无名稿件编号。（将由编辑插入）

使用 OPTIONAL 和 UNION 表达式高效执行 SPARQL 查询

Lei Zou · Yue Pang · M. Tamer Özsu · 陈佳琪

arXiv:2303.13844v1 [cs.DB] 24 Mar 2023

收稿日期：date / 录用日期：date

摘要 RDF 数据集的不断增长促使研究人员致力于优化 SPARQL 查询处理。现有大多数研究都聚焦于基本图模式（BGP），而忽视了 SPARQL 中其他重要的运算符，例如 UNION 和 OPTIONAL。包含这些运算符的 SPARQL 查询（简称 SPARQL-UO）给查询计划生成带来了严峻挑战。本文基于一种新颖的基于基本图模式的评估树（BE 树）查询计划表示方法，提出了以 BGP 执行为构建模块执行 SPARQL-UO 查询的技术。在此基础上，我们提出了一系列由成本驱动的 *BE* 树变换，通过缩减搜索空间和中间结果大小来生成更高效的计划，并提出候选剪枝 技术，进一步提高查询执行效率。实验结果证实，我们的方法比最先进方法快几个数量级。

关键词 图数据库 · SPARQL 查询优化·OPTIONAL 表达式· UNION 表达式

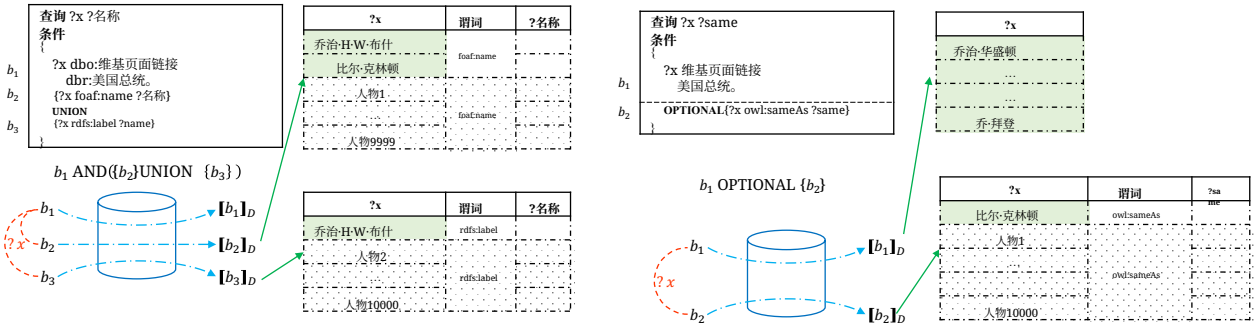
Lei Zou 北京大学，中国北京 电子邮箱：zoulei@pku.edu.cn Yue Pang 北京大学，中国北京 电子邮箱：michelle.py@pku.edu.cn M. TamerÖzsu 滑铁卢大学，加拿大滑铁卢 电子邮箱：tamer.ozsu@uwaterloo.ca 陈佳琪 北京大学，中国北京 电子邮箱：chenjiaqi93@pku.edu.cn

1 引言

知识图谱的广泛发展带来了许多 RDF（资源描述框架）数据管理问题。RDF 是知识图谱事实上的数据模型，其中每条边都是一个由〈主语、谓词、宾语〉组成的三元组。作为访问 RDF 数据集的标准语言，SPARQL 一直是大量研究的重点。现有的大多数研究聚焦于基本图模式（BGP）的执行 [17,24,23], ，它是 SPARQL 的基本构建模块。另一方面，如何执行和优化包含图模式运算符的查询（例如 UNION 和 OPTIONAL），受到的关注则少得多。

UNION 和OPTIONAL表达式是 SPARQL 语法中的关键组成部分。资源描述框架是一种半结构化数据模型，不要求底层数据遵循预定义的模式。这为整合来自不同来源的 RDF 数据提供了灵活性，但也带来了查询挑战，因为相同的信息在 RDF图中可能有多种表示方式。在这种情况下，UNION 运算符至关重要，因为它可以对以不同方式表达的信息进行分组。例如，在 DBpedia[13]，（一个从维基百科中提取的开放领域知识图谱）中，人物的名称使用谓词 〈foaf:name〉 或 〈rdfs:label〉 表示。因此，要完整检索一组人物（例如美国总统）的所有名称，就必须使用 UNION 运算符（图 1(a)）。

除了表示形式的多样性之外，不完整性也是 RDF 数据集的另一项特征。具体而言，某个实体可能缺少大多数其他类似实体所具有的一些属性或关系（这些实体很可能存储在关系数据库的同一张表中）。在这种情况下，OPTIONAL 运算符至关重要，因为它允许将某些属性或关系作为可选信息附加。对于



(a) UNION 子句 (b) OPTIONAL 子句

图 1：一个包含UNION 和OPTIONAL子句的示例查询

例如，图 1(b) 中的 OPTIONAL 查询会检索所有美国总统，以及数据库中指向他们、但不在同一维基百科页面上的其他引用（通过谓词 owl:sameAs）。由于并非每位总统在数据库中都有多个引用，因此带有谓词 owl:sameAs 的三元组被置于OPTIONAL表达式中，这样没有其他引用的总统仍会保留在结果中。

UNION 和OPTIONAL 表达式在现实世界的 SPARQL 工作负载中被广泛使用，也是 SPARQL 1.1 规范的一部分。近期的实证研究 [9] 表明UNION 和 OPTIONAL 表达式分别出现在来自各种端点的真实 SPARQL 查询日志中 25.10% 和 31.72% 的有效查询中。本文研究包含UNION 和OPTIONAL 表达式的 SPARQL 查询的高效执行，并将其简称为 SPARQL-UO 查询。

我们的方法。 鉴于基本图模式已得到充分研究，理想的做法是在性能良好的基本图模式引擎上构建 SPARQL-UO 查询优化。因此，我们首先提出一种基于基本图模式的查询评估方案。具体而言，我们提出一种用于评估 SPARQL-UO 查询的基于基本图模式的评估（BE）树（定义8）。然而，如果直接评估 BE 树，某些基本图模式可能会产生大量中间结果。因此，我们提出一种 BE 树变换方法，以生成更高效的查询计划。

我们引入两种变换：合并 和注入，分别针对UNION 和 OPTIONAL 运算符。这些变换能够在保持查询语义的同时，为降低基本图模式、UNION和 OPTIONAL 运算符求值过程中的成本创造机会。由于转换 BE 树的方式有很多种，我们设计了一种成本模型，用于衡量基本图模式及这些运算符的求值成本，并选择最能降低成本的变换。鉴于可能的变换构成了庞大的搜索空间，我们提出一种贪心策略，逐步确定变换步骤——随后，使用基于基本图模式的方案对变换后的 BE 树进行求值，并通过一种名为候选剪枝的查询时优化加以增强：该优化会在可能的情况下，动态剪枝基本图模式求值的搜索空间。

总而言之，我们的贡献如下：

1. 我们提出了一种用于表示 SPARQL-UO 查询评估计划的新型 BE 树，并设计了两种 BE 树变换原语：合并 和注入，以生成更高效的 SPARQL-UO 查询计划。2. 我们为 SPARQL-UO 查询提出了一种成本模型和一种成本驱动的 BE 树变换算法。3. 我们设计了一种名为候选剪枝的查询时优化方法，通过剪枝搜索空间来增强基于基本图模式的查询评估方案。4. 我们在大规模真实和合成 RDF 数据集上进行了实验，结果表明，我们的方法比现有技术快几个数量级。

