南京大学
NANJING UNIVERSITY

研 究 生 毕 业 论 文
（申 请 硕 士 学 位）

论 文 题 目　基于组斯坦纳树的知识图谱搜索算法研究

作 者 姓 名　石雨轩

专 业 名 称　计算机科学与技术

研 究 方 向　语义网

指 导 教 师　程龚 副教授

2021 年 **5** 月 **28** 日

学　　号：**MG1833063**

论文答辩日期：**2021** 年 **5** 月 **24** 日

指 导 教 师：　　（签字）

Research on Group Steiner Tree Based Search Algorithms over Knowleage Graphs

by

Yuxuan Shi

Supervised by

Associate Professor Gong Cheng

A dissertation submitted to
the graduate school of Nanjing University
in partial fulfilment of the requirements for the degree of
MASTER
in
Computer Science and Technology

Department of Computer Science and Technology
Nanjing University

May 28, 2021

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：基于组斯坦纳树的知识图谱搜索算法研究

计算机科学与技术 专业 2018 级硕士生姓名：石雨轩

指导教师（姓名、职称）：程龚 副教授

摘　要

知识图谱是一张用点代表实体，用边代表实体间关系的图，基于关键词的查询是利用复杂知识图谱的常见方式。传统方法将关键词查询建模为组斯坦纳树问题，查找连接所有关键词的最小权子图。组斯坦纳树是NP-难问题，现有的有理论质量保证的算法并不能在大规模知识图谱上运行。本文提出了两个高效的近似算法 KeyKG 和 KeyKG$^+$ ，并证明 KeyKG 与 KeyKG$^+$ 的近似比都是 $g-1$. 这两个算法基于中心标注法，每个点标注一些可达点，借助可达点计算任意两点间的最短距离。本文提出了两种中心标注：一个新的静态中心标注，它使用新启发式来优化剪枝界标标注法；一个新的动态中心标注，它翻转并聚合查询相关的静态中心标注以加速点集的距离查询。实验表明本文的算法在真实知识图谱上能取得较好的近似结果，且能在百万级的知识图谱上毫秒级的完成查询。

进一步的，现有研究建议最小化子图中实体间的语义距离，以提升结果的语义内聚性，高效计算语义内聚子图是必须解决的问题。本文把这个问题形式化为二次组斯坦纳树问题，组斯坦纳树问题是这个问题的一种特例，因此二次组斯坦纳树也是NP-难问题，本文为此设计了两个近似算法 **QO** 和 **EO** ，并证明 **QO** 的近似比为 $2(g-1)^2$，**EO** 的近似比为 $(g-1)^2|V|$。此外，本文还提出包括剪枝和排序在内的多种启发式来提升算法的实际表现。实验表明本文的算法能将结果的语义内聚性提升 3-4 倍，且能在百万级的知识图谱上秒级的完成查询。

关键词： 知识图谱，关键词搜索，组斯坦纳树，中心标注法，语义内聚性，二次组斯坦纳树

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Research on Group Steiner Tree Based Search Algorithms over Knowleage Graphs

SPECIALIZATION: Computer Science and Technology

POSTGRADUATE: Yuxuan Shi

MENTOR: Associate Professor Gong Cheng

Abstract

A knowledge graph represents a set of entities and their relations. To explore the content of a large and complex KG, a convenient way is keyword-based querying. Traditional methods answer an exploratory keyword query by computing a group Steiner tree, which is a minimum-weight subgraph that connects all the keywords in the query. Group Steiner tree is a NP-hard problem, existing algorithms with provable quality guarantees have prohibitive run time on large graphs. This thesis proposes two approximation algorithms: `KeyKG` and `KeyKG`$^+$, and proves the approximation ratio of `KeyKG` and `KeyKG`$^+$ are both $g-1$. The algorithms rely on Hub Labeling, which labels each vertex in a graph with a list of vertices reachable from it to compute distances and shortest paths. This thesis devises two hub labels: a conventional static hub label that uses a new heuristic to improve pruned landmark labeling, and a novel dynamic hub label that inverts and aggregates query-relevant static labels to more efficiently process vertex sets. Experiments show that the approaches can compute a reasonably good approximation of answers in milliseconds on real-world million-scale knowledge graphs.

Furthermore, recent studies have suggested improving the semantic cohesiveness of a query answer by minimizing the pairwise semantic distances between the entities in a subgraph, but it remains unclear how to efficiently compute such a subgraph. This thesis formulates it as a quadratic group Steiner tree problem. Group Steiner tree problem is a subproblem of quadratic group Steiner tree problem, so quadratic group Steiner tree problem is also NP-hard. This thesis designs two approximation algorithms for the proposed problem: **QO** and **EO**, and proves that the approximation ratio of **QO** and **EO** are $2(g-1)^2$ and $(g-1)^2|V|$ respectively. This thesis further presents various

heuristics, e.g., pruning and ranking strategies, to improve their practical performance. Experiments show that the algorithms can improve the cohesiveness of the answers by up to 3-4 times, and compute answers in seconds on million-scale knowledge graphs.

keywords: knowledge graph, keyword search, group Steiner tree, hub labeling, semantic cohesiveness, quadratic group Steiner tree

目　录

插图清单

附表清单

第一章　引言

本章简要介绍本文的研究主题。第1.1节介绍研究背景，第1.2节概括本文的研究内容以及贡献，第1.3节描述后续章节的结构。

1.1　研究背景

知识图谱（Knowledge Graph，简称 KG）是一张用点代表实体，用边代表实体间关系的图 [1]，通常规模巨大，例如著名的 DBpedia[2] 有几百万个实体，谷歌的知识图谱有十亿个实体。知识图谱为结构化数据提供便利，被广泛的应用于增强网络搜索、电子商务、社交网络以及很多其他领域 [3]。图1–1是谷歌知识图谱运用的一个例子，它清晰的提供了关于吴恩达的一些个人信息。

当用户打算分析或使用知识图谱时，常用的办法是借助类似 SPARQL 的形式化查询。但当用户不熟悉形式化查询、不了解具体知识图谱的模式或者查询中包含难以使用形式化查询表达的模糊语义时，机器辅助的知识图谱探查就显得尤为重要。人们设计了分面浏览 [4]，相关推荐 [5] 和举例查询 [6] 等多种有效的方法和工具进行机器辅助的知识图谱探查 [7, 8]，其中大多数直观且受欢迎的用户接口都是基于关键词搜索。

关键词搜索可以让用户不使用特定查询语言，直接查询网络数据。查询的关键词是用户输入的应当与数据匹配的字、词，与关键词相关的数据被抽取出并用适当的形式作为答案呈现给用户，关键词匹配、数据抽取、答案构成与数据的底层结构和查询的内容相关。关键词查询已经在多个数据库上被研究过 [9]，目前在知识图谱上的应用引起了新的关注 [10–13]。

一种常见的知识图谱上的关键词搜索是把每个关键词映射到图中的一个点并抽取出包含这些点的最小权树 [10, 14–22]，这个问题被称为最小权斯坦纳树问题 [23]。更形式化的说，给定一个带边权的图和一个关键词查询，每个关键词匹配到图中的点集（即关键词可以匹配到的所有点），目标是找到一棵覆盖所有关键词的结果树（即每个关键词匹配集中的至少一个点在结果树中）并最小化树的总边权。这个优化问题是著名的组斯坦纳树问题（Group Steiner

图 1-1: 谷歌知识图谱提供的关于吴恩达的信息

Tree，简称 GST）[24]。若给定的图只有点权没有边权，优化目标是最小化树的总点权，则是点权版本的组斯坦纳树问题。

基于 GST 的关键词查询计算量很大，这个问题是 NP-难的。现有的有质量保证的近似算法运行时间过长，无法在大图上运行。文献 [25] 说明现有的算法 [14, 15, 19] 在图版本的 IMDb 上进行一次关键词查询需要花费数千秒，IMDb 只有一百六十万个点和六百一十万条边，显然这些算法在更大的图上（例如 DBpedia）运行时间会更令人难以接受。因此尽管距离 GST 首次被提出已经过去几十年，基于 GST 的搜索算法的速度依然不能满足实际需求，这是开发高效的基于知识图谱的关键词搜索系统时需要面对的重要挑战。

基于 GST 的关键词搜索给边或者点以权重，权越小说明其越重要，并最小化结果树的权重，这假定了用图中重要部分做聚合就能得到一个好的结果。但是，目前的研究正在挑战这个假定 [13, 26]。文献 [13] 分析了从工业知识图谱抽取出的查询结果，认为聚合图中重要部分的做法并不合适，因为这样的结果可能不能提供有意义的信息。文献建议应当提升结果的语义内聚性，语义内聚性

是结果树中实体间的语义距离之和。语义距离的定义应当与图上的距离不同，图上靠得很近的点语义距离也可能很远（例如不同的主题）。文献 [26] 中用户实验汇报了一个相似的结果：用户更喜欢实体更相似的查询结果，尽管这些实体本身并不重要。文献 [13] 中的观点和文献 [26] 中的用户实验给出了改进基于 GST 的关键词搜索的方向，若想进一步提升知识图谱搜索结果的质量，如何形式化和高效解决计算关键词搜索的语义内聚子图就成为了必须要处理的问题。

1.2 研究内容

对于组斯坦纳树问题，本文提出了新的算法计算关键词查询的近似答案，提出的算法兼具质量和效率。在百万个点的图上可以毫秒级得到有质量保证的近似结果。这些算法依赖于中心标注法（Hub Labeling，简称 HL）[27]—一种每个点保存可达点标签集的数据结构，HL 可以高效的计算最短距离和最短路径。本文提出了两种中心标注来实现效率上的巨大提升。

第一个算法 KeyKG 为每个关键词挑选一个匹配点并用一棵树覆盖所有的匹配点。对于 g 个关键词的查询，KeyKG 是 $g-1$ 近似算法，即 KeyKG 算出的树的总边权最多是最优解的 $g-1$ 倍。一方面 g 实际上通常很小，另一方面这个问题有 $O(\ln g)$ 的不可近似性 [24]，即没有多项式时间的算法能够以 $O(\ln g)$ 的近似比处理这个问题，因此 $g-1$ 的近似比可以接受。为了高效的计算最短距离和最短路径，本文对现有的 [28] 进行拓展，提出基于介度中心性的 HL。这个 HL 是离线建立的且与查询无关，所以它是静态的。在大的知识图谱上，使用静态 HL 的 KeyKG 至少比现有最好的算法 [17, 19] 快一个数量级，并能计算出质量相当的好结果。

第二个算法 KeyKG$^+$ 使用一个新的 HL 拓展 KeyKG 。这个 HL 在给定一个具体查询时在线建立，倒转并聚合查询相关的静态标签，因此是动态的。它减少了传统静态 HL 计算点集最短距离时的重复计算，从而加速了 KeyKG 。虽然需要额外的时间在线建立，使用动态 HL 总体上依然能带来几个数量级的加速。特别的，在 DBpedia 上，KeyKG$^+$ 与 KeyKG 计算出一样的结果，但只花费几十毫秒，这说明 KeyKG$^+$ 是一个可以在真实场景下使用的算法。

经典的最小边（点）权斯坦纳树问题旨在最小化子图中的边（点）权重之和。为了结合语义内聚性，本文提出了一个新的目标函数：点权之和（代表显著性）与点对语义距离之和（代表内聚性）的线性组合。语义距离可以使用任

意的伪三角函数计算，并不局限于现有研究中 [26, 29] 语义距离或相似度的实现。考虑到点对语义距离是一个二次项，这个问题称为二次组斯坦纳树问题（Qadratic Group Steiner Tree Problem，简称 QGSTP），这个二次项极大地提升了问题的难度。显然 GST 问题是 QGSTP 一种特例，因此 QGSTP 是 NP-难问题，目前没有已知算法能给出这个问题的近似解。

本文提出了两个基于相关路径集（Relevant Path Set，简称 RPS）的近似算法：近似比为 $2(g-1)^2$ 的算法 **QO** 和近似比为 $(g-1)^2|V|$ 的算法 **EO** 。对于 g 个关键词的查询，一个 RPS 是连接公共根与查询中 g 个不同关键词节点的路径的集合。本文使用动态规划计算某种代价最优的 RPS 并将其路径拼接为有近似比保证的结果。**QO** 与 **EO** 的主要区别在于 **QO** 计算了全局最优的 RPS，而 **EO** 只计算局部最优的 RPS，这导致 **QO** 速度慢、理论近似比小，**EO** 则速度快、理论近似比大。为了进一步加速 **QO** 与 **EO** ，本文提出了几个有效剪枝策略和一个点排序策略。这些策略不会影响结果的近似比，却能使得 **EO** 以数十秒的时间完成百万点级知识图谱上的查询。

本文的贡献总结如下：

- 针对组斯坦纳树问题，本文设计了两个近似算法：KeyKG 和 KeyKG$^+$ ，并证明 KeyKG 与 KeyKG$^+$ 的近似比都是 $g-1$。为了高效在线计算最短距离和最短路径，本文使用一个启发式算法得到新的静态 HL，并提出了查询相关的动态 HL。实验表明提出的算法比现有算法快至少三个数量级，在 DBpedia 上可以达到毫秒级。
- 本文提出了计算语义内聚子图的新问题—二次斯坦纳树问题。针对二次组斯坦纳树问题，本文提出了两个近似算法：**QO** 和 **EO** ，并证明 **QO** 的近似比为 $2(g-1)^2$，**EO** 的近似比为 $(g-1)^2|V|$。对于这两个算法，本文还提出了几个启发式优化，包括三个剪枝动态规划搜索空间的策略和一个尽早寻找更优解的点排序策略。实验表明 **EO** 在百万个点的知识图谱上可以进行秒级的计算。

1.3 文章结构

第2章介绍论文的知识背景以及相公工作。第3章定义组斯坦纳树问题，并介绍 KeyKG 、静态 HL、动态 HL 和 KeyKG$^+$ 。第4章定义二次组斯坦纳树问题，并介绍 **QO** 和 **EO** 。第5章对本文进行总结和展望。

第二章　背景知识和相关工作

本章详细介绍文章涉及的背景知识以及相关工作。第2.1节介绍必要的背景知识，第2.2节介绍图数据上关键词搜索的相关工作，第2.3节介绍距离先知的相关工作。

2.1　背景知识

本节定义必要的术语并形式化需要解决的问题，表2-1中汇总了本文用到的主要记号。

2.1.1　知识图谱

知识图谱是有标签且相互有联系的实体的集合。为了简洁，只形式化知识图谱中与本文相关的部分。知识图谱是一个简单无向图 $G = \langle V, E \rangle$，其中 V 是有限的 n 个点的集合，代表实体，$E \subseteq V \times V$ 是有限的 m 个无序点对的集合，代表实体间的关系。自环、重边和边的方向对本文的方法没有影响，可以忽略。存在权重函数 $\texttt{weight} : E \mapsto \mathbb{R}^{0+}$ 给予每条边一个非负权重，weight 越小边越重要。存在权重函数 $\texttt{wt} : V \mapsto \mathbb{R}^{0+}$ 给予每个节点一个非负权重，wt 越小点越重要。权重函数的具体实现不是本文关注的重点，任何满足上述条件的权重函数都可以被使用。

图2-1是一个知识图谱，它有 7 个点和 9 条边，点上和边上有非零实数作为权重。为了表述方便，所有点和边都用符号表示，实际的知识图谱里这些都应该是有意义的文字。

2.1.2　图相关的术语

一个点的度是与它相连的边的条数。按标准的方式定义路径，路径 p 的长度为 p 中所有边的权重之和，用 $\texttt{len}(p)$ 表示。两个点 $u, v \in V$ 的距离

表 2-1: 主要记号

$G = \langle V, E \rangle$	知识图谱
$Q = \{k_1, \ldots, k_g\}$	（关键词）查询
g	Q 中的关键词数
$K_i = \texttt{hits}(k_i)$	与 k_i 匹配的节点集
$T = \langle V_T, E_T \rangle$	结果树
$T^* = \langle V_{T^*}, E_{T^*} \rangle$	最优结果树
`weight`	边权函数
`wt`	点权函数
`sd`	点对的语义距离
`WT`	组斯坦纳树结果的代价
`cost`	二次组斯坦纳树结果的代价
α	`cost` 中的系数
`getD`	点对最短距离函数
`getSP`	点对最短路径函数
`L`	静态标签集
`bc`	介度中心性
`pred`	最短路的前驱
r	根节点
ρ^*	T^* 的根节点
$\mathbf{P}_r = \{P_1, \ldots, P_g\}$	r-RPS
$P_i = \langle V_{P_i}, E_{P_i} \rangle$	r-K_i 的路径
`pcost`	RPS 的代价
$\mathbf{P}_r^{\min}$	`pcost` 局部最小的 r-RPS
$\mathbf{P}^{\#} = \mathbf{P}_{r^{\#}}^{\min}$	`pcost` 全局最小的 RPS
$T^{\#} = \langle V_{T^{\#}}, E_{T^{\#}} \rangle$	从 $\mathbf{P}^{\#}$ 得到的结果

为 G 中连接 u 与 v 的最短路径的长度，用 $\texttt{dist}(u, v)$ 表示，若无法联通，则 $\texttt{dist}(u, v) = +\infty$。

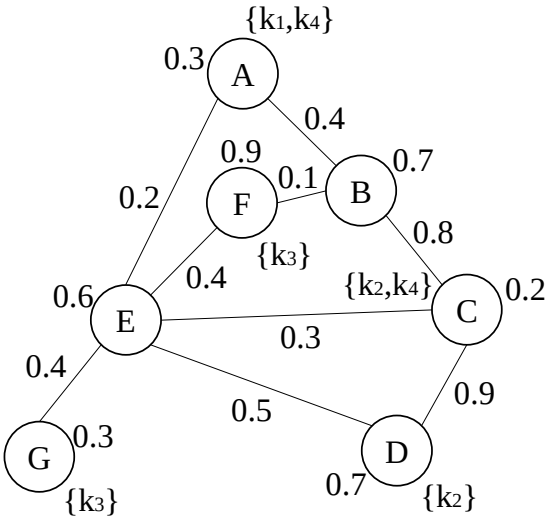


图 2-1: 形式化的知识图谱

2.1.3 关键词匹配

令 $\mathbb{K}$ 为所有关键词的集合。关键词匹配是一个检索函数 $\texttt{hits} : \mathbb{K} \mapsto 2^V$，将关键词映射到 V 的子集。具体的检索函数依赖于知识图谱的实现，与本文的技术并不相关。例如在图2-1中，每个点关联一个关键词集合，依次可定义检索函数：

$$\texttt{hits}(k_1) = \{A\}, \texttt{hits}(k_2) = \{C, D\}, \texttt{hits}(k_3) = \{F, G\}, \texttt{hits}(k_4) = \{C\}.$$

这里的检索函数只把关键词匹配到节点。本文的方法可以直观的通过图细分支持边匹配：边 (u, v) 的细分会产生一个新点 w，并用两条边 (u, w) 和 (w, v) 替换 (u, v)，因此 (u, v) 的边匹配可以被转化为 w 的点匹配。为了简化文章的描述，问题的定义里忽略边匹配。

2.1.4 关键词查询

关键词查询 $Q \subseteq \mathbb{K}$ 是一个有限的关键词集合。给定 g 个关键词 $Q = \{k_1, \ldots, k_g\}$，对于每个 $1 \le i \le g$，缩写 $\texttt{hits}(k_i)$ 为 K_i 并称它们为关键词节点。对于 $G = \langle V, E \rangle$，Q 在 G 上的结果是一棵树 $T = \langle V_T, E_T \rangle$ 使得：

1. $V_T \subseteq V, E_T \subseteq E$ 且 T 是一棵树，
2. V_T 包含每个 K_i 中的至少一个点，即 $V_T \cap K_i \neq \emptyset$，
3. T 最小化某一个代价函数。

条件 (1) 和 (2) 定义了树的范围，条件 (3) 需要树有最小的权重。

2.2 图数据的关键词搜索

2.2.1 匹配式方法

匹配式方法 [11, 12, 30–35] 的主旨是首先把关键词查询转化为一个结构化的查询（例如是资源描述框架（Resource Description Framework，简称 RDF）图上的 SPARQL 查询），然后在图上利用结构化查询借助标准的后端 [31–33, 36–38] 检索答案。这种转化通常借助图的结构，产生结构化查询（例如图模式）进行子图匹配以查找结果。

