative, relevant knowledge graphs[C/OL]//BESSIERE C. Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20. International Joint Conferences on Artificial Intelligence Organization, 2020: 1748-1754. https://doi.org/10.24963/ijcai.2020/242.

[6] KAPOOR S, SARWAT M. Bounded-diameter minimum-cost graph problems [J/OL]. Theory of Computing Systems, 2007, 41(4): 779. https://www.proquest.com/scholarly-journals/bounded-diameter-minimum-cost-graph-problems/docview/237224226/se-2.

[7] MASHREGHI A, ZAREI A. When diameter matters: Parameterized approximation algorithms for bounded diameter minimum steiner tree problem[J/OL]. Theory of Computing Systems, 2016, 58(2): 287-303. https://www.proquest.com/

scholarly-journals/when-diameter-matters-parameterized-approximation/docview/1756028395/se-2.

[8] KOU L, MARKOWSKY G, BERMAN L. A fast algorithm for steiner trees[J]. Acta Informatica, 1981, 15: 141-145.

[9] BYRKA J, GRANDONI F, ROTHVOSS T, et al. Steiner tree approximation via iterative randomized rounding[J/OL]. J. ACM, 2013, 60(1). https://doi.org/10.1145/2432622.2432628.

[10] SHI Y, CHENG G, KHARLAMOV E. Keyword search over knowledge graphs via static and dynamic hub labelings[C/OL]//WWW '20: Proceedings of The Web Conference 2020. New York, NY, USA: Association for Computing Machinery, 2020: 235–245. https://doi.org/10.1145/3366423.3380110.

[11] LI R H, QIN L, YU J X, et al. Efficient and progressive group steiner tree search [C/OL]//SIGMOD '16: Proceedings of the 2016 International Conference on Management of Data. New York, NY, USA: Association for Computing Machinery, 2016: 91–106. https://doi.org/10.1145/2882903.2915217.

[12] DING B, XU YU J, WANG S, et al. Finding top-k min-cost connected trees in databases[C]//2007 IEEE 23rd International Conference on Data Engineering. 2007: 836-845.

[13] SUN Y, XIAO X, CUI B, et al. Finding group steiner trees in graphs with both vertex and edge weights[J/OL]. Proc. VLDB Endow., 2021, 14(7): 1137–1149. https://doi.org/10.14778/3450980.3450982.

[14] SHI Y, CHENG G, TRAN T K, et al. Keyword-based knowledge graph exploration based on quadratic group steiner trees[C/OL]//ZHOU Z H. Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21. International Joint Conferences on Artificial Intelligence Organization, 2021: 1555-1562. https://doi.org/10.24963/ijcai.2021/215.

[15] COHEN E, HALPERIN E, KAPLAN H, et al. Reachability and distance queries via 2-hop labels[C]//SODA '02: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms. USA: Society for Industrial and Applied Mathematics, 2002: 937–946.

[16] AKIBA T, IWATA Y, YOSHIDA Y. Fast exact shortest-path distance queries on

large networks by pruned landmark labeling[C/OL]//SIGMOD '13: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York, NY, USA: Association for Computing Machinery, 2013: 349–360. https://doi.org/10.1145/2463676.2465315.

[17] YOGHOURDJIAN. H, ELBASSUONI. S, JABER. M, et al. Top-k keyword search over wikipedia-based rdf knowledge graphs[C]//Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2017) - KDIR. SciTePress, 2017: 17-26.

[18] MILLER A, FISCH A, DODGE J, et al. Key-value memory networks for directly reading documents[C/OL]//SU J, DUH K, CARRERAS X. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. Austin, Texas: Association for Computational Linguistics, 2016: 1400-1409. https://aclanthology.org/D16-1147.

[19] HASIBI F, NIKOLAEV F, XIONG C, et al. Dbpedia-entity v2: A test collection for entity search[C/OL]//SIGIR '17: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, NY, USA: Association for Computing Machinery, 2017: 1265–1268. https://doi.org/10.1145/3077136.3080751.

[20] ANYANWU K, MADUKO A, SHETH A. Semrank: ranking complex relationship search results on the semantic web[C/OL]//WWW '05: Proceedings of the 14th International Conference on World Wide Web. New York, NY, USA: Association for Computing Machinery, 2005: 117–127. https://doi.org/10.1145/1060745.1060766.

[21] CHENG G, SHAO F, QU Y. An empirical evaluation of techniques for ranking semantic associations[J]. IEEE Transactions on Knowledge and Data Engineering, 2017, 29(11): 2388-2401.

[22] HALPERIN E, KRAUTHGAMER R. Polylogarithmic inapproximability[C/OL]//STOC '03: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing. New York, NY, USA: Association for Computing Machinery, 2003: 585–594. https://doi.org/10.1145/780542.780628.

致谢

日月如梭，斗转星移，我在武汉大学的求学时光即将画上句号。在毕业论文的最后，我想对所有在完成毕设工作中给予我支持与帮助的人们送上我最真挚的感谢。

首先，我非常感谢南京大学计算机科学与技术系建立万维网软件研究组（Websoft），作为我即将投入科研生涯的地方，小组为我提供了海量的实验资源，优秀的实验设备，舒适的科研环境，在小组的氛围下，我成功的完成了这篇论文。

我还非常感谢研究组的程龚导师，他对我的毕设论文严格要求、认真监督，提出了许多宝贵的意见，同时，也感谢研究组中的知识计算平台小组，小组同学为我提供了许多宝贵的建议，你们耐心地为我解决疑惑，提供建议，给我的学习和研究提供了无私的帮助；还感谢我在武汉大学的指导老师董文永，他为我提供了一些远程的支持。

特别的，感谢我在武汉大学的 ACM-ICPC 集训队的同学，是他们远程给予了我情绪上的支持；感谢和我一起来南京大学做毕设的两名合租室友，是他们天天和我一块儿吃饭；感谢瑞幸咖啡中的加浓美式，是它们提高了我的工作效率。

最后，需要特别感谢的是我的父母。父母的养育之恩无以为报，他们是我求学路上的坚强后盾，在我面临人生选择的迷茫之际，为我排忧解难，他们对我无私的爱与照顾是我不断前进的动力。

再次感谢所有给予我帮助的人们！