本文其余部分安排如下：第2节简要回顾相关工作。第3节介绍一些必要的预备知识。第4节介绍 BE 树计划表示与变换，以及查询评估方案。第5节提出成本驱动的计划选择算法，第6节介绍查询时优化。第7节通过实验评估我们的方法，并在第8节总结全文。

2 相关工作

尽管 SPARQL 查询优化已得到广泛研究，但大多数研究都聚焦于基本图模式求值 [7], , 包括基于图的方法和关系型方法。基于图的方法包括有效索引策略（例如 gStore [24]）和连接顺序优化（例如 WCOJ [12]）方面的研究。相比之下，关系型方法依赖关系型数据库管理系统，并将资源描述框架图视为三列表或其他复杂的

表结构 [5,10]。随后，SPARQL 查询处理被映射为相应的关系型操作，如 Apache Jena [22] 和 Virtuoso [3] 中所做的那样。关系型基本图模式优化方法主要关注如何高效组织数据（例如，属性表 [22], 、垂直分区 [4] 和单表穷举索引 [17]）。然而，这些基本图模式优化技术无法优化UNION 和OPTIONAL，因为这些运算符的语义与连接有着根本区别。正如后文所述，我们的解决方案以基本图模式求值为基础构件，而我们提出的优化技术作用于比基本图模式求值技术更高的层次。

现有关于含UNION和OPTIONAL 运算符的 SPARQL 的研究主要集中在理论层面，研究其语义和复杂度 [18]。例如，Letelier 等人 [15] 提出了 WDPT（良构模式树），重点分析一类名为良构模式的 SPARQL 图模式的包含关系与等价性，并识别其求值中的可处理部分。然而，尚无研究从系统角度探讨 SPARQL-UO 查询优化，即如何设计高效的 SPARQL 查询处理器来执行 SPARQL-UO 查询。据我们所知，LBR [7] 是唯一考虑 OPTIONAL查询优化的工作。该研究设计了一种名为 GoSN 的新数据结构。它与 WDPT 相似，但着重于 OPTIONAL 模式求值的实际方面。具体而言，该研究提出了一种查询重写技术，以减少左外连接的中间结果；左外连接是OPTIONAL所表示的连接语义。为移除不一致的变量绑定，LBR 使用了先前针对 SQL 左外连接研究的空值化和 最佳匹配技术 [20]。LBR 还提出了一种半连接策略来剪枝候选项，并扩展了用于无环内连接最小化的运算符 [8]。然而，该方法沿着连接变量图执行两遍半连接扫描，这种执行策略会在查询执行过程中引入额外开销。本文提出了一种更全面的方法，同时处理UNION和 OPTIONAL。实验还表明，我们的技术在OPTIONAL查询上的表现显著优于 LBR。

请注意，由于语义相似，我们用于优化 UNION 表达式的技术也可应用于带并集的合取关系查询。事实上，所有 SPARQL-UO 查询都可以等价地映射为 SQL，但 OPTIONAL表达式的映射涉及 SQL 中的子查询，因此若不进行重大调整，我们的技术便无法应用 [19,11]。我们注意到，近期有一项演示 [6] 提出了一种针对带并集的合取查询的连接下推 技术：如果成本模型判定这样做更高效，就将连接条件下推到并集集合中。这与我们的方法应用于关系查询时的做法基本相似，但该演示没有说明所采用的成本模型，因此无法进行进一步比较。

3 预备知识

RDF 数据集。 表1是一个包含七个三元组的 RDF 数据集示例（定义如下）。

定义 1（RDF 数据集） 设两两不相交的无限集合 I、B 和 L 分别表示 IRI、空白节点和字面量。RDF 数据集 D 是一个三元组集合$D = \{t_1, t_2, ..., t_{|D|}\}$，其中每个三元组都是一个三元组$t = \langle$主 语、属 性、宾 语$\rangle \in (I \cup B) \times I \times (I \cup B \cup L)$。

SPARQL 查询——语法。 假设存在一个无限集合 V，表示查询中出现的变量。所有变量均以问号（?）开头，以区别于 IRI 和字面量，因此集合 V 与 I 和 L不相交。本文聚焦于 查询 查询，这类查询通过将查询中的图模式与数据集进行匹配来检索结果。需要指出的是，SPARQL 还提供其他查询形式，用于更新数据库、构造 RDF 数据集、询问数据库中是否存在某个图模式，以及描述资源等；这些内容不在本文讨论范围之内。查询 查询的形式为“查询 $v_1 v_2 \ldots v_k$ 条件 {...}”，其中 查询 子句表示查询头部，而条件 子句表示查询主体（图2(a)）。查询 子句决定了需要出现在查询结果中的投影变量，而条件 子句给出了需要在 RDF 数据集上进行匹配的组图模式，该模式可能由许多其他类型的图模式组成，定义如下。

定义 2（三元组模式） 三元组$t \in (V \cup I) \times (V \cup I) \times (V \cup I \cup L)$ 是一个三元组模式。

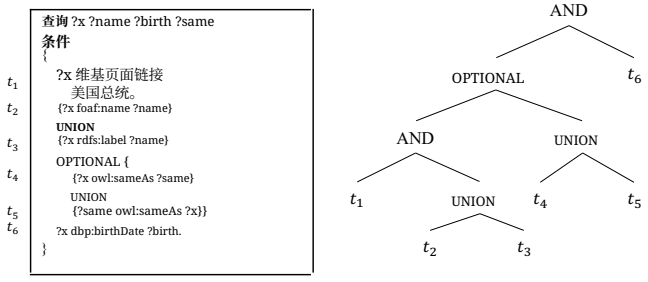
基本图模式（BGP）由三元组模式组成。为了给出 BGP 的形式化定义，我们需要先引入可合并性的概念。

定义 3（可合并三元组模式） 如果且仅当$t_1 = \langle s_1, p_1, o_1 \rangle$ 和$t_2 = \langle s_2, p_2, o_2 \rangle$ 中的$\{s_1, o_1\}$ 与$\{s_2, o_2\}$ 至少共享一个变量，我们就称这两个三元组模式可合并。

直观地说，如果两个三元组模式在主语或宾语位置上具有公共变量，那么它们就是可合并的。由于基本图模式由三元组模式组成，我们可以将可合并性扩展到基本图模式，要求其中某些组成三元组模式是可合并的。

表1：一个 RDF 数据集示例

主语	谓词	宾语
dbr:George_W._Bush	foaf:name	“乔治·沃克·布什"@en
dbr:George_W._Bush	rdfs:label	“乔治·W·布什"@en
dbr:George_W._Bush	dbo:维基页面链接	dbr:美国总统
dbr:Bill_Clinton	foaf:name	“比尔·克林顿"@en
dbr:比尔·克林顿	dbo:维基页面链接	dbr:美国总统
dbr:比尔·克林顿	dbp:birthDate	“1946-08-19"^ ^xsd:date
dbr:Bill_Clinton	owl:sameAs	fbp:Clinton_William_Jefferson_1946-



(a) (b)

图 2：（a）SPARQL 查询示例和（b）二叉树表达式

定义 4（可合并的 BGP）如果存在$t^{i}_{1} \in b_1$ 和$t^{i}_{2} \in b_2$，使得t_1和t_2 是可合并三元组模式，我们就称基本图模式b_1 和b_2 是可合并的。