当关键词查询有具体的搜索意图且能用图模式表示时，匹配式方法是合适的选择。但是很多情况下关键词查询是模糊的，不能用具体的图模式表示，匹配式方法并不合适。

2.2.2 检索式方法

检索式方法通常预先计算并索引很多大小受限的子图，这些子图可作为候选结果（例如 EASE[39] 中的 r-半径图、SAINT[40] 中的元组单元）。关键词搜索被转化成一个传统的信息检索问题—检索并排序所有与查询相关的子图。

检索式方法的可拓展性和高效性是因为极大的限制了搜索空间，也因此牺牲了搜索质量。此外，若查询中的关键词本身不相互靠近（例如它们的距离超出了预先定义的索引子图大小的上界），则检索式方法无法找到任何结果。

2.2.3 抽取式方法

图数据上基于关键词的探查已经被深入研究过，现有的做法把它建模成不同的组合优化问题—抽取图中的一个最优子图，使得其最优化某个评分函数。在文献 [14–18] 中，目标是计算一个最小权 GST，即最小化子树中点权或者边权之和。在文献 [10, 19–22] 中，最优化目标是子树中根到叶子的路径总边权和或者叶子到叶子的路径总边权和，这类变式可以使用双向搜索、基于距离的剪枝和并行化等优化技术。在文献 [41–43] 中，结果子图的点数受限，最优化目标是子图点权之和最大。

上述的所有抽取式方法有一个共同的问题，它们的目标函数都只简单聚合子图中元素的权重，然而重要节点和边的聚合不一定是一个有意义的结果。文献 [26] 已经发现用户更喜欢实体同构的子图。目前还没有文献处理过语义内聚的问题，最相关的工作可能是 [29]，它考虑无权图，旨在最大化子图点对的相似度。但是，文献 [29] 中的启发式解法只能处理基于带重启的随机游走的相似度，不能应用于一般化问题。

2.2.4 基于 GST 的方法

本文的研究方向属于抽取式方法，具体是基于 GST 的方法或者其变式。BANKS [14] 将从关键词节点到公共根的路径拼接成树来近似得到一个最小权 GST。BANKS-II [19] 使用双向搜索来提升 BANKS 的效果。BLINKS [20] 利用预先计算距离和图分割来加速双向搜索。文献 [10] 利用图摘要来剪枝搜索空间。除了这些近似算法，DPBF [15] 基于动态规划找到最小权 GST，PrunedDP++ [17] 利用 A* 搜索优化 DPBF，达到目前精确算法的最优结果。

表 2-2: 基于 GST 的近似算法对比。BANKS，BANKS-II 和 BLINKS 的结果引用自文献 [25]。（n 是点的个数；m 是边的条数；g 是关键词的个数）

算法	近似比	运行时间
BANKS [14]	$O(g)$	$O(n^2 \log n + nm)$
BANKS-II [19]	$O(g)$	$O(n^2 \log n + nm)$
BLINKS [20]	$O(g)$	依赖于划分
KeyKG$^+$	$O(g)$	$O(n^2 g + ng^3)$

由于标准的 GST 是 NP-难问题，DPBF 和 PrunedDP++ 之类的精确算法时间复杂度是指数级的，无法在大图上运行。表2-2汇总了有理论质量保证的多项式时间算法（包含本文提出的 KeyKG$^+$ ）。注意表中只包含有实际应用价值的算法，有些算法理论近似比很小，但是实际运行时间过长，如 2-Star [44] 近似比为 $O(\sqrt{g}\ln g)$，却无法在大图上运行，所以被排除。

2.2.5 斯坦纳树

GST 问题是斯坦纳树问题的推广，在原本的斯坦纳树问题下，就关键词搜索而言，每个关键词仅匹配一个节点。STAR [16]、SketchLS [45] 等解决斯坦纳树问题的算法不能直接应用于 GST 问题。

2.2.6 图搜索的内聚性

图搜索已经有内聚性的定义，文献 [46] 定义内聚性为三角形结构稠密的紧致子图。按文献 [47] 定义，查询会形成关键词组，结果子图中联通组内关键词的路径无法包含其他组内的关键词。

上述关于内聚性的定义（以及它们的算法）与本文的内容有本质的不同，它们关注于结构上的内聚，本文的定义适用于任意的内聚性度量，例如量化的语义内聚性。

2.3 距离先知

2.3.1 中心标注法

中心标注法 [27] 是一种流行的回答点对最短距离的距离先知 [48]。剪枝界标标注法（Pruned Landmark Labeling，简称 PLL）[28] 是 HL 的一种实现，它利用界标剪枝迪杰斯特拉算法，产生的标签集的大小与剪枝搜索的点序有关。原版 PLL 按点度排序，随后的 RXL[49] 使用最短路径树来建立更好的点序，SHP[50] 用重要节点作为界标，重要性是一些启发式指标的组合。

2.3.2 近似距离先知和其他距离先知

有些距离先知被用来计算近似距离 [51–55]，不过由于这样会影响问题的近似比，所以并不适用这里的场景。路网上有更加高效的距离先知 [56–59]，不过知识图谱上并没有路网上的特殊特征，因此这些算法并不适用。

第三章　基于中心标注法的组斯坦纳树近似算法

本章具体介绍组斯坦纳树的近似算法。第3.1节给出问题的定义。第3.2节介绍基于静态中心标注法的近似算法 KeyKG 并提出一个新的静态中心标注 SHL。第3.3节进一步将静态标注转化为动态标注，提出基于动态中心标注法的近似算法 KeyKG$^+$，KeyKG$^+$ 与 KeyKG 近似比一致，但更加高效。第3.4节对提出的两个算法进行实验评估。

3.1　问题定义

问题描述　当 $\mathtt{WT}(T) = \sum_{e \in E_T} \mathtt{weight}(e)$ 是定义第2.1.4节中条件 (3) 的优化目标时，这个问题就是经典的组斯坦纳树问题 [24]。

问题复杂度　计算最小权 GST 是 NP-难问题，有 $O(\ln g)$ 的不可近似性 [24]。本章的目标是得到知识图谱上关键词查询的最小权 GST 的近似结果。

例 3-1　例如，考虑图3-1中 $K_1 = \{B, F\}$，$K_2 = \{E\}$ 和 $K_3 = \{C, D\}$ 的关键词查询 $Q = \{k_1, k_2, k_3\}$，右上角的结果树 T_E 包含了 $B \in K_1$，$E \in K_2$ 和 $C \in K_3$，就是一个可能的结果。

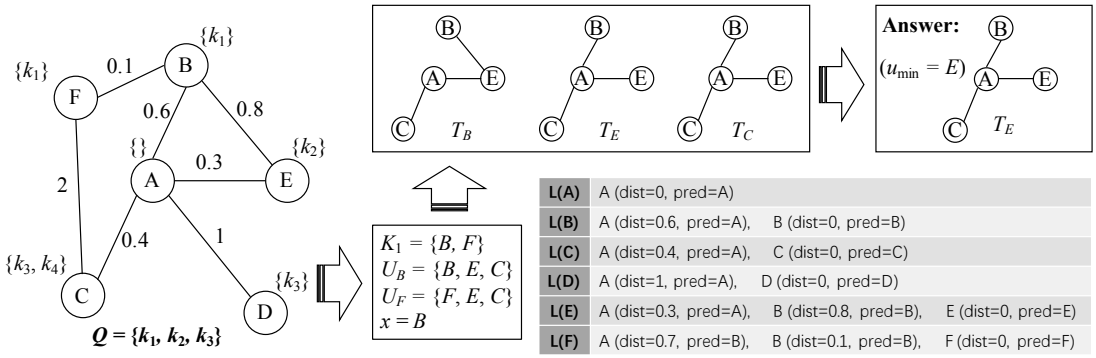


图 3-1: KeyKG 的运行例子

3.2 基于静态中心标注法的近似算法

由于问题是 NP-难的，本节提出第一个近似算法 KeyKG，在第3.2.1节中描述算法并分析其近似比。第3.2.2节给出 KeyKG 的近似比证明。KeyKG 基于两个子过程：计算最短距离的函数 getD 和计算最短路径的函数 getSP。为了支持高效计算 getD 和 getSP，第3.2.3节提出了中心标注法的新实现。最终，在第3.2.4节给出 KeyKG 运行时间的分析。

3.2.1 KeyKG 算法

算法3.1描述了算法 KeyKG，它能找出一个覆盖所有的 g 个关键词的知识图谱上的 GST。KeyKG 首先贪心地选择一个互相靠近的关键词节点的集合，对于每个 $1 \le i \le g$，这个集合包含一个 K_i 中的点，这些被包含的点用 U_x 表示（第 1-8 行）。接着 KeyKG 迭代地拓展最短路，贪心地查找一个 GST 覆盖 U_x，这个 GST 用 $T_{u_{\min}}$ 表示（第 9-18 行）。

具体的，对于每个 $v_1 \in K_1$（第 1 行），KeyKG 查找每个其他 K_i 中距离 v_1 最近的点（第 2-4 行）。令 U_{v_1} 为这些点 v_i 的集合（包含 v_1）并令 W_{v_1} 为从 v_1 到这些点的距离之和（第 5-6 行），则每个 $v_1 \in K_1$ 都有一个对应的 W_{v_1}。令 $x \in K_1$ 是这些点中 W_{v_1} 最小的（第 8 行）。这样对于每个 $1 \le i \le g$，U_x 都包含 K_i 中的一个点，这些点被选择的原因是它们彼此靠近，覆盖所有这些点的 GST 的总权重可能较小。

KeyKG 的剩余部分以每个 $u \in U_x$ 为起点创建了一个 GST T_u 并挑选这些 $|U_x|$ 个 GST 中权重最小的。具体的，每个 T_u 初始化都只有一个点 u（第 9-10 行），接着迭代地添加节点直到 T_u 覆盖了 U_x（第 11 行）。添加节点的方式是选择 T_u 中的一个点 $s_{\min}$ 与 T_u 外的一个点 $t_{\min} \in U_x$，它们之间的距离最小（第 12 行）。$s_{\min}$ 到 $t_{\min}$ 的最短路被添加到 T_u 中（第 13-14 行）。这个用最短路贪心拓展的做法可能产生一个权重小的 GST。每个 $u \in U_x$ 有一个对应的 T_u，令 $u_{\min} \in U_x$ 是对应 T_u 权重最小的节点（第 17 行），KeyKG 将返回 $T_{u_{\min}}$（第 18 行）。

KeyKG 依赖于计算两点间最短距离的 getD（第 3 行和第 12 行）和计算两点间最短路径的 getSP（第 13 行），这两者的细节在第3.2.3节中。

例 3-2 在图3-1中，给定 $Q = \{k_1, k_2, k_3\}$，假定 $K_1 = \{B, F\}$，$K_2 = \{E\}$ 且 $K_3 =$

算法 3.1 KeyKG

输入: 知识图谱 $G = \langle V, E\rangle$, g 个关键词集合 $K_1, \ldots, K_g$
输出: 一个覆盖 $K_1, \ldots, K_g$ 的 G 的 GST

for $v_1 \in K_1$ **do**
  for $i \leftarrow 2$ to g **do**
    $v_i \leftarrow \arg\min_{v \in K_i} \texttt{getD}(v_1, v)$
  end for
  $U_{v_1} \leftarrow \{v_i : 1 \le i \le g\}$
  $W_{v_1} \leftarrow \sum_{i=2}^{g} \texttt{dist}(v_1, v_i)$
end for
$x \leftarrow \arg\min_{v_1 \in K_1} W_{v_1}$
for $u \in U_x$ **do**
  $T_u = \langle V_{T_u}, E_{T_u}\rangle \leftarrow \langle \{u\}, \emptyset\rangle$
  while $U_x \not\subseteq V_{T_u}$ **do**
    $\langle s_{\min}, t_{\min}\rangle \leftarrow \arg\min_{\langle s,t\rangle \in V_{T_u} \times (U_x \setminus V_{T_u})} \texttt{getD}(s, t)$
    $p \leftarrow \texttt{getSP}(s_{\min}, t_{\min})$
    将 p 中的节点与边加入 T_u
  end while
end for
$u_{\min} \leftarrow \arg\min_{u \in U_x} \texttt{WT}(T_u)$
return $T_{u_{\min}}$

$\{C, D\}$。对于 $B \in K_1$，挑选 $E \in K_2$，$C \in K_3$，会产生 $U_B = \{B, E, C\}$。对于 $F \in K_1$，会产生 $U_F = \{F, E, C\}$。由于 $W_B = 1.8$，$W_F = 2$，有 $W_B < W_F$，则 $x = B$。接着对于 $B, E, C \in U_B$，分别产生 T_B、T_E、T_C，可得这三者的权重满足 $\texttt{WT}(T_E) = \texttt{WT}(T_C) < \texttt{WT}(T_B)$。因此 $u_{\min} = E$（或 $u_{\min} = C$），最终将 T_E（或 T_C）返回。

3.2.2 近似比分析

对于包含 g 个关键词的查询，定理 3-1说明 KeyKG 是一个 $g-1$ 近似算法，即得到的 GST 的权重最多是理论最好结果的 $g-1$ 倍。一方面，实际上 g 通常很小，另一方面，这个问题的近似比无法低于 $O(\ln g)$[24]，因此这个近似比是可以接受的。

定理 3-1 对于 $g \ge 2$，KeyKG 是一个 $g-1$ 近似算法。

证明: 令 $T_{\text{opt}} = \langle V_{\text{opt}}, E_{\text{opt}} \rangle$ 为一个最优解，接着证明下面的不等式成立：

$$\texttt{WT}(T_{u_{\min}}) \leq (g-1) \cdot \texttt{WT}(T_{\text{opt}}). \tag{3-1}$$

注意到对于 $1 \leq i \leq g$，存在一个 $o_i \in V_{\text{opt}}$ 满足 $o_i \in K_i$。对于 $2 \leq i \leq g$，令 $p_{1,i}$ 是 T_{opt} 中 o_1 到 o_i 的路径，有 $\texttt{len}(p_{1,i}) \leq \texttt{WT}(T_{\text{opt}})$，因此

$$\sum_{i=2}^{g} \texttt{dist}(o_1, o_i) \leq \sum_{i=2}^{g} \texttt{len}(p_{1,i}) \leq (g-1) \cdot \texttt{WT}(T_{\text{opt}}). \tag{3-2}$$

现在考虑 $u = x$ 时 T_u 的创建（第 10-15 行）。令 $\mathbf{P}$ 为被增加到 T_x 中的所有最短路的集合（第 14 行），有

$$\texttt{WT}(T_{u_{\min}}) \leq \texttt{WT}(T_x) \leq \sum_{p \in \mathbf{P}} \texttt{len}(p). \tag{3-3}$$

对于每个 $p \in \mathbf{P}$，有 $\texttt{len}(p) = \texttt{dist}(s_{\min}, t_{\min})$ (第 13 行)，因此根据 $\langle s_{\min}, t_{\min} \rangle$ 的定义 (第 12 行) 得：

$$\sum_{p \in \mathbf{P}} \texttt{len}(p) \leq \sum_{v \in U_x \setminus \{x\}} \texttt{dist}(x, v) = W_x \leq W_{o_1}, \tag{3-4}$$

此处最后的不等式时由于 x 的定义（第 8 行）。根据 W 的定义（第 6 行）和 U 的定义（第 5 行和第 3 行）得

$$W_{o_1} = \sum_{v \in U_{o_1} \setminus \{o_1\}} \texttt{dist}(o_1, v) \leq \sum_{i=2}^{g} \texttt{dist}(o_1, o_i). \tag{3-5}$$

最后，连列式 (3-3)，式 (3-4)，式 (3-5) 和式 (3-2)，可得式3-1。 □

3.2.3 静态中心标注法

KeyKG 依赖于计算两点间最短距离的 getD 和计算两点间最短路径的 getSP，但在大的知识图谱上，直接在线计算（例如迪杰斯特拉算法）这两个子过程运行时间过长，离线保存任意点对的距离和路径空间占用过大。为了时间与空间的取舍，本算法考虑使用中心标注法 [27]。本文把它称为静态中心标注法，它是离线创建的且不随着查询变化。作为对比，还有动态中心标注法，它是在线创建的且与查询相关，动态 HL 的细节将在第3.3节中介绍。

下面回顾一下静态 HL 的基本思想和它实现 getD 的方法。接着提出创建静态 HL 的新方法并拓展支持 getSP。

基本思想 一个静态 HL 是为图离线创建的索引结构 [27]。对于 $G = \langle V, E\rangle$，静态 HL 可以被看成函数 $\mathtt{L} : V \mapsto 2^V$，把点映射到点集，这个函数需要满足下面的条件：$\forall u, v \in V$，若这两个点在 G 中联通，则 $\exists h \in \mathtt{L}(u) \cap \mathtt{L}(v)$ 使得 h 在 u 到 v 的最短路径上。对于 $v \in V$，$\mathtt{L}(v)$ 被称为 v 的标签集。在标准的 L 索引结构中，每个 $\mathtt{L}(v)$ 是按标识排序的标签列表。对于每个 $h \in \mathtt{L}(v)$，它到 v 的距离 $\mathtt{dist}(v, h)$ 也被预先计算并保存在标签集中。例如，图 3-1中知识图谱的静态 HL 在右下角，现在请忽略图中的 pred。

使用 L，对于 $u, v \in V$，$\mathtt{getD}(u, v)$ 的计算不再需要访问原图：

$$\mathtt{getD}(u,v) = \begin{cases} \min\limits_{h \in \mathtt{L}(u) \cap \mathtt{L}(v)} \mathtt{dist}(u,h) + \mathtt{dist}(v,h) & \mathtt{L}(u) \cap \mathtt{L}(v) \neq \emptyset, \\ \infty & \mathtt{L}(u) \cap \mathtt{L}(v) = \emptyset. \end{cases} \tag{3-6}$$

此处 $\mathtt{dist}(u, h)$ 和 $\mathtt{dist}(v, h)$ 与 h 一起分别被存储在 $\mathtt{L}(u)$ 和 $\mathtt{L}(v)$ 中。

例 3-3 用静态 HL 计算图 3-1中的 $\mathtt{getD}(E, F)$，由于 $\mathtt{L}(E) \cap \mathtt{L}(F) = \{A, B\}$，得

$$\begin{aligned} \mathtt{getD}(E,F) &= \min\{\mathtt{dist}(E,A) + \mathtt{dist}(F,A),\ \mathtt{dist}(E,B) + \mathtt{dist}(F,B)\} \\ &= \min\{0.3 + 0.7,\ 0.8 + 0.1\} = 0.9. \end{aligned}$$

但是，标准的 L 索引结构不能支持高效计算 getSP。

构建索引的优化 根据式 (3-6)，若点上有更少的标签，getD 的在线计算将会更快，但最小化平均标签数是 NP-难问题且有 $O(\log n)$ 的不可近似性 [60]。目前已有很多不同的启发式方法来创建给定图上小的标签集 [48]。在这些方法中，PLL[28] 是最受欢迎的，它基于迪杰斯特拉算法并使用标签集剪枝搜索空间。下面优化 PLL 来得到更小的标签集以加速 getD。

算法 3.2拓展优化了 PLL，建立了一个静态 HL。标准的 PLL 运行 n 次迪杰斯特拉算法（第 3-24 行）并迭代地拓展点标签集（第 4 行和第 12 行）。使用 $\mathtt{L}_i(v)$ 来代表 i 次迭代后 v 的标签集。在第 i 次迭代中，迪杰斯特拉算法以 $v_i \in V$ 为起点（第 7 行)，遍历其它的节点并计算它们到 v_i 的距离，将距离存在数组 d 中（第 14-15 行），接着将 v_i 加入它们的标签集（第 12 行）。有些点 u 可能未被访问因此它的标签就可以被省掉。当 u 到 v_i 的距离可以用建立好的 $\mathtt{L}_{i-1}$ 计算时就可以进行这样的剪枝（第 11 行）。