定义5（基本图模式（BGP））一个 BGP 递归定义如下：

1. 三元组模式 t 是一个 BGP；2. 如果 P_1 和 P_2 是可合并的 BGP，P_1 AND P_2 也属于 BGP。

定义6（图模式、组图模式）

图模式递归定义如下：

1. 如果 P 是一个 BGP，则 P 是一个图模式；2. 如果 P 是一个组图模式（定义如下），则 P 是一个图模式；3. 如果 P_1 和 P_2 都是图模式，则 P_1 AND P_2 也是一个图模式；4. 如果 P_1 和 P_2 都是图模式，{P_1} UNION {P_2}、P_1 OPTIONAL {P_2} 也都是图模式。注意，{P_i} 表示一个组图模式（定义如下）；

组图模式 P 递归定义如下：

1. 如果 P 是图模式，那么{P} 就是组图模式。

图2展示了一个示例 SPARQL 查询，其中包含六个三元组模式（$t_{1..6}$）以及UNION 和OPTIONAL运算符。

SPARQL 查询—语义。 由于 OPTIONAL 子句是左结合的，且运算符的优先级定义如下：{} ≺ UNION ≺ AND ≺ OPTIONAL。

随后，可以将图模式转换为一个表达式，其中包含三元组模式、内置条件以及二元运算符 AND、UNION和 OPTIONAL，这些运算符以两个图模式作为操作数。此类表达式可以等价地表示为一棵二叉树，其中每个叶节点表示一个三元组模式，每个内部节点表示一个二元运算符。图2(b)展示了图2(a)中查询最外层组图模式所对应的二叉树表达式。

在 RDF 数据集 D 上匹配图模式 P（记作 $[\![P]\!]_D$），会产生一个由映射组成的多重集（即多重集合）{μ_1、μ_2、…、μ_n}，其中可能包含重复的映射。映射 $\mu: V \mapsto U$ 是从 V 到 (I ∪L) 的偏函数，其中 V 表示查询中出现的变量，I 和 L 分别表示 IRI 集和字面量集。映射 μ中出现的变量集合记作 dom(μ)。如果对于所有满足μ_1(的变量 v ∈ $dom(\mu_1) \cap dom(\mu_2)$，都有) = μ_2(v)，则定义映射 μ_1 和 μ_2 兼容（记作 $\mu_1 \sim \mu_2$）。直观地说，这意味着 μ_1 和 μ_2 的公共变量都映射到相同的值。如果 μ_1和 μ_2 兼容，那么 $\mu_1 \cup \mu_2$ 也是一个映射。如果映射 μ_1 和 μ_2 不兼容，则将这种情况记作 $\mu_1 \nsim \mu_2$。

我们用 Ω_1 和 Ω_2表示两个映射多重集，并按如下方式定义多重集上的若干运算符：

1. $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1 \wedge \mu_2 \in \Omega_2 \wedge \mu_1 \sim \mu_2\}$.
2. $\Omega_1 \cup_{bag} \Omega_2 = \{\mu_1 \mid \mu_1 \in \Omega_1\} \bigcup_{\text{映射多重集}} \{\mu_2 \mid \mu_2 \in \Omega_2\}$.
3. $\Omega_1 \mid \Omega_2 = \{\mu_1 \in \Omega_1 \mid \forall \mu_2 \in \Omega_2 : \ \mu_1 \nsim \mu_2\}$.4.$\Omega_1 \bowtie \Omega_2 = (\Omega_1 \bowtie \Omega_1) \bigcup_{\text{映射多重集}} (\Omega_1 \mid \Omega_1)$

注意，上述运算符都会保留重复元素，因为它们遵循多重集语义。

定义 7（在 RDF 数据集上对图模式求值） 在 RDF 数据集 D 上对图模式 P 的求值（记为 $[\![P]\!]_D$）递归定义如下：

1. 若P 是一个三元组模式t， $[\![P]\!]_D = \{\mu \mid var(t) = dom(\mu) \wedge \mu(t) \in D\}$ （$var(t)$ 表示t中出现的所有变量，且 $\mu(t)$表示t 中出现的所有变量都被替换为 μ）。

2. 若 P ={P1}， $[\![P]\!]_D = [\![P_1]\!]_D$。3. 若 P =(P1 AND P_2)，$[\![P]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$。4. 若 P =(P1 UNION P_2)， $[\![P]\!]_D = [\![P_1]\!]_D \bigcup$多重集$[\![P_2]\!]_D$。5. 若 P =(P1 OPTIONAL P_2)， $[\![P]\!]_D = [\![P_1]\!]_D$ d⋈ $[\![P_2]\!]_D$。我们称 P1 为OPTIONAL左侧图模式，P2 为 OPTIONAL-右图模式。

4 计划表示：基于BGP的评估树

评估图模式 P 最直接的方法是在二叉树表示上采用自底向上策略。在每一步中，我们要么评估一个三元组模式，要么执行一个二元运算符（AND、UNION 或 OPTIONAL）。这种基于二叉树的评估严格遵循第3节讨论的 SPARQL语义，但由于二叉树表达式叶节点上的每个三元组模式都会生成大量中间结果，因此存在一些固有的性能局限。为说明这一点，请看图3中的简单 SPARQL 查询。请注意，该查询最外层的组图模式只包含一个基本图模式。按照基于二叉树表达式的方法，我们首先需要获得 $[\![t_1]\!]_D$ 和 $[\![t_2]\!]_D$。显然，三元组模式 t_2 会生成大量中间结果，因为数据库中的大多数人物都将出生日期作为一个属性。

显然，使用基本图模式求值作为执行 SPARQL 查询的基本构建模块更为可取，并采用经过优化的基本图模式查询求值方法，例如 RDF-3x [17], SW-store [5], gStore[24] 和 Jena [22]中使用的方法。因此，在我们的方法中，我们设计了一棵基于基本图模式的评估树（BE 树），用于表示 SPARQL 查询评估计划。

4.1 BE树结构

定义8（基于BGP的评估树（BE 树））给定一个组图模式Q，其对应的 BE 树$T(Q)$ 递归定义如下：

– $T(Q)$ 的根节点是一个组图模式节点（定义6），表示查询Q；– $T(Q)$ 的内部节点可以是以下类型之一：｛UNION、OPTIONAL或组图模式｝节点：– UNION节点表示连接两个或多个组图模式的UNION 表达式，这些组图模式称为UNION连接的组图模式。它有两个或多个子节点，且这些子节点都是组图模式节点；– OPTIONAL节点表示连接OPTIONAL左侧和OPTIONAL右侧图模式的OPTIONAL 表达式。它恰好有一个子节点：

即OPTIONAL右侧图模式，它是一个组图模式节点；– $T(Q)$ 的叶节点是一个 BGP 节点（定义 5 ）。

根据上述定义，BE 树中的每个叶节点都对应一个基本图模式，每个内部节点则对应一个组图模式、UNION 表达式或OPTIONAL 表达式。图4展示了 BE 树的一般结构。边标签表示允许出现的此类型子节点的数量：k 表示必须恰好出现k 个此类子节点，而$k..*$表示可以出现k 个或更多此类子节点。为方便起见，我们将一组兄弟节点称为节点的层级。

从 SPARQL 查询构建 BE 树十分简单。图模式之间的连接在 BE 树中通过节点之间的兄弟关系隐式表示。因此，我们首先将一个组图模式节点设为根节点，表示查询中最外层的组图模式。然后，按照原始顺序，将最外层组图模式中的每个连接图模式作为根节点的子节点。对于每个嵌套的组图模式，我们依次将其视为子树的根节点，并递归执行上述过程。

请注意，这种构建过程会生成定义8中未提及的三元组模式节点。为消除这些节点，我们将兄弟三元组模式节点合并为最大的基本图模式节点，使其无法再进一步合并（三元组模式和基本图模式的可合并性定义见定义3和定义4）。基本图模式节点位于其组成部分中最左侧的三元组模式原先所在的位置。显然，通过这一构建过程，SPARQL 查询与 BE 树之间存在一一映射关系。