算法 3.2 建立静态 HL

输入： 知识图谱 $G = \langle V, E \rangle$
输出： G 的静态 HL

```
1:  L_0(v) ← ∅ for all v ∈ V
2:  按 bc 值对 V 递减排序
3:  for i ← 1 to n do
4:      L_i(v) ← L_{i−1}(v) for all v ∈ V
5:      visited[v] ← 0 for all v ∈ V
6:      d[v_i] ← 0 and d[v] ← ∞ for all v ∈ V \ {v_i}
7:      PQ ← 包含点 v_i 的优先队列   //点 v 的优先级设置为 d[v]
8:      while PQ 非空 do
9:          u ← PQ.pull()
10:         visited[u] ← 1
11:         if d[u] < getD(v_i, u) then
12:             L_i(u) ← L_{i−1}(u) ∪ {v_i}  //dist(u, v_i) = d[u]
13:             for w ∈ N(u) s.t. visited[w] = 0 do
14:                 if d[u] + weight((u, w)) < d[w] then
15:                     d[w] ← d[u] + weight((u, w))
16:                     pred(w, v_i) ← u  //for L_i(w)
17:                 end if
18:                 if w ∉ PQ then
19:                     PQ.insert(w)
20:                 end if
21:             end for
22:         end if
23:     end while
24: end for
25: return L_n
```

剪枝的正确性证明在文献 [28] 中，本节优化的目的是剪枝更多的标签。期望的情况是在早期的循环中能完成更多点对的距离查询，则后面的循环里剪枝就更有效。直观上，这就是选择穿过很多最短路径的点作为迪杰斯特拉的起点。PLL 的原版实现里按度递减的顺序进行迪杰斯特拉，这是因为度大的点可能出现在很多点对的最短路中。本节中做法不同：点按照介度中心性递减排序（第 2 行）。点的介度中心性定义为

$$\mathsf{bc}(v) = \sum_{s,t \in V \setminus \{v\}} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{3-7}$$

此处 σ_{st} 是 s 到 t 的最短路的数目，$\sigma_{st}(v)$ 是 σ_{st} 中穿过 v 的路径数。使用 Brandes 算法 [61] 计算精确介度中心性在大图上运行时间过长，因此，这里使用源采

算法 3.3 getSP 的实现

输入: 知识图谱 $G = \langle V, E \rangle$ 和两个点 $u, v \in V$
输出: G 中 u 和 v 的最短路径

```
1:  h_min ← arg min_{h∈L(u)∩L(v)} dist(u,h) + dist(v,h)
2:  p ← 只包含点 u 的路径
3:  y ← u
4:  while y ≠ h_min do
5:      在 L(y) 中查找 h_min
6:      将 y 与 pred(y, h_min) 之间的边加入 p 中
7:      y ← pred(y, h_min)
8:  end while
9:  y ← v
10: while y ≠ h_min do
11:     在 L(y) 中查找 h_min
12:     将 y 与 pred(y, h_min) 之间的边加入 p 中
13:     y ← pred(y, h_min)
14: end while
15: return p
```

样算法选择度最高的节点作为枢纽。关于源采样算法和枢纽的定义来自于文献[62]。

索引结构的拓展 为了支持高效计算 getSP，需要拓展 L 的索引结构。在算法3.2中，每个 $v_i \in \mathtt{L}(w)$ 不仅存储 $\mathtt{dist}(w, v_i)$，还存储以 v_i 为根的最短路搜索树中 w 的前驱，用 $\mathtt{pred}(w, v_i)$（第 16 行）表示。保存 pred 不会增长静态 HL 的渐进空间复杂度。

基于拓展后的标签，算法3.3实现了 getSP。为了计算 u 和 v 之间的最短路径 p，首先找到它们在 p 上的中心点 $h_{\min}$（第 1 行），接着不断的沿着前驱重建路径 p 上的 u-$h_{\min}$（第 2-8 行）和 v-$h_{\min}$（第 9-14 行）。

例 3-4 计算图3-1上的 $\mathtt{getSP}(D, F)$，首先检索到 $h_{\min} = A$，路径 p 上 D-A 的部分是一条边 (D, A)，路径 p 上 F-A 的部分是两条边 (F, B) 和 (B, A)，计算如下：$\mathtt{L}(F)$ 中有 $\mathtt{pred}(F, A) = B$，$\mathtt{L}(B)$ 中有 $\mathtt{pred}(B, A) = A$。最终将两条路径拼接得到 $p = D$-A-B-F。

3.2.4 运行时间分析

本节分析 KeyKG 的运行时间。令 $t_{\texttt{getD}}$ 和 $t_{\texttt{getSP}}$ 分别代表 getD 和 getSP 的运行时间，对于每个 $1 \le i \le g$，$|K_i| \le n$，第 1-8 行的时间复杂度为 $O(n^2 g t_{\texttt{getD}})$。第 9-18 行，使用如下的技巧优化时间：对于每个 $U_x \setminus V_{T_u}$ 中的点，保存从 T_u 到它的最短距离，当 p 中的一个点被加入到 T_u 中（第 14 行），调用 getD 更新到 $U_x \setminus V_{T_u}$ 中每个点的距离。用这些距离来查找 $\langle s_{\min}, t_{\min} \rangle$（第 12 行），就不需要在此处调用 getD。尽管这个技巧需要额外的 $O(g^3)$ 的时间来找到最小的距离，但可以把 getD 的调用次数从 $O(ng^3)$ 降到 $O(ng^2)$。因此 KeyKG 总的时间为

$$O(n^2 g t_{\texttt{getD}} + n g^2 t_{\texttt{getD}} + g^3 + g^2 t_{\texttt{getSP}}). \tag{3-8}$$

分析 $t_{\texttt{getD}}$，考虑到式 (3-6) 可以使用类似归并排序的操作，$\texttt{getD}(u, v)$ 的时间复杂度为 $O(|\texttt{L}(u)| + |\texttt{L}(v)|)$，因此 $t_{\texttt{getD}}$ 是 $O(n)$。这比直接使用迪杰斯特拉快的多。此外，由于实际上 $|\texttt{L}(\cdot)| \ll n$，L 的总大小接近 $O(n)$，远比直接保存所有点对的距离小。因此，静态 HL 给出了时间与空间的一个不错的折中。

分析 $t_{\texttt{getSP}}$，注意到可用类似归并排序的操作 $O(n)$ 时间找到 $h_{\min}$（第 1 行），通过在标签集上进行二分查找，$h_{\min}$ 可以 $O(\log n)$ 的时间被定位到（第 5 行和第 11 行），又 while 循环运行 $O(n)$ 次，因此 $t_{\texttt{getSP}}$ 是 $O(n \log n)$。

现在可以总结结论。在式 (3-8) 中，将 $t_{\texttt{getD}}$ 与 $t_{\texttt{getSP}}$ 分别替换为 $O(n)$ 和 $O(n \log n)$，可得到 KeyKG 如下的运行时间：

$$O(n^3 g + n^2 g^2 + g^3 + g^2 n \log n). \tag{3-9}$$

不过这个时间复杂度实际上几乎不会达到。通常 $|K_i| \ll n$，$|V_{T_u}| \ll n$，$|\texttt{L}(\cdot)| \ll n$。此外，$g$ 通常也很小，所以如实验中显示的，KeyKG 实际耗时很少。

3.3 基于动态中心标注法的近似算法

本节提出第二个算法 KeyKG$^+$，它用一个新的在线建立的 HL 拓展 KeyKG 来进一步加速点集距离的计算。第3.3.1节具体描述动态 HL。基于动态 HL，第3.3.2节描述 KeyKG$^+$。第3.3.3节给出 KeyKG$^+$ 的近似比证明。最后，第3.3.4节分析它的运行时间。

3.3.1 动态中心标注法

在算法3.1中，每个 K_i $(2 \le i \le g)$，K_i 中所有点的静态标签都需要通过 `getD` 被访问 $O(gn)$ 次（第 3 行）：执行类似归并排序的操作找到某个点 $v_i \in K_i$ 的静态标签，使得它在 $\mathrm{L}(v_1)$ 中且最小化 v_1 到 K_i 的距离。这些重复的操作可以使用倒排索引结构动态 HL 来优化。动态 HL 把 K_i 中所有点的静态标签集合并。这样的动态 HL 与查询相关，因此需要在线创建。

具体的，动态 HL 是 $(g-1)\times n$ 的矩阵 M。行代表关键词点集 $K_2,\ldots,K_g$，列代表中心点。M 的第 $(i-1)$ 行，M_{i-1}，倒转并合并 K_i 中点的静态标签集。M_{i-1} 的第 j 个元素，用 $M_{i-1,j}$ 表示。当点 $h_j \in V$ 在 K_i 中某个点的静态标签集中时，$M_{i-1,j}$ 非空。$M_{i-1,j}$ 为 K_i 中的一个点，它的静态标签集包含 h_j，且到 h_j 的距离最小

$$M_{i-1,j} = \begin{cases} \underset{u\in K_i \text{ s.t. } h_j\in \mathrm{L}(u)}{\arg\min} \ \mathtt{dist}(u,h_j) & h_j \in \bigcup_{u\in K_i} \mathrm{L}(u), \\ \text{null} & h_j \notin \bigcup_{u\in K_i} \mathrm{L}(u). \end{cases} \tag{3-10}$$

若 h_j 不在任意 K_i 内点的静态标签集中，则令 $M_{i-1,j}$ 为空。使用二维数组保存 M，因此可以常数时间随机访问任意位置。对于非空的 $M_{i-1,j}$，到 h_j 的距离也被保存下来。M_{i-1} 可以直接从 K_i 与 `L` 得到而不需要访问原来的图，如第3.3.2节所示，在 v_i 的计算中，M_{i-1} 可以替换 K_i 内点的静态标签集并借助其紧致性和随机访问能力提升计算效率。

表 3-1: 例子的动态 HL(M)，、$M_{i,j}$ 的下表代表 $\mathtt{dist}(M_{i,j},h_j)$.

	A	B	C	D	E	F
K_2	$E_{(0.3)}$	$E_{(0.8)}$	null	null	$E_{(0)}$	null
K_3	$C_{(0.4)}$	null	$C_{(0)}$	$D_{(0)}$	null	null

例 3-5 表3-1展示了图3-1中的动态 HL M。考虑 $i=3$，由于 $K_3=\{C,D\}$，有

$$\mathrm{L}(C)\cup\mathrm{L}(D)=\{A,C\}\cup\{A,D\}=\{A,C,D\}. \tag{3-11}$$

因此，在第二行只有这三个条目非空。这里举例说明 $M_{2,A}=C$ 的计算过程：$h_j=A$，且 $A\in\mathrm{L}(C)$，$A\in\mathrm{L}(D)$，由于 $\mathtt{dist}(C,A)<\mathtt{dist}(D,A)$，$M_{2,A}$ 选择 C 而不是 D。注意 $\mathtt{dist}(C,A)$ 和 $\mathtt{dist}(D,A)$ 都是通过图3-1中的静态 HL 检索到的。最后，在动态 HL 的 $M_{2,A}$ 条目中填上 $\mathtt{dist}(C,A)=0.4$。

算法 3.4 KeyKG$^+$

输入: 知识图谱 $G = \langle V, E\rangle$, g 个关键词集合 $K_1, \ldots, K_g$
输出: 一个覆盖 $K_1, \ldots, K_g$ 的 G 的 GST

1: **for** $i \leftarrow 2$ to g **do**
2: 创建 M_{i-1}
3: **end for**
4: **for** $v_1 \in K_1$ **do**
5: **for** $i \leftarrow 2$ to g **do**
6: $v_i = \underset{M_{i-1,j} \text{ s.t. } h_j \in \mathrm{L}(v_1) \text{and } M_{i-1,j} \neq \text{null}}{\arg\min} \texttt{dist}(v_1, h_j) + \texttt{dist}(M_{i-1,j}, h_j)$
7: **end for**
8: $U_{v_1} \leftarrow \{v_i : 1 \le i \le g\}$
9: $W_{v_1} \leftarrow \sum_{i=2}^{g} \texttt{dist}(v_1, v_i)$
10: **end for**
11: $x \leftarrow \underset{v_1 \in K_1}{\arg\min} W_{v_1}$
12: **for** $u \in U_x$ **do**
13: $T_u = \langle V_{T_u}, E_{T_u}\rangle \leftarrow \langle \{u\}, \emptyset\rangle$
14: 为 V_{T_u} 创建 M'_u
15: **while** $U_x \nsubseteq V_{T_u}$ **do**
16: **for** $t_i \in (U_x \setminus V_{T_u})$ **do**
17: $s_i = \underset{\substack{M'_{u,j} \text{ s.t. } h_j \in \mathrm{L}(t_i) \\ \text{and } M'_{u,j} \neq \text{null}}}{\arg\min} \texttt{dist}(t_i, h_j) + \texttt{dist}(M'_{u,j}, h_j)$
18: **end for**
19: $\langle s_{\min}, t_{\min}\rangle \leftarrow \underset{\langle s_i, t_i\rangle}{\arg\min} \texttt{dist}(s_i, t_i)$
20: $p \leftarrow \texttt{getSP}(s_{\min}, t_{\min})$
21: 将 p 中的节点和边加入 T_u
22: 更新 M'_u
23: **end while**
24: **end for**
25: $u_{\min} \leftarrow \underset{u \in U_x}{\arg\min} \texttt{WT}(T_u)$
26: **return** $T_{u_{\min}}$

3.3.2 KeyKG$^+$ 算法

算法3.4是基于 KeyKG 的拓展算法 KeyKG$^+$ 。KeyKG$^+$ 在两处创建和使用动态 HL 来提升总体的运行效率但不会改变计算结果。

第一处，对关键词节点 $K_2, \ldots, K_g$ 创建 M（第 1-3 行）。接着使用 M_{i-1} 来查找 v_i（第 6 行）：对于每个 $h_j \in \mathrm{L}(v_1)$，从 $\mathrm{L}(v_1)$ 检索到 $\texttt{dist}(v_1, h_j)$ 并从 M_{i-1}

的 $M_{i-1,j}$ 检索到 $\texttt{dist}(M_{i-1,j}, h_j)$，若 $M_{i-1,j}$ 非空，则计算

$$\texttt{dist}(v_1, h_j) + \texttt{dist}(M_{i-1,j}, h_j), \tag{3-12}$$

这个值代表 v_1 到 K_i 所有通过 h_j 的路径的最短距离。最终 v_i 通过查找所有的 $h_j \in \mathrm{L}(v_1)$ 得到：

$$v_i = \mathop{\arg\min}_{\substack{M_{i-1,j} \text{ s.t. } h_j \in \mathrm{L}(v_1) \\ \text{and } M_{i-1,j} \neq \text{null}}} \texttt{dist}(v_1, h_j) + \texttt{dist}(M_{i-1,j}, h_j). \tag{3-13}$$

KeyKG$^+$ 中 v_i 的计算（第 6 行）与 KeyKG 的计算（第 3 行）是等价的，但是效率更高。

例 3-6 在图 3-1中，当 $B \in K_1$ 是 v_1 时，由于 $\mathrm{L}(B) = \{A, B\}$，表 3-1中 $M_{1,A} = M_{1,B} = E$，挑选 $E \in K_2$ 为 v_2；$M_{2,A} = C$ 且 $M_{2,B} = \text{null}$，挑选 $C \in K_3$ 为 v_3。当 $F \in K_1$ 是 v_1 时，由于 $\mathrm{L}(F) = \{A, B, F\}$，$M_{1,A} = M_{1,B} = E$ 且 $M_{1,F} = \text{null}$，挑选 $E \in K_2$ 为 v_2；$M_{2,A} = C$ 且 $M_{2,B} = M_{2,F} = \text{null}$，挑选 $C \in K_3$ 为 v_3。这些选择与 KeyKG 中选择一致。

在第二处，如同对 K_i 创建 M_{i-1}，对 V_{T_u} 也创建 M'_u（第 14 行）。当 T_u 在循环中加入点时（第 21 行），使用被加入点的静态标签更新 M'_u（第 22 行）。对于每个 $t_i \in (U_x \setminus V_{T_u})$，使用 M'_u 查找与 t_i 距离最近的 $s_i \in V_{T_u}$（第 16-18 行），类似于使用 M_{i-1} 来查找 v_i。最终通过查找所有的 $\langle s_i, t_i \rangle$ 找到 $\langle s_{\min}, t_{\min} \rangle$（第 19 行）。KeyKG$^+$ 中的 $\langle s_{\min}, t_{\min} \rangle$ 的计算（第 16-19 行）与 KeyKG 的计算（第 11-12 行）是等价的，但是效率更高。

3.3.3 近似比分析

KeyKG$^+$ 与 KeyKG 计算出一样的结果，因此也是 $(g-1)$ 近似算法。

评注 3-1 KeyKG$^+$ 与 1-Star [63] 有一些相似的技术。这两个算法都依赖于最短距离和最短路径，有相同的近似比。但是 KeyKG$^+$ 设计的更细致，实际表现会更好。例如，在覆盖所有关键词节点时，1-Star 只是把一个节点到所有其他节点的路径直接拼接，而 KeyKG$^+$ 贪心地构造出一个最小权树。由于提出的静态与动态 HL 能够快速的计算最短距离与最短路径，KeyKG$^+$ 比 1-Star 快的多。

3.3.4 运行时间分析

与 KeyKG 第 1-4 行的 $O(n^3g)$ 相比，KeyKG$^+$ 创建 M_{i-1} 需要 $O(n^2)$ 的时间，查找 v_i 需要 $O(n)$ 的时间—基于数组的 M 支持 $O(1)$ 时间的随机访问。这把 KeyKG 第 1-7 行的时间缩减到 $O(n^2g)$。类似的，KeyKG 第 9-12 行需要 $O(n^2g^2 + g^3)$，KeyKG$^+$ 这部分（第 12-19 行，第 22 行）需要 $O(n^2g + ng^3)$，$ng^3 \ll n^2g$，因此 $O(n^2g + ng^3)$ 接近于 $O(n^2g)$，这个值优于 $O(n^2g^2 + g^3)$。

此外，再次考虑 KeyKG$^+$ 中 getSP 的运行时间（第 20 行）。在算法 3.3中，首先，KeyKG$^+$ 里 $h_{\min}$ 直接被查找到（第 17 行），KeyKG 原本检索 $h_{\min}$ 的 $O(n)$ 被降到了 0。其次，在 T_u 的创建过程中，二分搜索静态标签的次数不再是 $O(gn)$ 而是与 $|E_{T_u}| \in O(n)$ 成正比。因此，getSP 总的运行时间被从 KeyKG 的 $O(g^2n\log n)$ 降到 $O(gn\log n)$。

总的来说，KeyKG$^+$ 的运行时间为：

$$O(n^2g + n^2g + ng^3 + gn\log n), \quad \text{i.e.,} \quad O(n^2g + ng^3). \tag{3-14}$$

类似于第 3.2.4节中关于 KeyKG 运行时间的讨论，这个最坏时间复杂度实际上几乎不能达到。KeyKG$^+$ 的实际运行时间很低，实验显示比 KeyKG 快几个数量级。

3.4 实验评估

本实验的目的是验证以下的研究假设（Research Hypothesis，简称 RH）：

RH1 本章提出的方法在典型的知识图谱上速度很快且远超目前最好的方法 [17, 19]，同时能提供质量相当的好答案。

RH2 提出的静态 HL 比现有的 HL[28, 49, 50] 占用的空间更小，因此能更高效的计算最短距离和最短路径。

RH3 使用动态 HL 可以极大的提升总体的运行效率。

所有的实验都运行在基于 3.5GHz 的至强处理器和 32GB 内存的 Jvm 上。

知识图谱 实验中使用三种不同大小的知识图谱：

- MONDIAL，[①] 一个小型知识图谱。

[①] www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.rdf

表 3-2: 知识图谱和相应的查询

	知识图谱		关键词查询	
	$\|V\|$	$\|E\|$	数量	g
MONDIAL	37,303	109,577	40	1–4
LinkedMDB	748,593	1,216,325	200	1–10
DBpedia	5,765,042	17,557,947	429	1–10

- LinkedMDB，[①] 一个中型知识图谱。
- DBpedia，[②] 一个大型知识图谱。

表 3-2汇总了这些图的大小。所有的知识图谱都载入到内存中。

关键词查询 之前的实验验证 [25] 使用了 MONDIAL，本实验复用了该文献给出的 40 个关键词查询。对于 LinkedMDB，从 WikiMovies[64] 中随机采样了 200 个问句并将自然语言问题中的停用词和标点符号删除，转化成关键词查询。对于 DBpedia，本实验复用 DBpedia-Entity v2[65] 给出了 429 个查询。表 3-2展示了这些查询的数量和大小。

关键词匹配 这三张知识图谱都是 RDF 格式。本实验中的检索函数 `hits` 把关键词 k 映射到可读标签（如 `rdfs:label`）包含 k 的点。查询中的每个关键词至少与最大连通分支中的一个点匹配，因此保证 GST 结果树存在。

边权 没有标准的方式定义知识图谱的边权，本实验中每条边等概率随机的采样一个 $(0, 1000)$ 中的实数。

3.4.1 实验设置

在基于 GST 或者类似 GST 的知识图谱关键词搜索的现有方法中，挑选两个代表性的算法：

- **PrunedDP++**[17] 是 GST 问题目前最好的精确算法，它基于 A* 搜索。
- **BANKS-II**[19] 是 GST 问题目前最好的近似算法，它基于双向搜索。

本实验中重新实现了这两个算法。

实验没有比较 DPBF[15] 和 BANKS[14]，PrunedDP++ 和 BANKS-II 分别是这两者的优化算法。实验中没有 BLINKS[20]，因为它不能处理类似于

[①] www.cs.toronto.edu/~oktie/linkedmdb/linkedmdb-latest-dump.zip
[②] downloads.dbpedia.org/2016-10/core-i18n/en/mappingbased_objects_en.tql.bz2

LinkedMDB 和 DBpedia[25] 这样的大图。实验中没有非 GST 问题的算法，这些方法不能直接与本章提出的算法比较，例如 STAR [16] 是斯坦纳树问题的算法，它假定检索函数是单射函数。

本章提出的静态 HL 按照介度中心性排序，优化了 PLL [28]。实验包含现有的 3 种最好的 HL：

- **PLL** 是 PLL [28] 原版算法，按点度排序。
- **RXL** [49] 基于最短路径树优化 PLL。
- **SHP** [50] 基于重要路径优化 PLL。

本文从文献 [50] 的作者处得到这些算法的实现。

3.4.2 算法实现

本节给出 KeyKG 与 KeyKG$^+$ 的设置细节，同时实现它们的几个变式来全面地分析算法中的关键部分。

KeyKG 与 KeyKG$^+$ 的设置 KeyKG 与 KeyKG$^+$ 的静态 HL 都保存在内存中。在 KeyKG$^+$ 中，动态 HL 是在线建立并保存至内存中的。在建立静态 HL 时，使用 200 个枢纽来近似计算介度中心性 [62]。MONDIAL，LinkedMDB 和 DBpedia 的静态 HL 分别占用 37MB，183MB 和 7,704MB 内存。

算法变式

- KeyKG$^+$-**D** KeyKG$^+$ 的变式，用 MySQL 把静态 HL 保存在硬盘中。这有助于展示算法在内存受限场景下的表现。
- KeyKG-**PLL** KeyKG 的变式，用 PLL 替换本章提出的静态 HL。这将有助于评估基于介度中心性的启发式的效果。
- **SHL-10** 与 **SHL-100** 静态 HL **SHL** 的变式，分别使用 10 个枢纽和 100 个枢纽而非 200 个。**SHL** 的默认版本为 **SHL-200**。随着枢纽数目的增加，介度中心性会被更好的近似，但消耗的时间也会增加。这两个变式有助于展示枢纽个数对于静态 HL 的影响。

3.4.3 评价指标

对于知识图谱上关键词搜索，本章中的方法和两个基线算法都基于 GST，它们的目标是尽可能快的计算或者近似计算一个最小权 GST。因此，按照评价优化算法的标准方式，实验中度量近似的质量和运行时间。使用计算出的 GST

表 3-3: 成功查询的数目（SC）、超时查询的数目（TO）、查询的平均时间（Time/毫秒）

	MONDIAL			LinkedMDB			DBpedia		
	SC	TO	Time	SC	TO	Time	SC	TO	Time
PrunedDP++	40	0	75.65	194	6	357,274.93	413	16	379,250.20
BANKS-II	40	0	418.85	200	0	27,168.20	429	0	647,041.79
KeyKG$^+$	40	0	0.06	200	0	0.16	429	0	82.84
KeyKG$^+$-D	40	0	6.53	200	0	203.35	429	0	29,776.04
KeyKG	40	0	0.08	200	0	0.46	429	0	10,474.65
KeyKG-PLL	40	0	0.09	200	0	0.52	429	0	11,345.97

除以最小权 GST 得到近似比来比较近似的质量，这里不使用例如准确率、召回率、F1 等信息检索的度量，这些度量依赖于具体的边权定义，这与本章的内容无关。

用点标签集的平均大小比较静态 HL。根据式 (3-6) 和算法 3.3，更小的标签集将会使得距离与最短路的计算更快。

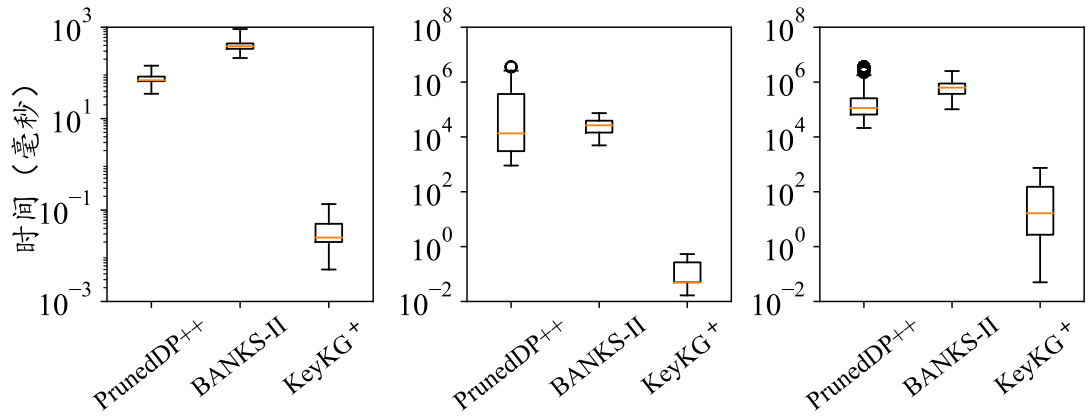
3.4.4 KeyKG 和 KeyKG$^+$ 的高效性（RH1）

为了展示本章算法的效率，用 KeyKG$^+$ 与 PrunedDP++ 和 BANKS-II 作比较。表 3-3总结了每个算法响应每个查询的平均时间。只有 PrunedDP++ 有超时（>1h）发生，此外，这个算法偶尔还会用尽内存（>32GB）。将 PrunedDP++ 超时或者用尽内存时找到的 GST 作为最优结果。

如表 3-3所示，所有的三个算法都能 1 秒以内完成 MONDIAL 上的一个查询。但是在更大的 LinkedMDB 和 DBpedia 上，PrunedDP 和 BANKS 需要几百秒。而 KeyKG$^+$ 在 LinkedMDB 耗时小于 1ms，DBpedia 耗时小于 100ms，比基线快 3 个数量级。

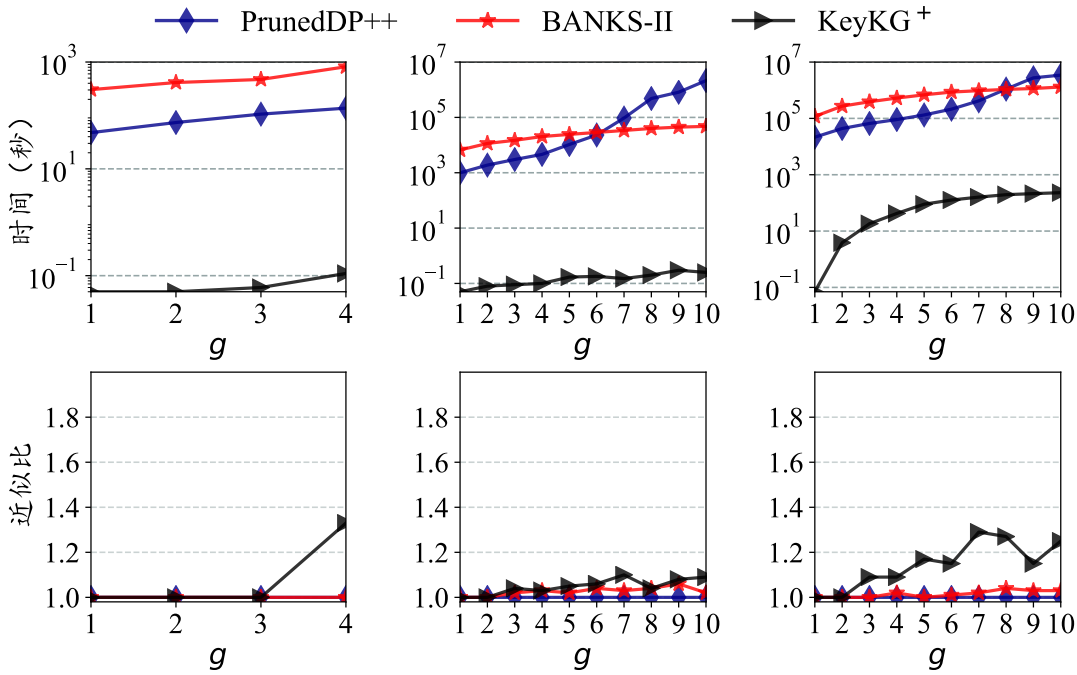
图 3-2用箱型图画出了所有查询的运行时间分布。KeyKG$^+$ 完成任意查询都不超过 1s，它最坏的表现至少比基线最好的表现快一个数量级。由于 PrunedDP++ 的运行时间是指数级的，它在图中出现了很多的离群值。

运行时间与查询中的关键词数目相关（g）。图 3-3的上部展示了不同 g 下的平均运行时间。总体来说，随着 g 的增长，耗时也在增长。KeyKG$^+$ 的运行时间在缓慢增加，说明它对于 g 的拓展性较好。作为对比，PrunedDP++ 的运行时间理论上随着 g 的增长指数级增长，且实际上也增长的很快。图 3-3的底



(a) MONDIAL (b) LinkedMDB (c) DBpedia

图 3-2: 查询运行时间的分布



(a) MONDIAL (b) LinkedMDB (c) DBpedia

图 3-3: 一个查询的平均时间和平均近似比，自变量为关键词的个数 (g)

部展示了查询的平均近似比。对于精确算法 PrunedDP++，它的近似比为 1.0。BANKS-II 和 KeyKG+ 计算出的结果在 LinkedMDB 上近似比低于 1.2，在其他知识图谱上低于 1.4，都能找到足够好的结果，不过 BANKS-II 比 KeyKG+ 的结果略好。它们的近似比随着 g 的增长都缓慢增长，与它们的理论近似比 $O(g)$ 一致。

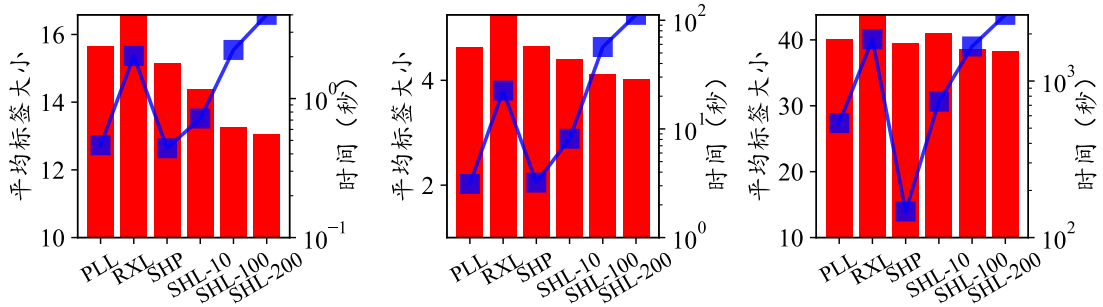
PrunedDP++ 和 BANKS-II 能产生最好或接近最好的 GST，它们的运行时间

却难以接受。KeyKG⁺ 达到了结果质量与运行时间的更好平衡，任何查询都不会超过 1s 并能找到接近最优的 GST。因此，以上的实验结果支撑了 RH1。

此外，在表 3-3中，将静态 HL 保存在硬盘中的 KeyKG⁺-D 只需要不到 1s 就可回答 MONDIAL 与 LinkedMDB 上的查询，证明 KeyKG⁺-D 有可能在内存受限的场景下工作。即使在这样的场景下，KeyKG⁺-D 在所有的知识图谱上也比基线算法快至少一个数量级。

3.4.5 新型静态 HL 的有效性（RH2）

为了展示本章提出的 SHL 的有效性，需要将 SHL 与 PLL、RXL 和 SHP 做比较。图 3-4中的条形图为点标签集的平均大小。SHL-200 在三张知识图谱上都建立了最小的 HL，与 PLL 和 RXL 相比，SHL-200 在 MONDIAL 上减少了 17%–21%，在 LinkedMDB 上减少了 13%–23%，在 DBpedia 上减少了 5%–13%。以上结果证明了基于介度中心性的启发式的有效性。SHL-200 也超越了 SHP，在不同的图上减少了 3%–14%。因此，以上的结果支撑了 RH2。



(a) MONDIAL (b) LinkedMDB (c) DBpedia

图 3-4: 点标签集的平均数量（条形图）；建立静态 HL 的平均时间（折线图）

表 3-3还说明了 SHL 对于查询时间的影响：在所有的三种知识图谱上，使用 SHL 的 KeyKG 的耗时都比使用 PLL 的 KeyKG-PLL 更小，在最大的 DBpedia 上，运行时间减少了 8%。

图 3-4中的折线图为创建静态 HL 的时间。对比 SHL-10，SHL-100 和 SHL-200，当增加枢纽的数量时创建时间变长，创建的 HL 更小。尽管 SHL-200 花费了最多的时间创建静态 HL，但这个创建的过程是离线的，不会对查询产生影响。实际上，在 MONDIAL，LinkedMDB 和 DBpedia 上，离线创建分别耗时 4s，112s 和 2,647s，这个时间是可以接受的。

3.4.6 动态 HL 的有效性（RH3）

为了展示动态 HL 的效果，需要用 KeyKG$^+$ 与 KeyKG 比较。如表 3-3所示，在 MONDIAL 和 LinkedMDB 上，两个算法都能在 1ms 以内完成一个查询。但在最大的 DBpedia 上，KeyKG$^+$ 依然能毫秒级完成查询，KeyKG 却需要花费超过 10s。当每个关键词都匹配到很多节点时，KeyKG 比 KeyKG$^+$ 慢很多，这是因为静态 HL 需要匹配数平方级次的 getD 调用。使用动态 HL，理论上这个部分会比动态 HL 快一个数量级，实验中也显示这个提升很显著。因此，实验结果支撑了 RH3。

此外，在表 3-3中，即使使用静态 HL，KeyKG 也在所有的知识图谱上比基线算法快至少一个数量级，证明基于 HL 的算法的优越性。

第四章　基于相关路径集的二次组斯坦纳树近似算法

本章具体介绍二次组斯坦纳树的近似算法。第4.1节给出问题的定义。第4.2节介绍近似算法的核心定义 RPS 以及与其相关的一些结论。第4.3节介绍基于 RPS 的质量导向算法 **QO** 。针对 **QO** 运行效率不佳的问题，第4.4节提出以质量换取效率的效率导向算法 **EO** 。第4.5节提出启发式优化以提高算法的运行效率，这些优化对 **QO** 与 **EO** 都适用。最后第4.6节进行实验评估。

4.1　问题定义

问题描述　对于每张图 $G = \langle V, E\rangle$，存在一个语义距离函数赋予每个点对一个非负实数，用 $\mathtt{sd} : V \times V \mapsto \mathbb{R}_{\geq 0}$ 表示。这个函数 $\mathtt{sd}$ 满足 $\forall u, v, w \in V$：

- 同一性：$\mathtt{sd}(v, v) = 0$
- 对称性：$\mathtt{sd}(u, v) = \mathtt{sd}(v, u)$
- 直递性：$\mathtt{sd}(u, v) \leq \mathtt{sd}(u, w) + \mathtt{sd}(w, v)$

语义距离的计算可以独立于图的结构以及点权，它与图中的距离（最短路径）不同，例如在图中相邻的点之间的语义距离可以很远。结果树 $T = \langle V_T, E_T\rangle$ 的代价是它的点权与点对语义距离之和：

$$\mathtt{cost}(T) = \alpha \sum_{v \in V_T} \mathtt{wt}(v) + (1 - \alpha) \sum_{\substack{v_i, v_j \in V_T \\ i<j}} \mathtt{sd}(v_i, v_j)\,, \tag{4-1}$$

此处的 $\alpha \in [0, 1]$ 是一个可变参数。在式 (4-1) 中，第一部分代表树中节点的显著性 (salience)，第二部分描述了树中节点的语义内聚性 (cohesiveness)。$\mathtt{wt}$ 和 $\mathtt{sd}$ 的计算与本文提出的方法是独立的，式 (4-1) 中可以使用任意的 $\mathtt{wt}$ 和 $\mathtt{sd}$。本文在实验中使用了已有的计算方法 $\mathtt{wt}$ 和 $\mathtt{sd}$。

当式 (4-1) 是定义第2.1.4节中条件 (3) 的优化目标时，这个问题被称为二

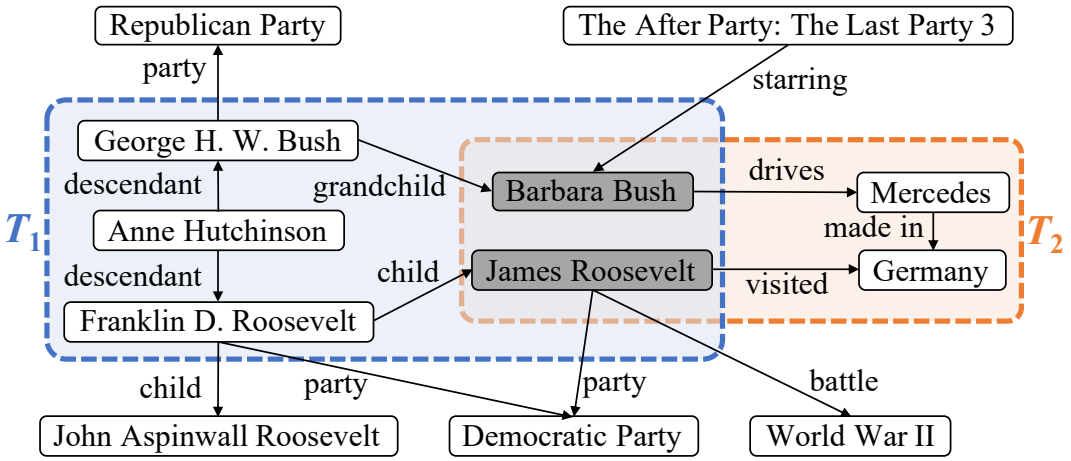


图 4-1: 一个知识图谱示例以及两个抽取出的子图。两个子图都是关键词查询 “barbara bush, james roosevelt” 的结果，其中 T_1 是一个语义内聚的子图，T_2 是一个语义上不内聚的子图。

次组斯坦纳树问题。它比经典组斯坦纳树问题 [24] 的优化目标多了一个代表点对语义距离的二次项。

问题复杂度 QGSTP 是一个 NP-难的优化问题。点权版本的斯坦纳树问题 [66] 可以直观的规约到 $\alpha = 1$ 且 $\forall 1 \le i \le g, |K_i| = 1$ 时的 QGSTP。由于点权版本的斯坦纳树问题是 NP-难问题，因此 QGSTP 也是一个 NP-难问题，QGSTP 的精确算法可拓展性通常较差，本章关注的重点是设计分析 QGSTP 的高效近似算法。

例 4-1 给定图4-1中的知识图谱，T_1 与 T_2 是关键词查询 “barbara bush, james roosevelt” 的结果。

4.2 相关路径集

4.2.1 相关路径集的定义

在图 $G = \langle V, E \rangle$ 中，一个 RPS 是与查询 $Q = \{k_1, \ldots, k_g\}$ 相关的路径的集合。具体的，给定一个根节点 $r \in V$，一个 r-RPS $\mathbf{P}_r = \{P_1, \ldots, P_g\}$ 是 g 条路径的集合，其中每个 $1 \le i \le g$，$P_i = \langle V_{P_i}, E_{P_i} \rangle$ 为路径 r-K_i，其中 r-v_i 是从 r 到一个关键词节点 $v_i \in K_i$ 的路径。注意对于 $i \ne j$，可能有 $K_i \cap K_j \ne \emptyset$ 且 $P_i = P_j$.