作为一个具体示例，图 2(a) 中查询的 BE 树如下所示（图5）。请注意，三元组模式 t_1 和 t_6 被合并为一个 BGP 节点；其他三元组模式均无法合并，因此各自构成独立的 BGP 节点。

算法 1 展示了基于 BE 树$T(Q)$ 的基于BGP的解决方案的伪代码，用于回答 SPARQL 查询Q。其基本思路是依靠底层的 BGP 求值引擎分别对每个 BGP 进行求值，然后根据 BE 树合并结果。首先将返回变量r（表示结果集）初始化为空（第 2 行）。随后处理 BE 树根节点的子节点（第 4–20 行）。

遍历根节点的子节点时，考虑以下几种情况：

– 如果当前子节点是组图模式节点，则通过调用该函数对其进行递归求值，调用对象为

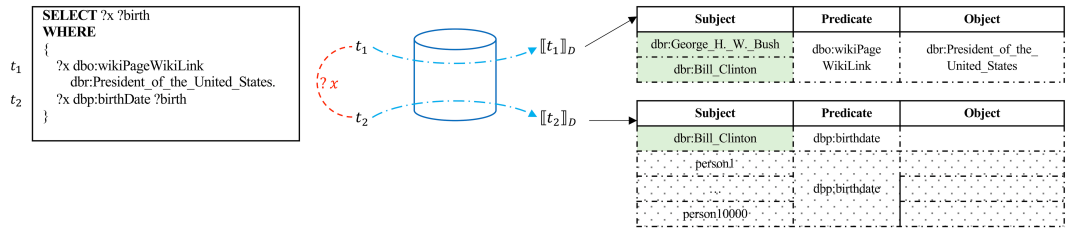


图 3：基于二叉树的查询评估的低效性

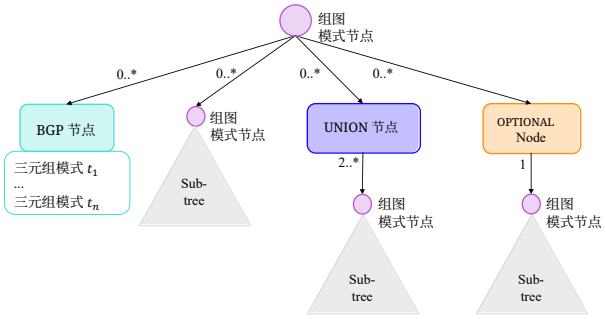


图 4：BE 树的层次结构

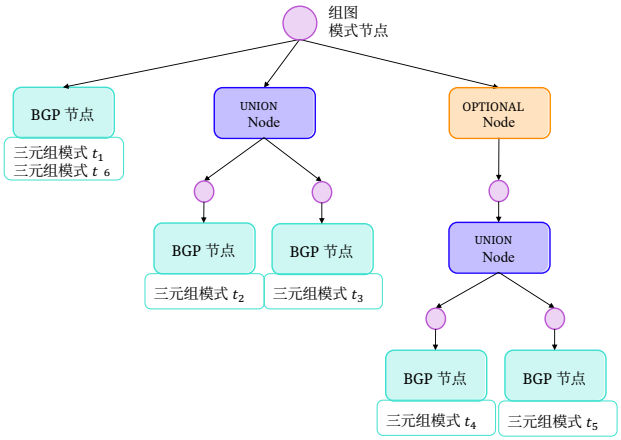


图 5：BE 树示例

the subtree 连接 (第 5–9 行)。 the retrieved result

– 如果当前子节点是 BGP 节点，则使用某种现有的 BGP 查询评估技术对其进行求值，并将检索到的结果与r 连接（第 10–11 行）。

– 如果当前子节点是UNION 节点，则递归地对其每个子组图模式节点进行求值，并通过 $\cup_{bag}$ 操作合并结果。最后将合并后的结果与r 连接（第 12–16 行）。

– 如果当前子节点是OPTIONAL 节点，则递归求值其子组图模式节点，并将检索到的结果与 r 进行左外连接（第 17–20 行）。

算法 1： 基于 BGP 的查询评估

输入： 资源描述框架数据集 D，BE 树 $T(Q)$

输出： $[\![Q]\!]_D$

1 **函数** 基于BGP的评估(D,$T(Q)$)：
2 $r \leftarrow \emptyset$;
3 令 $root$ 为 $T(Q)$ 的根节点;
4 **对每个**根节点的子节点 e_i **执行**
5 **如果** e_i 是组图模式节点**那么**
6 **如果** $r = \emptyset$ **那么**
7 $r \leftarrow$ 基于BGP的评估(D,$T(e_i)$);
8 **else**
9 $r \leftarrow r \bowtie$ 基于BGP的评估(D,$T(e_i)$);
10 **否则，如果** e_i 是一个 BGP 节点**则**
11 $r \leftarrow r \bowtie$ EvaluateBGP(D,e_i);
12 **否则，如果** e_i 是一个 UNION 节点，**则**
13 $u \leftarrow \emptyset$;
14 **对于** e_i 的每个子组图模式节点 P **do**
15 $u \leftarrow u \cup_{bag}$ 基于BGP的评估(D,$T(P)$);
16 $r \leftarrow r \bowtie u$;
17 **否则，如果**e_i 是一个OPTIONAL 节点**则**
18 获取P 的子组图模式节点 e_i;
19 $o \leftarrow$ 基于BGP的评估(D，$T(P)$);
20 $r \leftarrow r ⟕ o$;
21 **返回**r

4.2 BE 树变换

在上一小节中，我们直接对根据查询构建的 BE 树调用基于基本图模式的求值过程（算法 1）。不过，可以通过更改计划来提高查询求值的效率。为此，我们对 BE 树进行某些保持语义的转换。

4.2.1 目标

我们的目标是变换原始 BE 树，使生成的 BE 树具有以下属性：

– 有效性：生成的 BE 树应保持先前定义的树结构，并具有相同的节点类型。通过第4节介绍的直接构造过程，它应与一个语法上有效的 SPARQL 查询一一对应。

–效率：生成的 BE 树的求值效率应高于原始 BE 树。换言之，对生成的 BE 树进行求值的预期成本应更低。

4.2.2 保持语义的变换

我们着手变换 BE 树，并以此前提到的两个目标为指导。为在保持正确性的同时优化查询执行效率，我们需要利用 UNION 和 OPTIONAL 运算符固有的语义等价性，这种等价性通过以下两个定理作了形式化表述。

定理 1 对于任意图模式 P_1, P_2, P_3 以及任意 *RDF* 数据集 D，我们有

$$[\![P_1 \texttt{ AND } (P_2 \texttt{ UNION } P_3)]\!]_D = [\![(P_1 \texttt{ AND } P_2) \texttt{ UNION } (P_1 \texttt{ AND } P_3)]\!]_D.$$

证明 根据定义 7 以及多重集上的运算符定义，我们有

$$[\![P_1 \text{ AND } (P_2 \text{ UNION } P_3)]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2 \text{ UNION } P_3]\!]_D = [\![P_1]\!]_D \bowtie ([\![P_2]\!]_D \cup_{bag} [\![P_3]\!]_D) = ([\![P_1]\!]_D \bowtie [\![P_2]\!]_D) \cup_{bag} ([\![P_1]\!]_D \bowtie [\![P_3]\!]_D) = [\![P_1 \text{ AND } P_2]\!]_D \cup_{bag} [\![P_1 \text{ AND } P_3]\!]_D = [\![(P_1 \text{ AND } P_2) \text{ UNION } (P_1 \text{ AND } P_3)]\!]_D.$$