例 4-2 考虑例4-1，接下来的路径集组成了以 `A. Hutchinson` 为根的

RPS。

$$\begin{aligned}&\texttt{A. Hutchinson} \xrightarrow{\texttt{descendant}} \texttt{G. Bush} \xrightarrow{\texttt{spouse}} \texttt{B. Bush}\\&\texttt{A. Hutchinson} \xrightarrow{\texttt{descendant}} \texttt{F. Roosevelt} \xrightarrow{\texttt{child}} \texttt{J. Roosevelt}\end{aligned} \tag{4-2}$$

定义函数 vnum 来代表一个 RPS 中节点的数目。它只计算根节点 r 一次：

$$\text{vnum}(\mathbf{P}_r) = 1 + \sum_{P_i \in \mathbf{P}_r} |V_{P_i} \setminus \{r\}|. \tag{4-3}$$

评注 4-1 这里 vnum 只计算根节点 r 一次是为了得到更小的近似比。

4.2.2 相关路径集的转换

同一个查询 $Q = \{k_1, \ldots, k_g\}$ 的 r-RPS $\mathbf{P}_r = \{P_1, \ldots, P_g\}$ 与结果树 $T = \langle V_T, E_T \rangle$ 可以相互转换。

结果树转 RPS

如下的方式可以将 T 转成 $\mathbf{P}_r$：任意选择一个 $r \in V_T$ 作为根。对于 $1 \le i \le g$，令 $v_i \in V_T \cap K_i$ 是 T 中的一个关键词节点，P_i 即 T 中从 r 到 v_i 的唯一路径。注意由于 v_i 可能不唯一，$\mathbf{P}_r$ 也可能不唯一。

RPS 转结果树

$\mathbf{P}_r$ 转成 T 的基本思路是将 $\mathbf{P}_r$ 中所有的 r-v_i 路径逆序“融合”进 G 的一个子树 T 中，这个子树包含了所有的关键词且是联通的，接着不断移除 T 中不必要的节点和相应的边即可得到答案。这个算法4.1称为 `GenAns`。

正确性方面，用数学归纳法容易证明算法4.1第 1-10 行得到的 T^* 是一棵包含所有关键词且联通的树。第 14 行判断可删除的条件是这个节点没有孩子且删除后树依然覆盖所有关键词，则第 12-18 行保证得到的 T^* 极小。

时间复杂度方面，哈希 T^* 中所有的节点，可使第 1-10 行的复杂度是 $O(\text{vnum})$；对于第 12-18 行，用数组记录每个关键词被覆盖的次数，则第 14 行需要 $O(g)$ 判断是否可移除。第 15 行的移除只标记当前节点被移除而不真正在其父亲的孩子队列中移除，则所需时间为 $O(1)$。入队次数为叶子树加移除的节点数，为 $O(\text{vnum})$，所以第 12-18 行时间复杂度为 $O(g \cdot \text{vnum})$。综上，算法4.1的时间复杂度为 $O(g \cdot \text{vnum})$。

算法 4.1 GenAns

输入: $G = \langle V, E\rangle$ 且 $\mathbf{P}_r = \{P_1, \ldots, P_g\}$
输出: 结果树 T

```
1:  T* ← null
2:  for i = 1 to g do
3:      for all v ∈ reverse(P_i) do
4:          if v ∉ T* then
5:              T* ← T* ∪ v
6:          else
7:              break
8:          end if
9:      end for
10: end for
11: Q ← T* 的叶子节点
12: while Q 非空 do
13:     v ← Q.pull()
14:     if v is removable then
15:         T* ← T*\v
16:         Q.insert(v's father)
17:     end if
18: end while
19: return T*
```

注意由于删除的点和边的顺序可能不唯一，T 也可能不唯一。

路径数相关的引理

对于可以互相转化的 $\mathbf{P}_r$ 与 T，接下来的两个引理给出了 T 中的点在 $\mathbf{P}_r$ 的路径中出现的次数的上下界。

引理 4-1 T 中每个点至少出现在 $\mathbf{P}_r$ 的一条路径上。

证明: 用反证法，假设存在一个 $u \in V_T$ 未出现在 $\mathbf{P}_r$ 的任意一条路径上，用 GenAns 将 $\mathbf{P}_r$ 转化为一棵结果树 T'，由 $u \notin \mathbf{P}_r$ 得 T' 是 T 的真子图，这与 T 的结构最小性矛盾。 □

引理 4-2 除了根 r 之外的所有点最多出现在 $\mathbf{P}_r$ 的 $g-1$ 条路径上。

证明: 用反证法，假设存在一个 $u \in V_T$ 使得 $u \neq r$ 且 $\mathbf{P}_r$ 的所有路径上，定义所有 u 到 v_i 的子路径集为 $\mathbf{P}_u$，用 GenAns 将 $\mathbf{P}_u$ 转化为一棵结果树 T'，由 $r \notin T'$ 得 T' 是 T 的真子图，这与 T 的结构最小性矛盾。 □

4.2.3 相关路径集的代价函数

对于一个 r-RPS $\mathbf{P}_r = \{P_1, \ldots, P_g\}$，此处 $\forall 1 \le i \le g, P_i = \langle V_{P_i}, E_{P_i} \rangle$，定义如下的代价函数

$$\begin{aligned} \mathtt{pcost}(\mathbf{P}_r) = \alpha(\mathtt{wt}(r) + \sum_{P_i \in \mathbf{P}_r} \sum_{v \in V_{P_i} \setminus \{r\}} \mathtt{wt}(v)) \\ + (1-\alpha)\frac{\mathtt{vnum}(\mathbf{P}_r)}{2} \sum_{P_i \in \mathbf{P}_r} \sum_{v \in V_{P_i} \setminus \{r\}} \mathtt{sd}(r, v)\,. \end{aligned} \tag{4-4}$$

对于可以相互转化的 $\mathbf{P}_r$ 与 $T = \langle V_T, E_T \rangle$，以下的命题给出了 $\mathtt{cost}(T)$ 与 $\mathtt{pcost}(\mathbf{P}_r)$ 的近似关系。

命题 4-3 对于 $g > 1$，存在这样的近似关系：

$$\exists \rho \in V_T,\ \mathtt{pcost}(\mathbf{P}_\rho) \le (g-1)^2 \mathtt{cost}(T)\,. \tag{4-5}$$

证明: 接下来证明式 (4-6) 中的 ρ 满足命题

$$\rho = \arg\min_{r \in V_T} \sum_{v \in V_T \setminus \{r\}} \mathtt{sd}(r, v)\,. \tag{4-6}$$

根据式 (4-1)，有

$$\begin{aligned} \mathtt{cost}(T) &= \alpha \sum_{v \in V_T} \mathtt{wt}(v) + (1-\alpha) \sum_{\substack{v_i, v_j \in V_T \\ i<j}} \mathtt{sd}(v_i, v_j) \\ &= \alpha \sum_{v \in V_T} \mathtt{wt}(v) + (1-\alpha)\frac{1}{2} \sum_{r \in V_T} \sum_{v \in V_T \setminus \{r\}} \mathtt{sd}(r, v) \\ &\ge \alpha \sum_{v \in V_T} \mathtt{wt}(v) + (1-\alpha)\frac{|V_T|}{2} \min_{r \in V_T} \sum_{v \in V_T \setminus \{r\}} \mathtt{sd}(r, v)\,. \end{aligned} \tag{4-7}$$

使用式 (4-6) 重写式 (4-7)，得

$$\mathtt{cost}(T) \ge \alpha \sum_{v \in V_T} \mathtt{wt}(v) + (1-\alpha)\frac{|V_T|}{2} \sum_{v \in V_T \setminus \{\rho\}} \mathtt{sd}(\rho, v)\,. \tag{4-8}$$

式 (4-8) 右边的第一项乘以 $(g-1)^2$，得

$$\begin{aligned}(g-1)^2\alpha\sum_{v\in V_T}\mathtt{wt}(v) &= (g-1)^2\alpha(\mathtt{wt}(\rho)+\sum_{v\in V_T\setminus\{\rho\}}\mathtt{wt}(v))\\ &\geq \alpha\mathtt{wt}(\rho)+(g-1)\alpha\sum_{v\in V_T\setminus\{\rho\}}\mathtt{wt}(v)\\ &\geq \alpha(\mathtt{wt}(\rho)+\sum_{P_i\in\mathbf{P}_\rho}\sum_{v\in V_{P_i}\setminus\{\rho\}}\mathtt{wt}(v))\,.\end{aligned} \tag{4-9}$$

第一个不等式是由于 $g>1$。第二个不等式是由于引理 4-2。

式 (4-8) 右边的第二项乘以 $(g-1)^2$，得

$$\begin{aligned}&(g-1)^2(1-\alpha)\frac{|V_T|}{2}\sum_{v\in V_T\setminus\{\rho\}}\mathtt{sd}(\rho,v)\\ &= (1-\alpha)\frac{(g-1)|V_T|}{2}(g-1)\sum_{v\in V_T\setminus\{\rho\}}\mathtt{sd}(\rho,v)\\ &\geq (1-\alpha)\frac{\mathtt{vnum}(\mathbf{P}_\rho)}{2}(g-1)\sum_{v\in V_T\setminus\{\rho\}}\mathtt{sd}(\rho,v)\\ &\geq (1-\alpha)\frac{\mathtt{vnum}(\mathbf{P}_\rho)}{2}\sum_{P_i\in\mathbf{P}_\rho}\sum_{v\in V_{P_i}\setminus\{\rho\}}\mathtt{sd}(\rho,v)\,.\end{aligned} \tag{4-10}$$

两个不等式都是由于引理 4-2。

连列式 (4-8)，式 (4-9)，式 (4-10) 和式 (4-4)，得

$$(g-1)^2\mathtt{cost}(T)\geq\mathtt{pcost}(\mathbf{P}_\rho)\,, \tag{4-11}$$

□

4.3 质量导向的近似算法

解决 QGSTP 的基本思路是计算一个最小代价的 RPS，将 RPS 转化为一个结果树并保证近似比。本节具体介绍质量导向算法 **QO** 。

4.3.1 质量导向算法

寻找最小 cost 的结果很难，最小 pcost 的 RPS 更容易计算。考虑可以互相转换的最优结果树 T^* 以及 ρ^*-RPS $\mathbf{P}_{\rho^*}$，搜索全局最小 pcost 的 RPS $\mathbf{P}^{\#}$ 并将

算法 4.2 QO

输入: $G = \langle V, E\rangle$ 和 $Q = \{k_1, \ldots, k_g\}$
输出: 结果树 $T^{\#}$

```
1: P# ← null
2: for all r ∈ V do
3:     P_r^min ← FindRPS(G, Q, r)
4:     if pcost(P_r^min) < pcost(P#) then
5:         P# ← P_r^min
6:     end if
7: end for
8: T# ← GenAns(P#)
9: return T#
```

其转化为一棵结果树 $T^{\#}$。$T^{\#}$ 可能不是全局最优解，但 $\texttt{cost}(T^{\#})$ 与 $\texttt{cost}(T^{*})$ 的差异可以根据命题4-3通过 $\texttt{pcost}(\mathbf{P}^{\#})$ 与 $\texttt{pcost}(\mathbf{P}_{\rho^*})$ 进行限制。

为了找到 $\mathbf{P}^{\#}$，对于每个根节点 r，寻找 r-RPS 中 pcost 局部最小的 $\mathbf{P}_r^{\min}$。**QO** 穷举所有的点作为根节点，它可以保证一个较好的近似比，但运行速度较慢。

如算法4.2所示，$\mathbf{P}^{\#}$ 代表目前 pcost 全局最小的 RPS（第 1 行）。对于每个根节点 $r \in V$（第 2–7 行），可以找到 pcost 局部最小的 r-RPS：

$$\mathbf{P}_r^{\min} = \arg\min_{\mathbf{P}_r} \texttt{pcost}(\mathbf{P}_r). \tag{4-12}$$

使用子过程 FindRPS（第3行）可以找到 $\mathbf{P}_r^{\min}$，FindRPS 的细节将在第4.3.3节中说明。若 $\mathbf{P}_r^{\min}$ 的 pcost 比 $\mathbf{P}^{\#}$ 更小，则更新 $\mathbf{P}^{\#}$（第 4–5 行）。最终，使用第4.2.2节中介绍的 GenAns 生成结果树 $T^{\#}$ 并返回。

4.3.2 近似比分析

下面的定理说明算法 **QO** 存在近似比保证。

定理 4-4 算法 **QO** 是 QGSTP 的 $2(g-1)^2$ 近似算法。

证明: 令 T^{*} 为最优结果。

当 $g = 1$，T^{*} 是 K_1 中点权值最小的点。即 $T^{*} = T^{\#}$。

当 $g > 1$，令 $\mathbf{P}_{\rho^*}$ 为 T^* 转换得到的 ρ^*-RPS 且满足式 (4-5)：

$$\mathtt{pcost}(\mathbf{P}_{\rho^*}) \le (g-1)^2\mathtt{cost}(T^*). \tag{4-13}$$

在 **QO** 中，$T^\# = \langle V_{T^\#}, E_{T^\#}\rangle$ 是由 `GenAns` 从 $\mathbf{P}^\#$ 生成的。假定 $\mathbf{P}^\# = \mathbf{P}^{\min}_{r^\#}$ 是 $r = r^\#$ 时由 `FindRPS` 找到的 `pcost` 局部最小的 $r^\#$-RPS。根据式 (4-4)，有

$$\begin{aligned}
&2\mathtt{pcost}(\mathbf{P}^\#)\\
&= 2\alpha(\mathtt{wt}(r^\#) + \sum_{P_i\in\mathbf{P}^\#}\sum_{v\in V_{P_i}\setminus\{r^\#\}}\mathtt{wt}(v)) + (1-\alpha)\mathtt{vnum}(\mathbf{P}^\#)\sum_{P_i\in\mathbf{P}^\#}\sum_{v\in V_{P_i}\setminus\{r^\#\}}\mathtt{sd}(r^\#, v)\\
&\ge 2\alpha\sum_{v\in V_{T^\#}}\mathtt{wt}(v) + (1-\alpha)|V_{T^\#}|\sum_{v\in V_{T^\#}\setminus\{r^\#\}}\mathtt{sd}(r^\#, v)\\
&\ge 2\alpha\sum_{v\in V_{T^\#}}\mathtt{wt}(v) + (1-\alpha)\sum_{\substack{v_i,v_j\in V_{T^\#}\\ i<j}}\mathtt{sd}(r^\#, v_i) + \mathtt{sd}(r^\#, v_j)\\
&\ge 2\alpha\sum_{v\in V_{T^\#}}\mathtt{wt}(v) + (1-\alpha)\sum_{\substack{v_i,v_j\in V_{T^\#}\\ i<j}}\mathtt{sd}(v_i, v_j)\\
&\ge \mathtt{cost}(T^\#).
\end{aligned} \tag{4-14}$$

第一个不等式是由于引理 4-1。第二个不等式是由于引理 4-2。第三个不等式是由于 `sd` 的直递性。最后一个不等式是由于式 (4-1)。

在 **QO** 中，令 $\mathbf{P}^{\min}_{\rho^*}$ 为 $r = \rho^*$ 时 `FindRPS` 找到的 `pcost` 局部最小的 ρ^*，有

$$\mathtt{pcost}(\mathbf{P}^\#) \le \mathtt{pcost}(\mathbf{P}^{\min}_{\rho^*}) \le \mathtt{pcost}(\mathbf{P}_{\rho^*}). \tag{4-15}$$

连列式 (4-14)，式 (4-15) 与式 (4-13), 得

$$\mathtt{cost}(T^\#) \le 2\mathtt{pcost}(\mathbf{P}^\#) \le 2\mathtt{pcost}(\mathbf{P}_{\rho^*}) \le 2(g-1)^2\mathtt{cost}(T^*). \tag{4-16}$$

□

4.3.3 局部最小相关路径集

计算 `pcost` 局部最小的 r-RPS $\mathbf{P}^{\min}_r$ 是 **QO** 的关键步骤，本节给出一个基于动态规划的解法。

定义 pcost 的变式

$$\begin{aligned}\mathtt{pcost}'(\mathbf{P}_r) &= \mathtt{pcost}(\mathbf{P}_r) - \alpha\mathtt{wt}(r)\\ &= \alpha\sum_{P_i\in\mathbf{P}_r}\sum_{v\in V_{P_i}\setminus\{r\}}\mathtt{wt}(v) + (1-\alpha)\frac{\mathtt{vnum}(\mathbf{P}_r)}{2}\sum_{P_i\in\mathbf{P}_r}\sum_{v\in V_{P_i}\setminus\{r\}}\mathtt{sd}(r,v)\\ &= \sum_{P_i\in\mathbf{P}_r}\sum_{v\in V_{P_i}\setminus\{r\}}\alpha\mathtt{wt}(v) + \mathtt{vnum}(\mathbf{P}_r)\frac{1-\alpha}{2}\mathtt{sd}(r,v)\,.\end{aligned} \tag{4-17}$$

由于 $\alpha\mathtt{wt}(r)$ 出现在每个 r-RPS 的 pcost 中，最小化 pcost′ 函数等价于最小化 pcost：

$$\mathbf{P}_r^{\min} = \arg\min_{\mathbf{P}_r}\mathtt{pcost}(\mathbf{P}_r) = \arg\min_{\mathbf{P}_r}\mathtt{pcost}'(\mathbf{P}_r)\,. \tag{4-18}$$

此外，pcost′ 计算了路径集的代价和，它存在最优子结构，暗示可以使用动态规划。但 pcost′ 存在 $\mathtt{vnum}(\mathbf{P}_r)$，这依赖于 $\mathbf{P}_r$。这里的处理办法是考虑 $\mathtt{vnum}(\mathbf{P}_r)$ 所有可能值，这样每次 $\mathtt{vnum}(\mathbf{P}_r)$ 都是一个常数。

具体的，考虑使得 $\mathtt{vnum}(\mathbf{P}_r) = n$ 的 $\mathbf{P}_r$，对于起点为 r 的路径 $P = \langle V_P, E_P\rangle$，定义

$$\mathtt{pl}_n(P) = \sum_{v\in V_P\setminus\{r\}}\alpha\mathtt{wt}(v) + n\frac{1-\alpha}{2}\mathtt{sd}(r,v)\,. \tag{4-19}$$

对于 $v\in V$，m 条边构成的 r 到 v 的路径中 $\mathtt{pl}_n$ 的最小值这样计算：

$$\mathtt{pd}_n[v][m] = \begin{cases}0 & m=0,\ v=r\,,\\ \infty & m=0,\ v\neq r\,,\\ \min\limits_{u\in\mathtt{N}(v)}\mathtt{pd}_n[u][m-1] + \alpha\mathtt{wt}(v) + n\frac{1-\alpha}{2}\mathtt{sd}(r,v) & m>0\,.\end{cases} \tag{4-20}$$

此处 $\mathtt{N}(v)$ 是 G 中 v 的邻居的集合。则 m 条边构成的 r 到 K_i 的路径中 $\mathtt{pl}_n$ 的最小值为

$$\mathtt{pdk}_n[i][m] = \min_{v\in K_i}\mathtt{pd}_n[v][m]\,. \tag{4-21}$$

对于 $1\le I\le g$，令 $Q_I\subseteq Q$ 为 Q 中前 I 个关键词：

$$Q_I = \{k_1,\ldots,k_I\}\,. \tag{4-22}$$

迭代计算恰好包含 m 条边且与 Q_I 相关（由 I 条路径构成，其中第 i 条路径为

算法 4.3 子过程 `FindRPS`

输入： $G = \langle V, E\rangle$, $Q = \{k_1, \ldots, k_g\}$ 和 $r \in V$
输出： `pcost` 局部最小的 r-RPS $\mathbf{P}_r^{\min}$

$\mathbf{P}_r^{\min} \leftarrow$ null
for $n = 1$ to $g(|V|-1)$ **do**
  $\mathtt{pd}_n[r][0] \leftarrow 0$
  for $v \in V$ s.t. $v \neq r$ **do**
    $\mathtt{pd}_n[v][0] \leftarrow \infty$
  end for
  for $m = 1$ to $\min\{n-1,\ |V|-1\}$ **do**
    for $v \in V$ **do**
      $\mathtt{pd}_n[v][m] \leftarrow \min\limits_{u \in \mathbb{N}(v)} \mathtt{pd}_n[u][m-1] + \alpha \mathtt{wt}(v) + n^{\frac{1-\alpha}{2}} \mathtt{sd}(r, v)$
    end for
  end for
  for $i = 1$ to g **do**
    for $m = 0$ to $\min\{n-1,\ |V|-1\}$ **do**
      $\mathtt{pdk}_n[i][m] = \min\limits_{v \in K_i} \mathtt{pd}_n[v][m]$
    end for
  end for
  for $m = 0$ to $\min\{n-1,\ |V|-1\}$ **do**
    $\mathtt{pc}_n[1][m] = \mathtt{pdk}_n[1][m]$
  end for
  for $I = 2$ to g **do**
    for $m = 0$ to $n-1$ **do**
      $\mathtt{pc}_n[I][m] \leftarrow \min\limits_{0 \le x \le \min\{m,\ |V|-1\}} \mathtt{pc}_n[I-1][m-x] + \mathtt{pdk}_n[I][x]$
    end for
  end for
  $\mathbf{P}_r^n \leftarrow$ Reconstruct($\mathtt{pc}_n[g][n-1]$)
  if $\mathtt{pcost}'(\mathbf{P}_r^n) < \mathtt{pcost}'(\mathbf{P}_r^{\min})$ **then**
    $\mathbf{P}_r^{\min} \leftarrow \mathbf{P}_r^n$
  end if
end for
return $\mathbf{P}_r^{\min}$

r-K_i）的 `pcost′` 最小的 r-RPS：

$$\mathtt{pc}_n[I][m] = \begin{cases} \mathtt{pdk}_n[I][m] & I = 1\,, \\ \min\limits_{0 \le x \le \min\{m,\ |V|-1\}} \mathtt{pc}_n[I-1][m-x] + \mathtt{pdk}_n[I][x] & I > 1\,. \end{cases} \tag{4-23}$$

由于使得 $\mathtt{vnum}(\mathbf{P}_r) = n$ 的 $\mathbf{P}_r$ 一共有 $n-1$ 条边，需要的结果即为 $\mathtt{pc}_n[g][n-1]$。

最终，考虑所有可能的 $\mathrm{vnum}(\mathbf{P}_r)$，得

$$\min_{\mathbf{P}_r} \mathrm{pcost}'(\mathbf{P}_r) = \min_{1\le n\le 1+g(|V|-1)} \mathrm{pc}_n[g][n-1]. \tag{4-24}$$

上述的计算就是算法4.3中描述的子过程 FindRPS。对于每个可能的 n（第 2-29 行），计算 $\mathrm{pc}_n[g][n-1]$（第 3-24 行）并重建这个使得 $\mathrm{vnum}=n$ 且 pcost′ 最小的 r-RPS $\mathbf{P}_r^n$（第 25 行）。子过程 Reconstruct 需要一些额外的数组用标准的方式记录最小的路径和 RPS，由于比较直观，论文中不再描述。最终，更新 $\mathbf{P}_r^{\min}$（第 26-27 行）并返回 pcost 局部最优的 r-RPS（第 30 行）。

4.3.4 运行时间分析

FindRPS 的运行时间为 $O(g|V|(|V|+|V||E|+g|V|^2+|V|+g^2|V|^2))$，即 $O(g|V|^2|E|+g^3|V|^3)$，GenAns 的运行时间为 $O(g^2|V|)$。因此，算法 **QO** 的时间复杂度为 $O(g|V|^3|E|+g^3|V|^4)$。

4.4 效率导向的近似算法

针对 **QO** 时间复杂度过大的问题，本节使用以质量换时间的策略，具体介绍效率导向算法 **EO**。

4.4.1 效率导向算法

与计算每个 $r\in V$ 的 pcost 局部最小的 r-RPS 的算法 **QO** 不同，为了效率，算法4.4只计算某个 K_i 中的所有 r。这个算法称为 **EO**。算法 **EO** 计算 $K_{i_{\min}}$ 中所有的节点（第 3 行），这里 $K_{i_{\min}}$ 是 $K_1,\ldots,K_g$ 中包含关键词节点最少的点集（第 1 行）。剩余的部分与算法 **QO** 一致。

4.4.2 近似比分析

首先通过一个例子说明 **EO** 可能得到比 **QO** 更差的结果。

例 4-3 继续例子 4-2, 式 (4-2) 中的 RPS 在算法 **QO** 的搜索空间中但不在算法 **EO** 的空间中，这是因为根节点 A. Hutchinson 不是关键词节点，未被算法 **EO** 穷举到。

算法 4.4 EO

输入: $G = \langle V, E\rangle$ 和 $Q = \{k_1, \ldots, k_g\}$
输出: 结果树 $T^{\#}$

$i_{\min} \leftarrow \arg\min_{1\le i\le g} |K_i|$
$\mathbf{P}^{\#} \leftarrow$ null
for all $r \in K_{i_{\min}}$ **do**
  $\mathbf{P}_r^{\min} \leftarrow$ FindRPS(G, Q, r)
  if pcost($\mathbf{P}_r^{\min}$) < pcost($\mathbf{P}^{\#}$) **then**
    $\mathbf{P}^{\#} \leftarrow \mathbf{P}_r^{\min}$
  end if
end for
$T^{\#} \leftarrow$ GenAns($\mathbf{P}^{\#}$)
return $T^{\#}$

下面的定理证明算法 **EO** 也有一个近似比，尽管近似比不如算法 **QO** 。

定理 4-5 算法 **EO** 是 QGSTP 的 $(g-1)^2|V|$ 近似算法。

证明: 令 $T^* = \langle V_{T^*}, E_{T^*}\rangle$ 是一个最优结果。

当 $g = 1$ 或者 $|V_{T^*}| = 1$，T^* 是 $K_{i_{\min}}$ 中点权值最小的点。即 $T^* = T^{\#}$。

当 $g > 1$ 且 $|V_{T^*}| > 1$，对于 $\rho^* \in V_{T^*} \cap K_{i_{\min}}$，令 $\mathbf{P}_{\rho^*}$ 为 T^* 转换得到的 ρ^*-RPS 且满足式 (4-1)，有

$$\begin{aligned}
\frac{|V_{T^*}|}{2}\mathtt{cost}(T^*) &= \frac{|V_{T^*}|}{2}(\alpha \sum_{v\in V_{T^*}} \mathtt{wt}(v) + (1-\alpha) \sum_{\substack{v_i, v_j \in V_{T^*} \\ i<j}} \mathtt{sd}(v_i, v_j)) \\
&= \frac{|V_{T^*}|}{2}(\alpha \sum_{v\in V_{T^*}} \mathtt{wt}(v) + (1-\alpha)\frac{1}{2} \sum_{r\in V_{T^*}} \sum_{v\in V_{T^*}\setminus\{r\}} \mathtt{sd}(r, v)) \\
&\ge \frac{|V_{T^*}|}{2}(\alpha \sum_{v\in V_{T^*}} \mathtt{wt}(v) + (1-\alpha)\frac{1}{2} 2 \sum_{v\in V_{T^*}\setminus\{\rho^*\}} \mathtt{sd}(\rho^*, v)) \\
&\ge \alpha \sum_{v\in V_{T^*}} \mathtt{wt}(v) + (1-\alpha)\frac{|V_{T^*}|}{2} \sum_{v\in V_{T^*}\setminus\{\rho^*\}} \mathtt{sd}(\rho^*, v)\,.
\end{aligned} \tag{4-25}$$

类似于命题4-3中从式 (4-8) 到式 (4-11) 的推导，从式 (4-25) 可得到：

$$(g-1)^2 \frac{|V_{T^*}|}{2}\mathtt{cost}(T^*) \ge \mathtt{pcost}(\mathbf{P}_{\rho^*})\,. \tag{4-26}$$

类似于命题4-4中从式(4-13)到式(4-16)的推导，从式(4-26)可得到：

$$\texttt{cost}(T^{\#}) \le 2(g-1)^2\frac{|V_{T^*}|}{2}\texttt{cost}(T^*) \le (g-1)^2|V|\texttt{cost}(T^*). \tag{4-27}$$

□

4.4.3 运行时间分析

在最坏情况下，$K_{i_{\min}} = V$。因此算法 **EO** 与 **QO** 的时间复杂度是一样的。不过实际上 $|K_{i_{\min}}| \ll |V|$，**EO** 比 **QO** 快很多。

4.5 启发式优化

为了提升算法 **QO** 与 **EO** 的实际运行效果，本节提出几个启发式优化。第4.5.1节提出了对 `FindRPS` 的剪枝策略，第4.5.2节中提出了根点排序策略。

注意这些启发式优化在很多情况下有助于提升效果，但是它们并不改变算法本身的近似比以及时间复杂度。

4.5.1 剪枝策略

`FindRPS` 是算法 **QO** 与 **EO** 的关键步骤，本节提出三个剪枝策略来极大提升 `FindRPS` 的实际表现。

剪枝 n 的下界

在算法4.3中，最外层循环从 $n = 1$ 开始（第 2 行）。这里给出一个 $\textbf{vnum}(\mathbf{P}_r)$（即 n）的下界作为穷举的起点来剪枝 n 的下界。

具体的，用如下的方式拓展算法4.3。在算法的开始，插入一个以 r 为起点的宽度优先搜索来计算每个 $1 \le i \le g$，r-K_i 的路径的最少边数用 $l_{r,i}$。根据式(4-3)，$\textbf{vnum}(\mathbf{P}_r)$ 一个直观的下界是

$$L_r = 1 + \sum_{1\le i\le g} l_{r,i}. \tag{4-28}$$

在算法4.3中，设置最外层循环的起点为 $n = L_r$（第 2 行）。

剪枝 n 的上界

在算法4.3中，最外层循环的终点是 $n = g(|V| - 1)$（第 2 行）。这个值可能很大，可以借助 $\texttt{pcost}(\mathbf{P}_r)$ 的下界来剪枝 n 的上界。根据式 (4-17)，有

$$\begin{aligned} &\texttt{pcost}(\mathbf{P}_r) = \alpha\texttt{wt}(r) + \texttt{pcost}'(\mathbf{P}_r), \quad \text{此处} \\ &\texttt{pcost}'(\mathbf{P}_r) = \sum_{P_i \in \mathbf{P}_r} \sum_{v \in V_{P_i} \setminus \{r\}} \alpha\texttt{wt}(v) + n\frac{1-\alpha}{2}\texttt{sd}(r, v). \end{aligned} \tag{4-29}$$

注意到 $\texttt{pcost}'$ 计算的是 g 条路径的代价之和–每个 $1 \le i \le g$ 都有一条路径 r-K_i。想法是把一条 r-K_i 的代价映射到带权图上 r-K_i 的距离，这样可以用 r-K_i 的最短路径作为代价的下界。

为了实现这个想法，同样需要拓展算法4.3。在最外层循环开始的地方，创建一个带权有向图 $G_{r,n} = \langle V_{r,n}, E_{r,n} \rangle$，其中 $V_{r,n} = V$ 且每条边 $(u, v) \in E$ 对应于两条有向边 $\langle u, v \rangle, \langle v, u \rangle \in E_{r,n}$，有向边 $\langle u, v \rangle \in E_{r,n}$ 的边权为

$$\alpha\texttt{wt}(v) + n\frac{1-\alpha}{2}\texttt{sd}(r, v). \tag{4-30}$$

在 $G_{r,n}$ 上使用迪杰斯特拉算法来计算 $1 \le i \le g$ 的 r-K_i 的最短路径 $P_{r,n,i}$，它的距离记为 $d_{r,n,i}$。根据式 (4-29)，使得 $\texttt{vnum}(\mathbf{P}_r) = n$ 的 $\texttt{pcost}(\mathbf{P}_r)$ 的一个下界为

$$D_{r,n} = \alpha\texttt{wt}(r) + \sum_{1 \le i \le g} d_{r,n,i}. \tag{4-31}$$

随着 n 的增长，$D_{r,n}$ 也在增长，因此可以检测下面的不等式：

$$D_{r,n} \ge \texttt{pcost}(\mathbf{P}_r^{\min}). \tag{4-32}$$

若这个不等式对于当前的 n 和 $\mathbf{P}_r^{\min}$ 成立，就可以跳出最外层循环并返回当前的 $\mathbf{P}_r^{\min}$。这是因为对于当前以及更大的 n，不会再存在 $\mathbf{P}_r$ 有一个更小的 $\texttt{pcost}$。类似的，测试下面的不等式：

$$D_{r,n} \ge \texttt{pcost}(\mathbf{P}^{\#}). \tag{4-33}$$

若这个不等式对于当前的 n 和 $\mathbf{P}^{\#}$ 成立，就可以跳出最外层循环并返回当前的 $\mathbf{P}_r^{\min}$。

剪枝 m 的上界

在算法4.3中，有一些内部循环终点为 $m = \min\{n-1, |V|-1\}$ 或者 $x = \min\{m, |V|-1\}$，这些值可能很大（例如第 13 行、第 22 行）。这些 m 和 x 代表每个 $1 \le i \le g$ 中 r-K_i 路径中边的数目，可以计算边的数目来剪枝大的 m 和 x。具体的做法是利用剪枝 n 的上界中计算出的最小权 r-K_i 路径 $P_{r,n,i}$。令 $l_{r,n,i}$ 为 $P_{r,n,i}$ 中边的数量。假定 r-RPS $\mathbf{P}_r$ 是对于当前 n 来说使得 $\texttt{vnum}(\mathbf{P}_r) = n$ 的 pcost 局部最小的 r-RPS。若 $\forall 1 \le i \le g$，r-K_i 的路径 $P_i \in \mathbf{P}_r$ 包含超过 $l_{r,n,i}$ 条边，则可以使用 $P_{r,n,i}$ 替换 P_i 来产生另一个 r-RPS $\mathbf{P}'_r$。$\mathbf{P}'_r$ 的总边权由式 (4-30) 定义，不超过 $\mathbf{P}_r$ 且 vnum 更小。因此 $\texttt{pcost}(\mathbf{P}'_r) \le \texttt{pcost}(\mathbf{P}_r)$，即 $\mathbf{P}'_r$ 也是一个局部最小 pcost r-RPS 且它（或者一些其他 pcost 局部最小的 r-RPS）已经在之前更小 n 的枚举中被找到，则不再需要考虑当前 n 的 $\mathbf{P}_r$。

为了实现这个做法，在剪枝 n 的上界之后，需要进一步拓展算法4.3来缩小 m 和 x 的范围：

$$
\begin{aligned}
&\text{第 17 行：}\ m \text{ 改为 } \min\{n-1, |V|-1, l_{r,n,1}\},\\
&\text{第 21 行：}\ m \text{ 改为 } \min\{n-1, \sum_{1\le i\le I} l_{r,n,i}\},\\
&\text{第 22 行：}\ \max\{0, m - \sum_{1\le i\le I-1} l_{r,n,i}\} \le x \le \min\{m, |V|-1, l_{r,n,I}\},\\
&\text{第 13 行：}\ m \text{ 改为 } \min\{n-1, |V|-1, l_{r,n,i}\},\\
&\text{第 7 行：}\ m \text{ 改为 } \min\{n-1, |V|-1, \max_{1\le i\le g} l_{r,n,i}\}.
\end{aligned}
\tag{4-34}
$$

4.5.2 根点排序

在 **QO** 和 **EO** 中，它们最外层的循环穷举了一个根点集，按照不同的顺序穷举可能导致不同的运行时间。具体的，在剪枝 n 的上界中，目前 $\mathbf{P}^{\#}$ 的 pcost 被用来剪枝 FindRPS 中大的 n 并更早的终止 FindRPS。若一个 pcost 更小的 $\mathbf{P}^{\#}$ 能更早的被找到，即 **QO** 和 **EO** 的外循环能更早的被处理，FindRPS 将会更早的终止。下面有一个启发式优化来找到更好的根点，它将会帮助提升近似算法的实际表现。

算法4.5给出了根的顺序。为了找到更好的根，注意到式 (4-4) 里由短路径构成的 RPS 可能有小的 pcost。在算法4.5中，对于每个可能的根 $r \in V$，计算到每个关键词点集的路径 r-K_i，用 dist[r][i] 表示（第 1-5 行），这是由 g 个宽

算法 4.5 根点排序

输入: $G = \langle V, E\rangle$ 和 $Q = \{k_1, \ldots, k_g\}$
输出: G 的点序
1: **for** $i = 1$ to g **do**
2: **for** $r \in V$ **do**
3: $\texttt{dist}[r][i] \leftarrow r$ 到 K_i 的最短路长度
4: **end for**
5: **end for**
6: **for** $r \in V$ **do**
7: $\texttt{R}[r] \leftarrow \sum_{1 \le i \le g} \texttt{dist}[r][i]$
8: **end for**
9: **return** R

度优先搜索实现的，第 i 个搜索由 K_i 中所有的关键词节点开始。接着计算一个 r-RPS 中包含的最小边数，用 $\texttt{R}[r]$ 表示（第 6-8 行）。根点将按 R 递增排序。算法 4.5的时间复杂度是 $O(g(|V| + |E|) + g|V|)$，即 $O(g(|V| + |E|))$，这个复杂度与 **QO** 和 **EO** 的复杂度相比可直接忽略。

算法 4.5作为 **QO** 和 **EO** 的预处理步骤。在 **QO** 中，第 2 层的最外层循环按算法 4.5给出的顺序穷举所有点。令 $U_\tau \subseteq V$ 是算法 4.5计算出的前 τ 个点，在实验中 $\tau = 10$，在 **EO** 中，第 3 行的最外层循环穷举 $U_\tau \cup K_{i_{\min}}$。使用这个启发式，期望是能够减少 **QO** 和 **EO** 的运行时间，并提升 **EO** 的质量（由于考虑了更多的点）。

4.6 实验评估

本节的实验主要是为了验证三个研究假设：

RH4 本章提出的算法可以高效地计算出结果。

RH5 本章提出的算法可以有效计算出代价小的结果。

RH6 与传统的 GST 算法对比，本章提出的算法可以在以合理的代价提升结果的内聚性。

实验还包含一个分离实验和一个用户实验，所有的实验都运行在基于 3.5GHz 的至强处理器和 32GB 内存的 Jvm 上。

数据集 该实验使用仿真 LUBM 图谱以及真实 DBpedia 知识图谱进行实验。这六张图的具体大小汇总在表格 4–1中。

表 4–1: 实验中使用的知识图谱以及相应的查询

	知识图谱			查询	
	图源	$\|V\|$	$\|E\|$	查询数	g
LUBM-2U	LUBM	38,348	153,971	250	2–6
LUBM-10U	LUBM	207,440	850,433	250	2–6
LUBM-50U	LUBM	1,082,832	4,445,949	250	2–6
DBP-50K	DBpedia	50,000	79,689	188	2–6
DBP-500K	DBpedia	500,000	996,537	323	2–6
DBP-5M	DBpedia	5,765,042	18,657,561	397	2–6

LUBM[①]是一个著名的仿真知识图谱基准，它被用来进行大学领域的查询结果评估。本实验生成了三张知识图谱，分别描述 2 个大学（LUBM-2U），10 个大学（LUBM-10U）和 50 个大学（LUBM-50U）。

DBpedia[②]是一个知名的百科知识图谱。本实验抽取了三张子图，分别是 DBP-50K、DBP-500K 和 DBP-5M（完整的 DBpedia），这些图都是从度最大的点出发，随机挑选现有节点的邻居，最后得到这些节点在 DBpedia 上的导出子图。

这些图可以被分为三个种类。

- LUBM-2U 与 DBP-50K 是包含几万个点的**小型**知识图谱。
- LUBM-10U 与 DBP-500K 是包含几十万个点的**中型**知识图谱。
- LUBM-50U 与 DBP-5M 是包含几百万个点的**大型**知识图谱。

查询 对于 LUBM，为了评估不同的查询搜索空间，本实验按照文献 [17] 里的方法随机产生了一些查询。不同的查询其关键词数目 (g) 以及每个关键词匹配到的平均节点数（用 f 表示）不同。具体的，g 的取值为 {2, 4, 6}，默认值为 4，f 的取值为 {10, 100, 1000}，默认值为 100。当一个参数取非默认值时，另一个参数取默认值。每个查询都是随机从知识图谱中选取 g 个关键词集合 $K_i, 1 \le i \le g$，每个 K_i 中有 f 个点。对于每张知识图谱，本实验产生了 250 个查询，每个 g, f 参数组有 50 个查询。