请注意，定理 1 也可以直接推广到具有两个以上子节点的UNION节点。

定理 2 对于任意图模式 P_1，P_2 和任意 *RDF* 数据集 D，我们有

$$[\![P_1 \texttt{ OPTIONAL } P_2]\!]_D = [\![P_1 \texttt{ OPTIONAL } (P_1 \texttt{ AND } P_2)]\!]_D.$$

这两个等价关系对应于 BE 树上的两种保持语义的变换：将一个节点与其兄弟UNION 节点的子节点合并，以及将一个节点OPTIONAL 注入其兄弟节点的子节点中。我们将这些转换定义如下。

证明 类似地，我们有

$$[\![P_1 \text{ OPTIONAL } (P_1 \text{ AND } P_2)]\!]_D = ([\![P_1]\!]_D \bowtie [\![P_1 \text{ AND } P_2]\!]_D) \cup_{bag} ([\![P_1]\!]_D \setminus [\![P_1 \text{ AND } P_2]\!]_D) = ([\![P_1]\!]_D \bowtie ([\![P_1]\!]_D \bowtie [\![P_2]\!]_D)) \cup_{bag} ([\![P_1]\!]_D \setminus ([\![P_1]\!]_D \bowtie [\![P_2]\!]_D)) = ([\![P_1]\!]_D \bowtie [\![P_2]\!]_D) \cup_{bag} ([\![P_1]\!]_D \setminus [\![P_2]\!]_D) = [\![P_1 \text{ OPTIONAL } P_2]\!]_D.$$

定义 9（合并变换）合并变换是对一个节点执行的操作，该节点表示图模式 P_1，以及它的一个同级UNION节点；该 UNION 节点的子节点表示组图模式 P_2，P_3，<style id='14>⋯、P_n，且同时满足以下两个条件：

1. P_1 是一个基本图模式节点；
2. P_2，P_3，⋯、P_n 中的组图模式至少有一个是与P_1可合并的基本图模式节点的父节点。

该操作包含以下步骤：

1. 将P_1 插入为P_2，P_3，⋯、P_n的最左侧子节点；2. 如果可能，将P_1 与其他基本图模式子节点合并，直到所有基本图模式节点都达到最大；3. 将P_1 从其原始位置移除。

定义 10（注入变换）注入变换是对一个节点执行的操作，该节点表示图模式P_1，；其右侧的一个兄弟OPTIONAL 节点的子节点表示组图模式P_2，，且以下两个条件均成立：

1. P_1 是一个基本图模式节点；2. P_2 是一个可与P_1合并的基本图模式节点的父节点。

该操作包含以下步骤：

1. 将P_1 作为最左侧子节点插入P_2；2. 如果可能，将P_1 与其他基本图模式子节点合并，直到所有基本图模式子节点都达到最大。

图 6 和图 7 展示了这两类变换的实际示例。这些图中查询所针对的图数据库是 DBpedia，它是一个百科全书式的开放领域知识图谱，包含大量现实世界实体的信息。通过这些示例，我们定性地概述了这些变换对查询计划效率的影响。

在图 6 中，b_4 是右侧OPTIONAL 节点的孙代 BGP 节点；b_1，与b_1 以及b_4 可以合并。因此，可用的注入 变换会将b_4 与b_1，合并，从而提高效率。根据原始 BE 树，b_4 会被直接求值，其结果再与b_1的结果进行左外连接。由于大量实体具有?相同实体关系（该关系表示对同一现实世界对象的不同指代之间的等价性），b_4 有许多匹配项，导致其求值和左外连接的开销都很大。然而，美国总统只占少数实体，因此b_1 具有很强的选择性，而注入 变换正是利用了这一点。经过注入后，我们可以依靠底层求值引擎

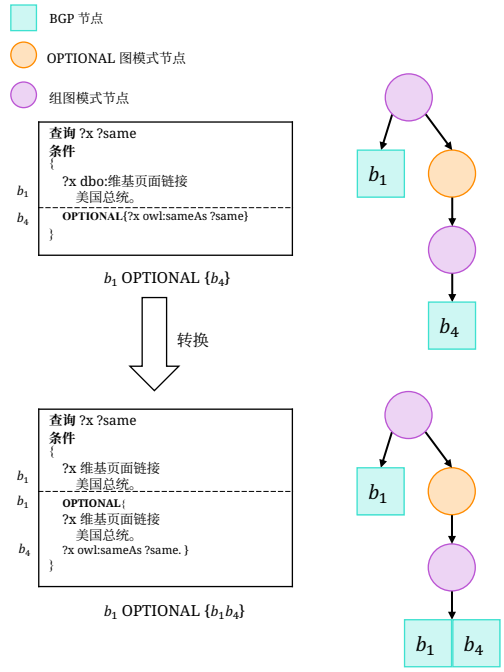


图 6：有利的注入变换

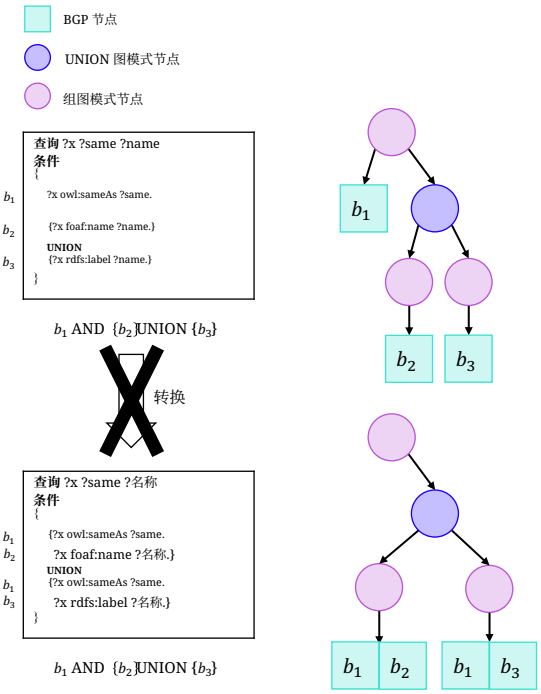


图 7：不利的合并变换

高效地求值合并后的$b_1 b_4$，方法是选择一种连接顺序，先求值选择性更强的b_1。由于$b_1 b_4$的结果数量相较于b_4有所减少，左外连接的开销也随之降低。

这个例子也有助于解释定义9和定义10中为何需要满足某些条件。

显而易见，只有通过合并基本图模式，才能加快基本图模式求值。如果没有发生合并，对合并后的或注入的BGP进行重复求值反而会带来额外开销。

然而，并非所有可用的转换都能提高效率。图 7 展示了一个示例UNION查询中的一种可用合并转换，该转换将基本图模式b_1与其同级UNION 节点合并。由于b_1的选择性较低，合并它既不会加快基本图模式求值，也不会减少中间结果的数量，反而会产生额外开销，因为现在必须对它求值两次。

5 成本驱动的计划选择

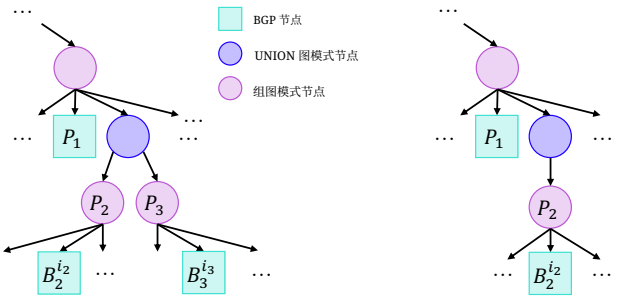
在上一节中，我们已经指出，不同的保语义 BE 树变换在效率方面存在差异。这是经典的基于成本的查询计划选择问题。本节将介绍用于评估合并 或注入 变换的成本模型，以及基于该模型的算法；该算法会根据给定的原始 BE 树确定要执行哪些变换。