对于 DBpedia，本实验复用了文献 [65] 中提供的 429 个查询，并删除了查询中无法被匹配的关键词。如表格 4–1所示，剩余的查询包含至少 2 个关键

[①] swat.cse.lehigh.edu/projects/lubm/

[②] downloads.dbpedia.org/2016-10/core-i18n/en/mappingbased_objects_en.tql.bz2

词，至多 6 个关键词（即 $2 \le g \le 6$）。本实验中的检索函数 `hits` 将关键词 k 映射到知识图谱中所有在文本标注里包含 k 的节点。

其他的检索函数也可以被使用，但是可能导致不同的搜索空间。由于论文长度的限制，本实验不对其他检索函数进行实验。

代价函数 本论文中提出的算法与点权 (`wt`) 和语义距离 (`sd`) 的计算方法无关，因此，具体的点权和语义距离的实现不是实验的关注点。本实验复用了被证明有效的权重和距离函数。

具体的，仿照文献 [19]，本实验中的点权函数 `wt` 依赖于归一化的 PageRank:

$$\mathtt{wt}(v) = 1 - \mathrm{sigmoid}\left(\log \frac{\mathrm{PageRank}(v)}{\min\limits_{u \in V} \mathrm{PageRank}(u)}\right). \tag{4-35}$$

对于语义距离 `sd`，DBpedia 上每个实体都标注了一些本体中的类型，仿照文献 [26]，本实验中 `sd` 的计算是本体类型集的杰卡德距离，这可以看成本体上的语义距离。不过 LUBM 上每个实体都只有一个类型，上述的杰卡德距离并不适用。在 LUBM，本实验使用 RDF2Vec[67] 生成 10 维嵌入向量，计算两个实体的角距离来度量它们的结构距离。杰卡德距离与角距离都是伪度量函数，满足问题的定义。

代价函数中的 α 设置依赖于具体的应用，为了评估多种场景，本实验考虑 α 的取值 {0.1, 0.5, 0.9}。

4.6.1 实验设置

QO 和 EO 这两个算法都使用了 4.5中的所有启发式优化。

基线 为了验证研究假设，本实验额外实现了 3 个基线算法。一个指数级的 **QGSTP 的精确算法**来验证研究假设 RH5；两个 GST 的算法 [15, 19] 来验证研究假设 RH6，**DPBF** [15] 是一个基于动态规划的精确算法，**BANKS-II** [19] 是一个基于双向搜索的近似算法。这两个算法都希望找到点权最小的结果但不关心语义距离。存在一些更快的算法 [17] 能最小化边权版本的 GST，这些算法并不能很容易地改造成点权版本的 GST，所以没有被用来作为基线比较。

4.6.2 评价指标

本实验使用运行时间（run time）、近似比（approximation ratio）与内聚比（cohesiveness ratio）分别作为研究假设 RH4、RH5 与 RH6 的评价指标。

对于 RH4，本实验计算每个算法处理每个查询的**运行时间 (run time)** 并设置超时为 200 秒，超时发生后相应的程序对于这个查询的运行将立即终止。超时终止时，运行时间将被设置为 200 秒，算法将返回当前的最优值。由于所有参与比较的算法都是逐渐计算更好的结果，这样的处理是合理的。例如 **QO** 与 **EO** 将会返回 $\texttt{GenAns}(\mathbf{P}^{\#})$，此处的 $\mathbf{P}^{\#}$ 代表超时发生时 `pcost` 全局最小的 RPS。

对于 RH5，本实验以标准的方式计算结果树 T 的**近似比（approximation ratio）**：

$$\text{approximation ratio} = \frac{\texttt{cost}(T)}{\texttt{cost}(T^*)}. \tag{4-36}$$

此处的 T^* 代表最优解。计算 T^* 是 NP-难问题，因此只能在小图上计算 T^* 以及对应的近似比。T^* 的计算不受超时的影响。

对于 RH6，本实验度量结果树 $T = \langle V_T, E_T \rangle$ 的内聚性（`coh`）：

$$\texttt{coh}(T) = \sum_{\substack{v_i, v_j \in V_T \\ i<j}} \texttt{sd}(v_i, v_j). \tag{4-37}$$

令 T_{D} 是 DPBF[15] 计算的最小 GST。将 T_{D} 作为基线，并 **EO** 与 **QO** 的**内聚比（cohesiveness ratio）**：

$$\text{cohesiveness ratio} = \frac{\texttt{coh}(T)}{\texttt{coh}(T_{\text{D}})}. \tag{4-38}$$

4.6.3 EO 的高效性（RH4）

表格 4-2展示了查询超时的百分比，表格 4-3展示了查询的平均耗时。

DPBF 和 BANKS-II 是最小点权 GST 目前最优的算法，而 GST 问题没有 QGSTP 难，这张表同时展示 DPBF 和 BANKS-II 的运行时间来说明 QGSTP 的难度。在中型知识图谱上，DPBF 至少有 20% 的查询超时。在最大的 DBP-5M，DPBF 与 BANKS-II 有超过 50% 的查询超时。尽管 QGSTP 更难，**EO** 依然在所有的图上比 DPBF 更快。**EO** 的运行时间与 BANKS-II 相当，它只会在大的图上超时。**EO** 在 DBpedia 上更快，BANKS-II 在 LUBM 上更快。在小型和

表 4–2: 查询超时的百分比

		LUBM-2U	-10U	-50U	DBP-50K	-500K	-5M
$\alpha = 0.1$	**QO**	20.00%	92.00%	100.00%	36.17%	98.45%	100.00%
	EO	0.00%	0.00%	20.00%	0.00%	0.00%	40.81%
$\alpha = 0.5$	**QO**	76.80%	100.00%	100.00%	39.89%	98.45%	100.00%
	EO	0.00%	0.00%	20.00%	0.00%	0.00%	23.93%
$\alpha = 0.9$	**QO**	96.80%	100.00%	100.00%	49.47%	98.45%	100.00%
	EO	0.00%	0.00%	20.00%	0.00%	0.00%	13.35%
	DPBF	0.00%	20.00%	63.20%	0.00%	21.98%	56.68%
	BANKS-II	0.00%	0.00%	0.00%	0.00%	0.00%	55.16%

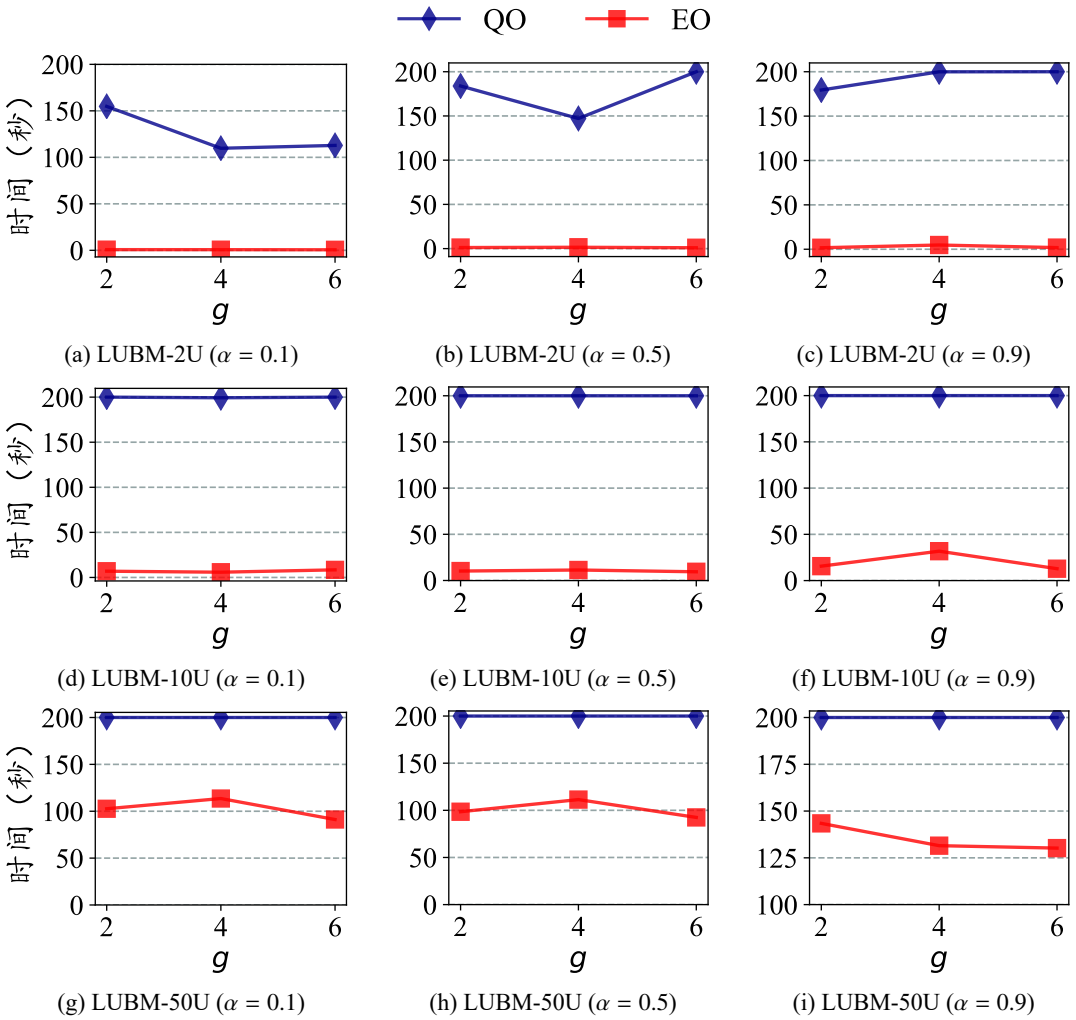
表 4–3: 平均运行时间（s）

		LUBM-2U	-10U	-50U	DBP-50K	-500K	-5M
$\alpha = 0.1$	**QO**	119.36	199.57	200.00	128.91	198.02	200.00
	EO	0.68	6.56	106.90	0.22	3.45	125.52
$\alpha = 0.5$	**QO**	165.05	200.00	200.00	133.55	198.13	200.00
	EO	1.35	10.74	105.03	0.23	3.56	103.53
$\alpha = 0.9$	**QO**	195.83	200.00	200.00	150.96	198.54	200.00
	EO	3.56	24.77	133.62	0.26	4.64	99.54
	DPBF	24.87	71.25	144.14	10.87	71.77	136.11
	BANKS-II	0.50	3.08	18.45	0.27	5.89	174.82

中型知识图谱上，**EO** 只需要几秒就可以完成一次查询，这展示出它速度上的优势。不过 **QO** 几乎在所有的图上都会有超时发生，小图上超时率达 20%，大图上超时率为 100%。

上面的结果部分支撑了 RH4 的正确性，**EO** 能在小型和中型知识图谱上高效的计算结果，运行时间与 BANKS-II 相当，而 **QO** 似乎难以实际使用。不过 **EO** 在大型知识图谱上依然有优化的空间。

本章提出的算法的运行时间与查询中关键词的数目（g）是相关的。在图4–2中，随着 LUBM 中 g 的增长，运行时间产生波动。在图4–3中，随着 DBpedia 中 g 的增长，运行时间总体增长。总的来说，**EO** 与 **QO** 在 g 上是可拓



(a) LUBM-2U (α = 0.1) (b) LUBM-2U (α = 0.5) (c) LUBM-2U (α = 0.9)

(d) LUBM-10U (α = 0.1) (e) LUBM-10U (α = 0.5) (f) LUBM-10U (α = 0.9)

(g) LUBM-50U (α = 0.1) (h) LUBM-50U (α = 0.5) (i) LUBM-50U (α = 0.9)

图 4–2: LUBM 上的平均运行时间

展的。

4.6.4 QO 和 EO 的有效性（RH5）

表格 4–4展示了结果的平均近似比。近似比小说明结果好，考虑到 **QO** 在很多查询上超时，近似比可能受到影响，实验额外实现了算法 **QO** *，这个变式不受超时限制。

QO 和 **QO** * 达到一样的近似比。它们在 LUBM-2U 上只是略好于 **EO**，在 DBP-50K 则完全相同。总的来说，**QO** 和 **EO** 的近似比非常小，在 1.03–1.18 之间。

上面的结果支撑了 RH5 的正确性。两个算法都能计算代价小的结果，]**EO** 的计算耗时比 **QO** 少了很多，却能产生几乎一样好的结果。

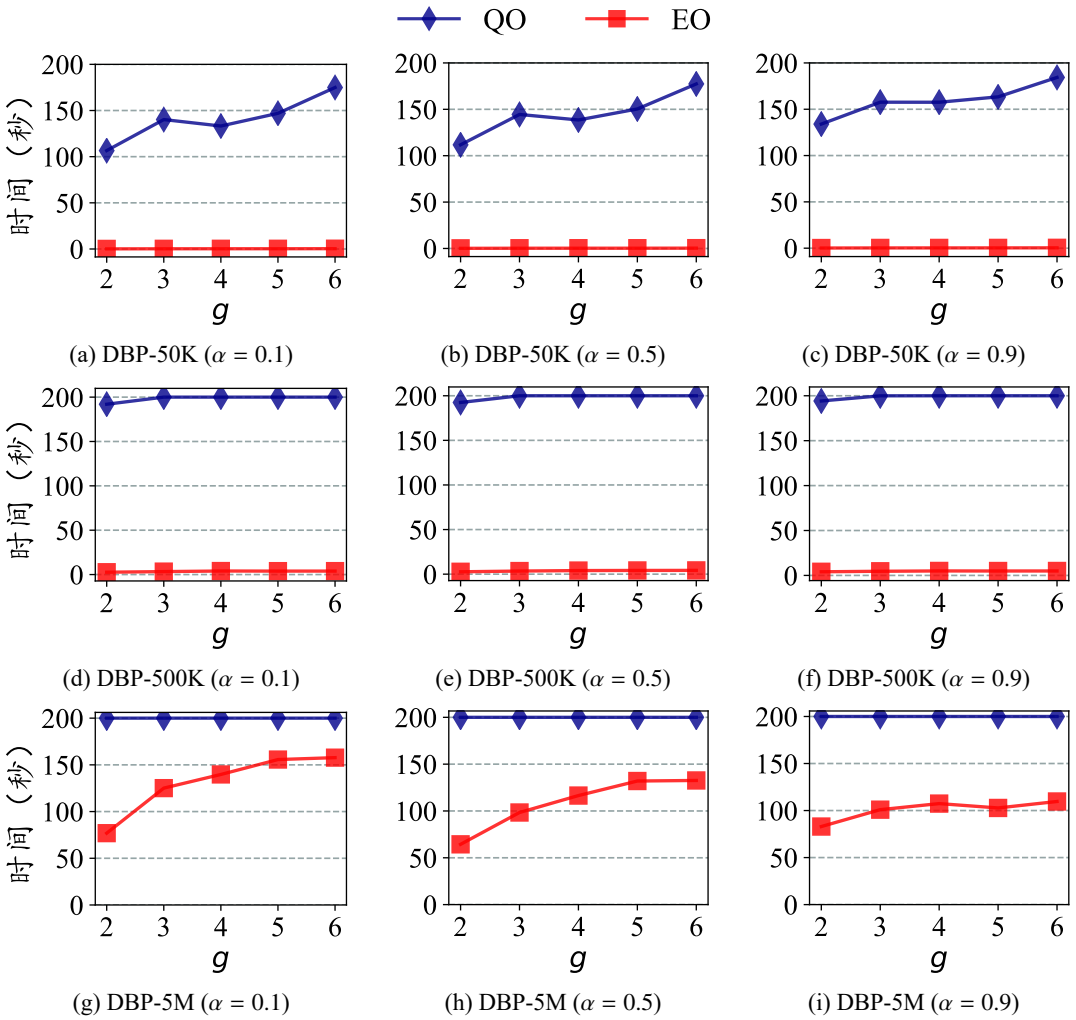


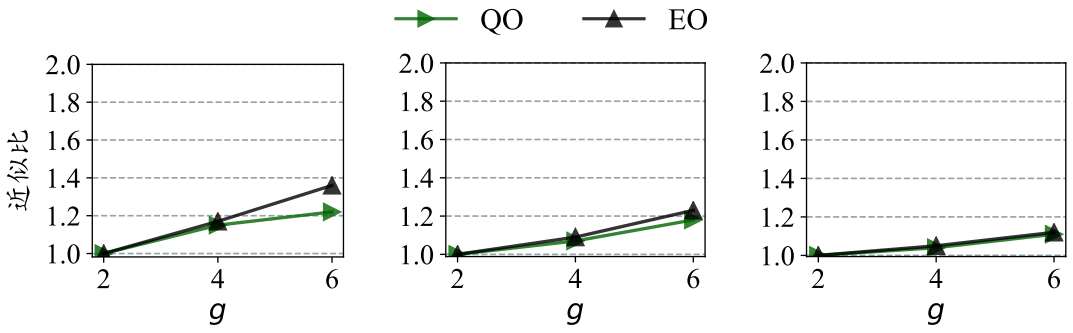
图 4-3: DBpedia 上的平均运行时间

除此之外，当 α 增长时，近似比变得更小，这是由于 α 越大，点权比语义距离更加重要，而点权相对容易近似。

表 4-4: 平均近似比 (* = 无超时限制)

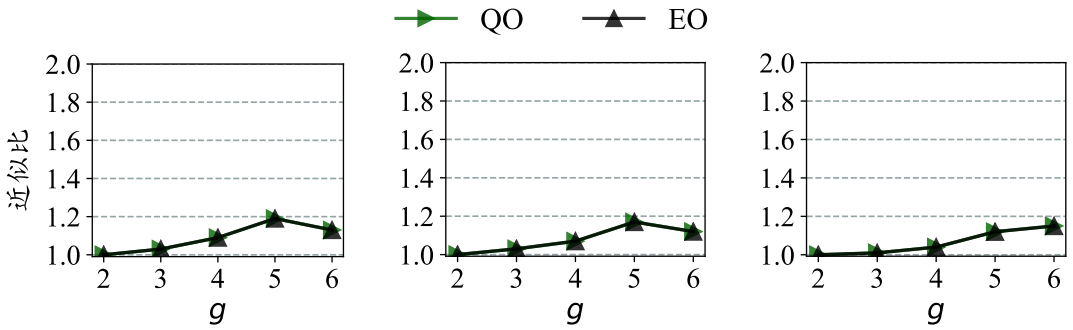
	LUBM-2U			DBP-50K		
	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$
QO	1.13	1.08	1.05	1.06	1.05	1.03
QO *	1.13	1.08	1.05	1.06	1.05	1.03
EO	1.18	1.10	1.06	1.06	1.05	1.03

本章提出的算法的近似比与查询中关键词的数目 (g) 是相关的。在



(a) LUBM-2U (α = 0.1) (b) LUBM-2U (α = 0.5) (c) LUBM-2U (α = 0.9)

图 4–4: LUBM 上的平均近似比



(a) DBP-50K (α = 0.1) (b) DBP-50K (α = 0.5) (c) DBP-50K (α = 0.9)

图 4–5: DBpedia 上的平均近似比

图4–4中，随着 LUBM 中 g 的增长，两个算法的近似比都缓慢增加。在图4–5中，随着 DBpedia 中 g 的增长，两个算法的近似比产生波动。总的来说，**QO** 与 **EO** 的近似比对 g 并不敏感。

4.6.5 QO 和 EO 的内聚性（RH6）

表4–5展示了结果的平均内聚比，内聚比小说明结果更内聚。

QO 和 **EO** 取得类似的结果。总的来说，它们的内聚比在 0.26–0.88。以 α = 0.5 为例，LUBM 上内聚比为 0.5，DBpedia 上内聚比低至 0.26。本章算法的结果比传统最小权 GST 算法的结果内聚 2-4 倍。更大的 DBpedia 上有更小的内聚比，在最大的 DBP-5M 上，内聚比为 0.26–0.35，即本章算法的结果比传统最小权 GST 算法的结果内聚 3-4 倍.