我们的成本模型处理 BE 树，因此其操作层级高于基本图模式求值。尽管如此，我们的成本模型仍依赖于对基本图模式的求值成本和结果大小的估算；只要底层基本图模式求值引擎的工作机制是透明的，就可以获得这些估算。此外，通常可以直接从底层基本图模式求值引擎的计划生成模块获得此类估算 [16]。为求完整，我们简要介绍 gStore [24] 和 Jena [22], 的基本图模式成本模型；我们在第5.1.2节中基于这两个系统实现并测试了我们的方法。

5.1 成本模型

5.1 SPARQL-UO 成本模型

我们的成本模型的基本思想源于前述示例（图 6 和图 7）所揭示的规律：SPARQL-UO 查询执行成本由两大部分构成：评估基本图模式的成本，以及通过UNION、OPTIONAL或隐式AND 操作组合部分结果的成本。我们主要关注变换所导致的成本差异，并将其称为 Δ- 成本。只有当变换的 Δ-成本为负（表明成本降低）时，才预期它能提高效率；自然，我们希望找到 Δ-成本最负的变换。



(a) 合并 (b) 注入

图 8：估算合并变换的 Δ-成本

下面我们将讨论如何估算 Δ-成本。考虑图 8 所示 BE 树的一部分，其中包含一个UNION 节点，该节点有两个组图模式子节点P_2 和P_3 ，它们分别具有 BGP 子节点 $B_2^{i_2}$ 和$B_3^{i_3}$。假设$B_2^{i_2}$ 和$B_3^{i_3}$ 与P_1可合并。根据定义 9 中的条件 (2)，P_2 和P_3 中至多有一个缺少可合并的 BGP 子节点；这种情况用空节点$B_2^{i_2}$ 或$B_3^{i_3}$ 表示。

合并变换只会影响一个 BGP 节点（P_1）及其兄弟 UNION 节点（$B_2^{i_2}$ 和$B_3^{i_3}$ ）的 BGP 子节点。变换后，这些 BGP 节点包含的三元组模式可能会发生变化，但这些节点仍会保留（变换产生的空 BGP 节点也会保留）。此外，由于该变换保留了查询语义，P_1的父节点的求值结果不会改变。因此，合并导致的成本差异仅涉及这些节点及其兄弟节点，如图 8(a) 所示。接下来可估算t_m变换合并前 BE 树的局部成本，其中$l(\cdot)$和$r(\cdot)$分别表示该节点左侧和右侧的所有兄弟节点

·，分别为：

$$cost(t_m) = cost(t_m, BGP) + cost(t_m, algebra) \quad (1)$$

$$cost(t_m, BGP) = cost(P_1) + cost(B_2^{i_2}) + cost(B_3^{i_3}) \quad (2)$$

$$\begin{aligned} 代价(t_m,\ 代数) = {} & f_{AND}(|res(P_1)|,\ |res(l(P_1))|,\ |res(r(P_1))|) \\ & + f_{AND}(|res(B_2^{i_2})|, |res(l(B_2^{i_2}))|, |res(r(B_2^{i_2}))|) \\ & + f_{AND}(|res(B_3^{i_3})|, |res(l(B_3^{i_3}))|, |res(r(B_3^{i_3}))|) \\ & + f_{UNION}(|res(P_2)|, |res(P_3)|) \end{aligned} \quad (3)$$

成本$(t_m,$ 基本图模式$)$ 可根据 BGP 求值引擎直接获得。成本$(t_m,$ 代数运算$)$ 是由于受影响的 BGP 节点的结果大小可能发生变化。在合并的情况下，成本$(t_m,$ 代数运算$)$ 由受影响的 BGP 节点与其左右兄弟节点之间执行隐式AND 的成本，以及对P_2 和P_3执行UNION 的成本构成。代数运算的成本是其操作数结果大小的函数（公式 3 中的f_{AND}和f_{UNION}）。这些函数可能因代数运算的不同实现而异。在我们的实验中，为了适配我们选择基于其构建实现的系统，f_{AND} 设为其参数的乘积，f_{UNION} 设为其参数的和。

请注意，对于 Δ 代价估算，我们还需要估算某些节点的结果大小。BGP 节点的结果大小可以通过调用或模拟底层 BGP 求值引擎的估算模块来估算。其他类型节点的结果大小则需要基于假设的数据分布进行估算。在我们的实验中，我们简单地将任何连接（包括 AND 和 OPTIONAL）的结果大小估算为所连接图模式结果大小的乘积，并将 UNION 的结果大小估算为 UNION 运算所涉及图模式的结果大小之和。

假设经过 合并 变换后，受影响的节点变为 $P_1{}'$、$B_2^{i_2\prime}$ 和 $B_3^{i_3\prime}$。要估算经过 t_m 后的局部代价（记为 代价 (t'_m)），我们只需将方程 2 和 3 中的 P_1，$B_2^{i_2}$ 和 $B_3^{i_3}$ 替换为 $P_1{}'$、$B_2^{i_2\prime}$ 和$B_3^{i_3\prime}$。因此，合并 的 Δ 代价可估算如下：

$$\Delta cost(t_m) = cost(t'_m) - cost(t_m) \quad (4)$$

对于注入操作（图 8(b)），情况也类似。因此，可以按如下方式估算注入变换t_i 之前 BE 树的局部代价：

$$cost(t_i) = cost(t_i, BGP) + cost(t_i, algebra) \quad (5)$$

$$cost(t_i, BGP) = cost(P_1) + cost(B_2^{i_2}) \quad (6)$$

$$\begin{aligned} cost(t_i,\ 代数) = {} & f_{AND}(|res(P_1)|,\ |res(l(P_1))|,\ |res(r(P_1))|) \\ & + f_{AND}(|res(B_2^{i_2})|, |res(l(B_2^{i_2}))|, |res(r(B_2^{i_2}))|) \\ & + f_{OPTIONAL}(|res(P_1)|, |res(P_2)|) \end{aligned} \quad (7)$$

假设经过注入变换后，受影响的节点变成$P_1{}'$ 和$B_2^{i_2\prime}$。要估算t_i之后的局部代价，只需将方程 6 和 7 中的P_1 和 $B_2^{i_2}$ 分别替换为$P_1{}'$和$B_2^{i_2\prime}$。随后按如下方式计算注入的 Δ-代价：

$$\Delta cost(t_i) = cost(t'_i) - cost(t_i) \quad (8)$$

5.1.2 基本图模式的成本模型

尽管底层基本图模式成本模型对于我们的 SPARQL-UO 成本模型是透明的（见方程 2 和 6），

为使论述完整，我们简要介绍 gStore 和 Jena 使用的基本图模式成本模型。基本图模式的求值由连接组成。因此，基本图模式计划 T 的成本是每个已执行连接操作 j的成本之和：

$$cost(T) = \sum_{j \in T} cost(j)$$

gStore 中的基本图模式求值使用最坏情况最优（WCO）连接，该连接会处理所有带有指定谓词、连接现有查询顶点与新扩展顶点的边。对于现有顶点上的每个结果元组，至少需要扫描一次所有此类边，以检查该元组能否扩展并匹配新扩展的顶点。假设现有顶点集合为 $\{v_1, \cdots, v_{k-1}\}$，新扩展的顶点为 v_k。那么，WCO 连接的代价可估算如下：

$$\begin{aligned} &\text{代价}(WCO\text{连接}(\{v_1, \cdots, v_{k-1}\}, v_k)) \\ &= \text{卡片}(\{v_1, \cdots, v_{k-1}\}) \times \min_{i \in [1,k-1]} \text{平均\_大小}(v_i, p) \end{aligned}$$