上面的结果支撑了 RH6。与传统的 GST 算法对比，本章的算法极大的提升了结果的内聚比，且付出的代价可以接受。事实上，**EO** 比 DPBF 更快，与 BANKS-II 的速度在伯仲之间。

表 4-5: 平均内聚比

		LUBM-2U	-10U	-50U	DBP-50K	-500K	-5M
$\alpha = 0.1$	**QO**	0.41	0.46	0.44	0.84	0.61	0.26
	EO	0.42	0.47	0.45	0.84	0.61	0.26
$\alpha = 0.5$	**QO**	0.48	0.51	0.48	0.84	0.61	0.26
	EO	0.49	0.51	0.49	0.84	0.62	0.26
$\alpha = 0.9$	**QO**	0.80	0.87	0.86	0.88	0.65	0.35
	EO	0.82	0.86	0.87	0.88	0.66	0.35

随着 α 的增加，内聚比变得更大，这是因为 α 越大，语义距离变得越不重要，**QO** 和 **EO** 更倾向于 GST 的结果。

4.6.6 分离实验

本部分分离实验意在说明启发式优化的作用。共分离了四个启发式：剪枝 n 的下界 (**w/o PSn**)、剪枝 n 的上界 (**w/o PLn**)、剪枝 m 的上界 (**w/o PLm**)、根点排序 (**w/o RVR**)。

表 4-6: 平均运行时间 (s)

	LUBM-10U			DBP-500K		
	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$
EO	6.56	10.74	24.77	3.45	3.56	4.64
w/o PSn	79.99	71.35	59.45	25.58	24.27	26.19
w/o PLn	200.00	200.00	200.00	199.51	199.51	199.51
w/o PLm	10.31	13.44	28.50	3.73	3.99	5.06
w/o RVR	11.52	15.22	30.02	4.08	3.95	4.81

表格 4-6展示了中型知识图谱上查询的平均时间。剪枝 n 的上界与下界对运行时间非常重要。**w/o PLn** 几乎在所有的查询上都达到超时，**w/o PSn** 比 **EO** 慢 2.4–12.2 倍。其他的启发式对运行时间的减少也是有作用的，与 **EO** 相比，**w/o PLm** 多花了 8%–57% 的时间，**w/o RVR** 多花了 4%–76% 的时间。

4.6.7 用户实验

文献 [26, 29] 已经说明了知识图谱搜索中语义内聚的效果。仿照它们的设定，本部分在 DBP-500K 上进行用户实验，用来对比 DPBF 生成的 GST 与本章算法生成的结果树。

用户实验有 16 个参与者，每个人评估 14-20 个查询。对于每个查询，结果树用有向图表示并乱序展示。每个参与者对每个结果树给出 1-4 的评分。

表 4-7: 用户评分

均值 ± 标准差:			
EO ($\alpha = 0.1$)	**EO** ($\alpha = 0.5$)	**EO** ($\alpha = 0.9$)	DPBF
2.552 ± 0.959	2.545 ± 0.974	2.498 ± 0.926	2.349 ± 1.102
显著性差异 ($p < 0.05$):			
EO ($\alpha = 0.1$) > DPBF;		**EO** ($\alpha = 0.5$) > DPBF.	

表格 4-7展示了共 281 个查询的评分的统计分析，**EO** 的分数比 DPBF 更高。重复测量 ANOVA 说明 **EO** 与 DPBF 的差异是显著的 ($p < 0.05$)，LSD post-hoc 分析说明 **EO** ($\alpha = 0.1$) 与 DPBF 之间、**EO** ($\alpha = 0.5$) 与 DPBF 之间存在统计性差异。

上述的结果再一次说明了 [26, 29] 中知识图谱搜索中语义内聚的作用。

第五章　总结与展望

5.1　工作总结

本文研究的是基于知识图谱的关键词搜索，主要分为基于组斯坦纳树的搜索技术和基于二次组斯坦纳树的搜索技术。

对于基于组斯坦纳树的搜索问题，第3章提出了两个新算法：基于静态 HL 的 KeyKG 和基于动态 HL 的 KeyKG$^+$ 。实验显示 KeyKG$^+$ 比现有的算法快 3 个数量级且结果质量的损失可以接受。特别的，在 DBpedia 上，KeyKG$^+$ 可以毫秒级计算出相对好的结果，说明 KeyKG$^+$ 具有实用性。这样的表现主要是由于基于介度中心性的静态 HL 和查询相关的动态 HL。这些 HL 的潜在应用显然不局限于本文中的算法。

对于基于二次组斯坦纳树的搜索问题，第4章提出了两个近似算法：质量导向的 **QO** 和效率导向的 **EO** ，并使用启发式优化保证内聚结果可以被高效地计算。对于提出的算法，本文不仅给出了它们的近似比和运行时间，还通过实验验证它们的实际效果，以及启发式优化的效果。

对于希望高性能的场景，可以使用 KeyKG$^+$ ，对于希望高质量的场景，可以使用 **EO** ，两种算法互补，方便用户使用。

5.2　未来展望

第3章中提出的算法速度快的关键在于中心标注法，但是中心标注法难以支持高效的在线删边操作，因此多项式时间生成 top-k GST 的框架 [68] 并不能直接应用在 KeyKG 与 KeyKG$^+$ 中。所以高效产生 top-k GST 依然是一个待探索的问题。此外，与现有的近似算法 BANKS-II 相比，KeyKG$^+$ 的实际效果仍有提升的空间。最后，存在一些又大又密的图使得所有的 HL 都不能建立较小的标签集，在这些图上 KeyKG$^+$ 并不适用，可以尝试不同的高效距离先知或者使用其他的技术。

第4章中形式化的 QGSTP 和它的近似算法有可能建模并更好的解决已经利

用 GST 建模的其他领域的问题，不过 **QO** 与 **EO** 有如下的限制：考虑有效性，尽管实验已经说明实际的近似比足够令人满意，理论上的最坏近似比依然有提升的空间。考虑效率，为了提升在百万级知识图谱上的查询速度，还可以尝试更多的启发式优化以及尝试使用索引技术。

致　谢

三年前我坐在同样的实验室里用同样的电脑写本科毕业论文的场景还历历在目，转眼间我就要硕士毕业了。这三年有很多收获，虽然有些不舍，却真的要挥手告别学生生涯了。

首先要感谢我的导师程龚老师。在我科研起步时给我指明道路，在我研究陷入瓶颈时帮我走出困境，在我论文多次被拒时依然帮我修改论文，帮我争取到去德国交换的机会，给我充足的时间去找满意的实习与工作。无论学习还是生活中我从没有感受到半点来自他的压力，说实话，我无法想象一个更好的导师应该是什么模样。祝他在以后的科研道路上继续勇攀高峰。

其次我要感谢 Websoft 实验室，实验室的氛围轻松活泼，同学们作息自由却干劲十足，能在这样的实验室是我的荣幸。祝实验室的同学们有美好的未来。

此外我要感谢我的家人，家中自由愉快的成长环境造就了今天的我，感谢他们让我成为了我想成为的人，我只希望他们身体健康。

最后我要感谢待了 7 年的南京大学，不管将来在何处漂泊，这里永远是我内心最温暖的港湾。仰天大笑出门去，我辈乃是南大人！

致　　谢

参考文献

[1] HOGAN A, BLOMQVIST E, COCHEZ M, et al. Knowledge Graphs[J]. CoRR, 2020, abs/2003.02320.

[2] LEHMANN J, ISELE R, JAKOB M, et al. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia[J]. Semant. Web, 2015, 6(2): 167–195.

[3] NOY N F, GAO Y, JAIN A, et al. Industry-scale knowledge graphs: lessons and challenges[J]. Commun. ACM, 2019, 62(8): 36–43.

[4] ARENAS M, GRAU B C, KHARLAMOV E, et al. Faceted search over RDF-based knowledge graphs[J]. J. Web Semant., 2016, 37-38: 55–74.

[5] ZHOU T, LI Z, CHENG G, et al. GREASE: A Generative Model for Relevance Search over Knowledge Graphs[C] // WSDM 2020. 2020: 780–788.

[6] LISSANDRINI M, MOTTIN D, PALPANAS T, et al. Graph-Query Suggestions for Knowledge Graph Exploration[C] // WWW 2020. 2020: 2549–2555.

[7] MARIE N, GANDON F L. Survey of Linked Data Based Exploration Systems[C] // IESD 2014. 2014.

[8] LISSANDRINI M, PEDERSEN T B, HOSE K, et al. Knowledge Graph Exploration: where are we and where are we going?[J]. ACM SIGWEB Newsl., 2020, 2020(Summer): 4.

[9] YU J X, QIN L, CHANG L. Keyword Search in Relational Databases: A Survey[J]. IEEE Data Eng. Bull., 2010, 33(1): 67–78.

[10] LE W, LI F, KEMENTSIETSIDIS A, et al. Scalable Keyword Search on Large RDF Data[J]. IEEE Trans. Knowl. Data Eng., 2014, 26(11): 2774–2788.

[11] SHEKARPOUR S, MARX E, NGOMO A N, et al. SINA: Semantic interpretation of user queries for question answering on interlinked data[J]. J. Web Semant., 2015, 30.

[12] HAN S, ZOU L, YU J X, et al. Keyword Search on RDF Graphs - A Query Graph Assembly Approach[C] // CIKM. 2017 : 227 – 236.

[13] CHENG G, KHARLAMOV E. Towards a semantic keyword search over industrial knowledge graphs (extended abstract)[C] // BigData 2017. 2017 : 1698 – 1700.

[14] BHALOTIA G, HULGERI A, NAKHE C, et al. Keyword Searching and Browsing in Databases using BANKS[C] // ICDE 2002. 2002 : 431 – 440.

[15] DING B, YU J X, WANG S, et al. Finding Top-k Min-Cost Connected Trees in Databases[C] // ICDE 2007. 2007 : 836 – 845.

[16] KASNECI G, RAMANATH M, SOZIO M, et al. STAR: Steiner-Tree Approximation in Relationship Graphs[C] // ICDE 2009. 2009 : 868 – 879.

[17] LI R, QIN L, YU J X, et al. Efficient and Progressive Group Steiner Tree Search[C] // SIGMOD 2016. 2016 : 91 – 106.

[18] LU X, PRAMANIK S, ROY R S, et al. Answering Complex Questions by Joining Multi-Document Evidence with Quasi Knowledge Graphs[C] // SIGIR 2019. 2019 : 105 – 114.

[19] KACHOLIA V, PANDIT S, CHAKRABARTI S, et al. Bidirectional Expansion For Keyword Search on Graph Databases[C] // VLDB 2005. 2005 : 505 – 516.

[20] HE H, WANG H, YANG J, et al. BLINKS: ranked keyword searches on graphs[C] // SIGMOD 2007. 2007 : 305 – 316.

[21] KARGAR M, AN A. Keyword Search in Graphs: Finding r-cliques[J]. Proc. VLDB Endow., 2011, 4(10) : 681 – 692.

[22] YANG Y, AGRAWAL D, JAGADISH H V, et al. An Efficient Parallel Keyword Search Engine on Knowledge Graphs[C] // ICDE 2019. 2019 : 338 – 349.

[23] VOSS S. Steiner's Problem in Graphs: Heuristic Methods[J]. Discr. Appl. Math., 1992, 40(1): 45–72.

[24] IHLER E. The Complexity of Approximating the Class Steiner Tree Problem[C] // WG. 1991: 85–96.

[25] COFFMAN J, WEAVER A C. An Empirical Performance Evaluation of Relational Keyword Search Techniques[J]. IEEE Trans. Knowl. Data Eng., 2014, 26(1): 30–42.

[26] CHENG G, SHAO F, QU Y. An Empirical Evaluation of Techniques for Ranking Semantic Associations[J]. IEEE Trans. Knowl. Data Eng., 2017, 29(11): 2388–2401.

[27] ABRAHAM I, DELLING D, GOLDBERG A V, et al. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks[C] // SEA. 2011: 230–241.

[28] AKIBA T, IWATA Y, YOSHIDA Y. Fast exact shortest-path distance queries on large networks by pruned landmark labeling[C] // SIGMOD. 2013: 349–360.

[29] BRYSON S, DAVOUDI H, GOLAB L, et al. Robust keyword search in large attributed graphs[J]. Inf. Retr. J., 2020, 23(5): 502–524.

[30] FU H, ANYANWU K. Effectively Interpreting Keyword Queries on RDF Databases with a Rear View[C] // ISWC. 2011: 193–208.

[31] TRAN T, CIMIANO P, RUDOLPH S, et al. Ontology-Based Interpretation of Keywords for Semantic Search[C] // ISWC + ASWC. 2007: 523–536.

[32] TRAN T, WANG H, RUDOLPH S, et al. Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data[C] // ICDE. 2009: 405–416.

[33] ZHOU Q, WANG C, XIONG M, et al. SPARK: Adapting Keyword Query to Semantic Search[C] // ISWC + ASWC. 2007: 694–707.

[34] GARCÍA G, IZQUIERDO Y, MENENDEZ E, et al. RDF Keyword-based Query Technology Meets a Real-World Dataset[C] // EDBT. 2017: 656–667.

[35] YANG M, DING B, CHAUDHURI S, et al. Finding Patterns in a Knowledge Base using Keywords to Compose Table Answers[J]. PVLDB, 2014, 7(14): 1809–1820.

[36] TRAN T, HERZIG D M, LADWIG G. SemSearchPro - Using semantics throughout the search process[J]. J. Web Semant., 2011, 9(4): 349–364.

[37] POUND J, HUDEK A K, ILYAS I F, et al. Interpreting keyword queries over web knowledge bases[C] // CIKM 2012. 2012: 305–314.

[38] SHEKARPOUR S, NGOMO A N, AUER S. Question answering on interlinked data[C] // WWW 2013. 2013: 1145–1156.

[39] LI G, OOI B C, FENG J, et al. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data[C] // SIGMOD 2008. 2008: 903–914.

[40] FENG J, LI G, WANG J. Finding Top-k Answers in Keyword Search over Relational Databases Using Tuple Units[J]. IEEE Trans. Knowl. Data Eng., 2011, 23(12): 1781–1794.

[41] TONG H, FALOUTSOS C. Center-piece subgraphs: problem definition and fast solutions[C] // KDD 2006. 2006: 404–413.

[42] KASNECI G, ELBASSUONI S, WEIKUM G. MING: mining informative entity relationship subgraphs[C] // CIKM 2009. 2009: 1653–1656.

[43] CHEN C, WANG G, LIU H, et al. SISP: a new framework for searching the informative subgraph based on PSO[C] // CIKM 2011. 2011: 453–462.

[44] BATEMAN C D, HELVIG C S, ROBINS G, et al. Provably good routing tree construction with multi-port terminals[C] // ISPD. 1997: 96–102.

[45] GUBICHEV A, NEUMANN T. Fast approximation of steiner trees in large graphs[C] // CIKM. 2012: 1497–1501.

[46] ZHU Y, ZHANG Q, QIN L, et al. Querying Cohesive Subgraphs by Keywords[C] // ICDE 2018. 2018: 1324–1327.

[47] DASS A, DIMITRIOU A, AKSOY C, et al. Incorporating Cohesiveness into Keyword Search on Linked Data[C] // WISE 2015, Part II. 2015 : 47 – 62.

[48] SOMMER C. Shortest-path queries in static networks[J]. ACM Comput. Surv., 2014, 46(4) : 45:1 – 45:31.

[49] DELLING D, GOLDBERG A V, PAJOR T, et al. Robust Distance Queries on Massive Networks[C] // ESA. 2014 : 321 – 333.

[50] LI Y, U L H, YIU M L, et al. An Experimental Study on Hub Labeling based Shortest Path Algorithms[J]. PVLDB, 2017, 11(4) : 445 – 457.

[51] ZHANG H, YU H, GOEL A. Pruning based Distance Sketches with Provable Guarantees on Random Graphs[C] // WWW. 2019 : 2301 – 2311.

[52] BAHMANI B, GOEL A. Partitioned multi-indexing: bringing order to social search[C] // WWW. 2012 : 399 – 408.

[53] SARMA A D, GOLLAPUDI S, NAJORK M, et al. A sketch-based distance oracle for web-scale graphs[C] // WSDM. 2010 : 401 – 410.

[54] CHEN W, SOMMER C, TENG S, et al. Compact Routing in Power-Law Graphs[C] // Lecture Notes in Computer Science, Vol 5805 : DISC. 2009 : 379 – 391.

[55] THORUP M, ZWICK U. Approximate distance oracles[J]. J. ACM, 2005, 52(1) : 1 – 24.

[56] GEISBERGER R, SANDERS P, SCHULTES D, et al. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks[C] // WEA. 2008 : 319 – 333.

[57] ZHU A D, MA H, XIAO X, et al. Shortest path and distance queries on road networks: towards bridging theory and practice[C] // SIGMOD. 2013 : 857 – 868.

[58] AKIBA T, IWATA Y, KAWARABAYASHI K, et al. Fast Shortest-path Distance Queries on Road Networks by Pruned Highway Labeling[C] // ALENEX. 2014 : 147 – 154.

[59] OUYANG D, QIN L, CHANG L, et al. When Hierarchy Meets 2-Hop-Labeling: Efficient Shortest Distance Queries on Road Networks[C] // SIGMOD. 2018 : 709 – 724.

[60] ANGELIDAKIS H, MAKARYCHEV Y, OPARIN V. Algorithmic and Hardness Results for the Hub Labeling Problem[C] // SODA. 2017 : 1442 – 1461.

[61] BRANDES U. A faster algorithm for betweenness centrality[J]. J. Math. Soc., 2001, 25(2) : 163 – 177.

[62] ALGHAMDI Z, JAMOUR F, SKIADOPOULOS S, et al. A Benchmark for Betweenness Centrality Approximation Algorithms on Large Graphs[C] // SSDBM. 2017.

[63] IHLER E. Bounds on the quality of approximate solutions to the Group Steiner Problem[C] // WG. 1990 : 109 – 118.

[64] MILLER A H, FISCH A, DODGE J, et al. Key-Value Memory Networks for Directly Reading Documents[C] // EMNLP. 2016.

[65] HASIBI F, NIKOLAEV F, XIONG C, et al. DBpedia-Entity v2: A Test Collection for Entity Search[C] // SIGIR 2017. 2017 : 1265 – 1268.

[66] KLEIN P N, RAVI R. A Nearly Best-Possible Approximation Algorithm for Node-Weighted Steiner Trees[J]. J. Algorithms, 1995, 19(1) : 104 – 115.

[67] RISTOSKI P, ROSATI J, NOIA T D, et al. RDF2Vec: RDF graph embeddings and their applications[J]. Semantic Web, 2019, 10(4) : 721 – 752.

[68] KIMELFELD B, SAGIV Y. Finding and approximating top-k answers in keyword proximity search[C] // PODS. 2006 : 173 – 182.

简历与科研成果

基本信息

石雨轩，男，汉族，1996 年 03 月出生，江苏省淮安人。

教育背景

2018 年 9 月 — 2021 年 6 月 南京大学计算机科学与技术系 硕士
2014 年 9 月 — 2018 年 6 月 南京大学计算机科学与技术系 本科

攻读硕士学位期间完成的学术成果

1. **Yuxuan Shi**, Gong Cheng, Evgeny Kharlamov. Keyword Search over Knowledge Graphs via Static and Dynamic Hub Labelings. In Proceedings of The Web Conference 2020 (WWW 2020), **CCF-A**
2. **Yuxuan Shi**, Gong Cheng, Trung-Kien Tran, Evgeny Kharlamov, Yulin Shen. Efficient Computation of Semantically Cohesive Subgraphs for Keyword-Based Knowledge Graph Exploration. In Proceedings of The Web Conference 2021 (WWW 2021), **CCF-A**
3. **Yuxuan Shi**, Gong Cheng, Trung-Kien Tran, Jie Tang, Evgeny Kharlamov. Keyword-Based Knowledge Graph Exploration Based on Quadratic Group Steiner Trees. In Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021), **CCF-A**

攻读硕士学位期间参与的科研课题

1. 国家重点研发计划课题：知识关联与事件推理类问题求解关键技术与系统（2018YFB1005100）

《学位论文出版授权书》

本人完全同意《中国优秀博硕士学位论文全文数据库出版章程》（以下简称“章程”），愿意将本人的学位论文提交“中国学术期刊（光盘版）电子杂志社”在《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》中全文发表。《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》可以以电子、网络及其他数字媒体形式公开出版，并同意编入《中国知识资源总库》，在《中国博硕士学位论文评价数据库》中使用和在互联网上传播，同意按“章程”规定享受相关权益。

作者签名：石雨轩

2021 年 5 月 28 日

论文题名	基于组斯坦纳树的知识图谱搜索算法研究				
研究生学号	MG1833063	所在院系	计算机科学与技术系	学位年度	2021
论文级别	☑ 硕士 ☐ 硕士专业学位 ☐ 博士 ☐ 博士专业学位 （请在方框内画勾）				
作者 Email					
导师姓名	程龚 副教授				

论文涉密情况：

☑ 不保密

☐ 保密，保密期 (_____年_____月_____日 至 _____年_____月_____日)

注：请将该授权书填写后装订在学位论文最后一页（南大封面）。