其中，基数($\{v_1, \cdots, v_{k-1}\}$)表示估算的基 数——查询顶点集$\{v_1, \cdots, v_{k-1}\}$上结果元组的估算数量；而平均_大小(v_i, p)表示以p 为谓词、以v_i 为主语或宾语的边（即 三元组）平均数量，具体取决于查询中v_i 与v_k 之间边的方向。

另一方面，Jena 中的基本图模式求值使用二元连接，其概念上类似于关系数据库中的哈希连接。它首先根据共同顶点，对结果规模较小的基本图模式的结果元组进行哈希。然后，对于另一个基本图模式的每个结果元组，查询哈希索引，以查找可兼容并合并的匹配项。假设要合并的两个基本图模式的查询顶点集分别为V_1 和V_2，。那么，二元连接的代价可估算如下：

$$\begin{aligned} &cost(BinaryJoin(V_1, V_2)) \\ &= 2 \times \min(card(V_1), card(V_2)) + \max(card(V_1), card(V_2)) \end{aligned} \tag{9}$$

其中，求和的第一部分表示构建哈希索引的代价，第二部分表示探测该索引的代价。

上述代价估算公式依赖于查询顶点集的基数估算。基数估算从单个三元组模式开始，其查询顶点集的精确基数可以通过使用常量作为键读取资源描述框架存储中预先构建的索引来获得。每当向集合中添加一个新的查询顶点时，我们就对候选结果集进行采样，并统计通过扩展到新查询顶点可以从样本生成多少个结果元组。然后，根据扩展后的结果元组数量与样本大小的比例，对先前的估算进行缩放，从而更新基数估算：

$$card(V_k) = \max(\frac{\#extend}{\#sample} \times card(V_{k-1}), 1)$$

请注意，更复杂的基数估算方法和 BGP 成本模型（例如 [21]）与本文的贡献是相互独立的。实验结果表明，我们的方法能够优化 SPARQL-UO 查询处理，效果显著；该方法考虑了上述简单但有效的 BGP 成本模型和基数估算方法。

5.2 成本驱动转换

本小节将讨论 BE 树变换算法。这些算法利用上述成本模型来决定如何进行转换，以获得执行效率最高的查询计划。

5.2.1 转换单个树层级

我们首先关注较简单的情况，即只考虑单个层级上的转换。

当一个基本图模式节点只有一个同级UNION 或OPTIONAL节点时，如何确定转换已由第 5.1.1 节中引入的成本模型涵盖。然而，实际上，可能有多个同级UNION 或OPTIONAL节点适合进行转换。请注意，根据定理 1 和定理 2，合并后的 BGP 会从其原始位置移除，而注入的 BGP 则保留在其原有位置。这意味着一个基本图模式只能与其同级UNION 节点中的一个合并，但可以注入多个同级OPTIONAL 节点。因此，为了确定合并变换，我们需要全面考察该层级上的所有UNION 节点，并选择成本最低的变换。另一方面，注入变换彼此独立，因此我们从左到右扫描基本图模式节点右侧的每个OPTIONAL 节点，并根据成本分别判断哪些节点适合进行变换。

单层 BE- 树上的变换决策如算法 2 所示。变换发生在输入组图模式节点 P 的子节点层。请注意，BGP 节点的 合并 变换只有在遍历完其所有同级 UNION 节点后（第 14 行）才能确定并执行，而 注入 操作则针对每个同级OPTIONAL节点分别决定（第 16 行）。用于计算每种可能变换的 Δ-成本的子程序见算法 3。

5.2.2 处理多个层级

处理整个 BE 树尤其具有挑战性，因为它通常包含多个层级，而且不同层级间的变换可能相互依赖。例如，若要变换组图模式 {P_1 OPTIONAL{P_2 OPTIONAL P_3}}(P_1，P_2 和 P_3 均为可合并的 BGP)，则有 2^3 种可能的变换，涉及是否将P_1 注入

算法 2： 单层 BE 树变换

输入： 资源描述框架数据集 D、BE 树 $T(Q)$、组图模式节点 P

1 **函数** 单层变换(D,$T(Q)$,P)：
2 　**对于** P_1 中的每个子节点 **执行**
3 　　**如果**P_1 是一个 *BGP* 节点**那么**
4 　　　最小 UNION 成本 ← 0;
5 　　　目标 UNION ← 空节点;
6 　　　**对于**UNION 节点 u（P 的子节点）**do**
7 　　　　当前最小 UNION 成本 ← 决定合并 (P_1, u);
8 　　　　**如果**当前最小 *UNION* 成本 ＜ 小于最小 *UNION* 成本 **那么**
9 　　　　　最小并集代价 ← 当前最小并集代价;
10 　　　　　目标UNION ← U;
11 　　　**如果**最小并集代价 ＜0 **则**
12 　　　　对 subBGPglobal 执行合并，并且目标UNION
13 　　　**对于每个**OPTIONAL P 右侧的节点 o_1 在 P 的子节点**确实**
14 　　　　决定注入 (P_1,O);

算法 3： BE 树变换的子程序

1 **函数** DecideMerge(P_1,U)：
2 　**若** 违反约束**则**
3 　　**返回** 0;
4 　originalCost ← 局部代价（方程 1、2 和 3）;
5 　minUnionCostCur ← 0;
6 　**对于每个**组图模式子节点 P_j *of* U **执行**
7 　　$BSet_j$ ← {B^i_j|B^i_j 是P的 BGP 子节点 $_j$ 可与P_1}合并;
8 　　**如果** $BSet_j = \emptyset$ **那么**
9 　　　向$BSet_j$添加一个空的基本图模式节点;
10 　**遍历**元组 ($B^{i_2}_{2}, B^{i_3}_{3}, \cdots$) 取自 $BSet_2, BSet_3, \cdots$ **执行**
11 　　对P_1 和U执行合并;
12 　　transformedCost ← 局部代价（方程 1、2 和 3）;
13 　　Δ代价 ← transformedCost - originalCost;
14 　　**如果** Δ代价 ＜ *minUnionCostCur* **则**
15 　　　minUnionCostCur ← Δ代价;
16 　　撤销合并;
17 　**返回** minUnionCostCur;
18 **函数** 决定注入(P_1,O)：
19 　**如果** 违反约束**则**
20 　　**返回**;
21 　原始代价 ← 局部代价（公式 1、6 和 7）;
22 　**对于**O 的子组图中的基本图模式子节点 B^i_2 与 P_1 可合并的模式节点 P_2 **执行**
23 　　对注入 P_1 和U 执行（合并*sub* 与 B^i_2）;
24 　　transformedCost ← 局部代价（方程 1、6 和 7）;
25 　　Δ成本 ← transformedCost - originalCost;
26 　　**如果** Δ成本 ≥ *0***那么**
27 　　　撤销 注入;

算法 4： 多级 BE 树转换

输入： 资源描述框架数据集 D，BE 树 $T(Q)$

1 **函数** MultiLevelTransform(D, $T(Q)$)：
2 　后序遍历 (D, $T(Q)$, Q);
3 **函数** PostOrderTraverse(D, $T(Q)$, P)：
4 　**对每个** P_1 属于 P 的子节点 **执行**
5 　　**如果**P_1 是一个组图模式节点**那么**
6 　　　后序遍历 (D,$T(Q)$, P_1);
7 　　**否则如果**P_1 是一个UNION 节点**则**
8 　　　**对于每个**子组图模式节点 P_i 的 P_1 **执行**
9 　　　　后序遍历 (D, $T(Q)$,P_i);
10 　　**否则，如果**P_1 是一个OPTIONAL 节点，**则**
11 　　　获取P_2 的子组图模式节点 P_1;
12 　　　后序遍历（D, $T(Q)$, P_2）;
13 　单层变换（D, $T(Q)$, P）;

P_2，、是否将P_2 注入P_3，，以及是否将P_1 注入P_3。因此，计划空间的规模会随 BE 树的深度呈指数增长。事实上，我们推测，在整个 BE 树上找到最优变换是一个 NP 困难组合优化问题。

为了平衡变换后的树的时间复杂度和效率，我们提出一种贪心策略，以决定对整个 BE 树进行哪些变换（算法 4）。具体来说，我们按照后序深度优先方式遍历 BE 树。只有在遍历完组图模式节点的所有子节点（第 4–12 行）后，我们才会考虑在其子节点层级进行可能的变换（第 13 行，该行调用算法 2）。通过这种方式，我们确保在考虑变换当前层级之前，所有较低层级都已完成适当的变换，并保证整个变换后的树比原始树更高效，且无需进行代价高昂的回溯。

应用变换后，BE 树仍保持树结构，且节点类型不变。定理 1 和定理 2 保证了变换后的 BE 树在语义上的正确性。因此，仍可调用求值算法（算法 1）对变换后的 BE 树进行求值。

6 查询时优化：候选剪枝

上一节介绍了如何通过 BE 树变换生成不同的 SPARQL-UO 查询计划，并在执行前依据成本估算选择有效的计划。本节将介绍候选日期剪枝，这是一种融入算法 1 的查询时优化，可提升效率。

候选剪枝的基本思想同样源自定理 1 和定理 2。求值结果之间的等价性意味着，UNION连接的图模式或OPTIONAL右侧组图模式的结果受到外层图模式中公共变量的约束。

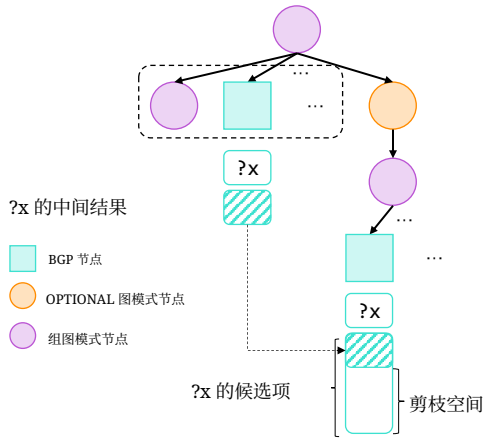


图 9：OPTIONAL 的候选剪枝OPTIONAL

因此，在求值过程中遇到UNION、OPTIONAL或组图模式节点时，可以在执行该节点的子基本图模式时，将公共变量上的当前结果设为候选结果 。图 9 展示了针对 OPTIONAL 查询的候选剪枝机制：已求值图模式中变量?x 的结果，作为OPTIONAL右侧组图模式的子基本图模式中?x 的候选结果，从而剪除?x 的冗余匹配结果；若独立求值该基本图模式，这些结果就会被实例化。

上述讨论表明，候选剪枝能够保持语义正确性。然而，要实现剪枝效果，我们需要确保候选结果的大小小于基本图模式（BGP）实际结果的大小。候选结果较小，也能减少扫描候选结果并将其设为候选项所产生的开销。我们对候选结果大小采用自适应阈值。作为树变换一部分调用的基本图模式成本模型（第 5.1.2 节）会估算基本图模式实际结果的大小；在可能的情况下，我们将该估算值用作候选结果大小的阈值。如果无法获得此类估算值，则根据数据集大小设置阈值。（有关我们实验中的阈值设置，请参阅第 7 节。）

为实现候选剪枝，我们按如下方式修改算法 1（注意，可将结果作为指针参数传递，以避免昂贵的复制操作）：

– 为BGPBasedEvaluation函数添加第三个参数*cand*，表示候选结果；– 处理UNION、OPTIONAL 或组图模式节点时，将当前结果*r* 作为第三个参数传递给 BGPBasedEvaluation（第 7、9、15 和 19 行）；– 将*cand* 作为第三个参数传递给EvaluateBGP（第 11 行）。只有当*cand* 的大小小于阈值时，才将其设为该 BGP 的候选结果。

树变换和候选剪枝分别在查询执行之前和执行期间生效，二者互为补充。执行之前，合并 或注入 变换针对高选择性基本图模式，拆分整体结果较大、原本无法通过候选剪枝处理的图模式。树变换还会向候选剪枝提供基本图模式结果大小的估计。另一方面，由于计划空间庞大，树变换必须逐层执行；而在执行期间，候选剪枝可以将小结果的剪枝效果传递到不同层级。例如，处理组图模式{P_1 OPTIONAL{P_2 OPTIONAL P_3}}的查询时，即使P_1 具有较高选择性，贪心变换策略也无法将其注入P_3 ，但其结果可以通过P_2作为P_3 的候选项。在一种特殊情况下，如果 UNION 或OPTIONAL 节点左侧只有一个基本图模式节点，那么对该基本图模式执行变换就等同于候选剪枝。在这种情况下，会跳过树变换，以避免额外开销。

7 实验

为了评估我们方法的有效性，我们采用 Jena 和 gStore 的基本图模式查询引擎，实现基于基本图模式的成本感知 SPARQL-UO 求值策略。我们从其 GitHub 仓库[1]获取截至2022年6月27日的 Jena 最新版本。所有在 Jena 上运行的实验均启用了基于统计信息的优化。我们从 gStore 的主分支（v0.91）派生出一个分支，并在此基础上实现了我们提出的 SPARQL-UO 优化器[2]。实验使用合成的（LUBM [2]）和真实的（DBpedia[3][1]）RDF 数据集，相关统计信息列于表2。我们的实现以及实验中使用的所有查询均可在匿名 GitHub 仓库中找到[4]。实验在一台 Linux 服务器上进行，该服务器配备 Intel Xeon Gold 6126 中央处理器（2.60 GHz）和 256 GB 内存。

[1] https://github.com/apache/jena.[2]我们的实现可在 https://anonymous. 4open.science/r/gStore-UO/ 获取。[3] 我们使用的 DBpedia 数据转储版本为 V3.9，可从 http://downloads.dbpedia.org/3.9/en/ 下载。我们使用所有 N-Triples文件的拼接结果。[4] https://anonymous.4open.science/r/gStore-UO-opt/。

表 2：数据集统计

数据集	三元组	实体	谓词	字面量
LUBM	534,355,247	86,990,882	18	44,658,530
DBpedia	830,030,460	96,375,582	57,471	59,825,935

表3：LUBM 上的查询统计

	查询	Type	计数$_{BGP}$	深度	$\|[\![Q]\!]_D\|$
第1组	q1.1	U	9	2	645,666
	q1.2	O	3	2	44,653,510
	q1.3	O	4	4	76
	q1.4	O	4	4	5,583
	q1.5	UO	6	3	4,348
	q1.6	UO	9	3	37
第2组	q2.1	O	3	1	4,176,432
	q2.2	O	4	3	8,698
	q2.3	O	4	3	13,124,940
	q2.4	O	2	3	10
	q2.5	O	2	2	10
	q2.6	O	2	2	7

表4：DBpedia 上的查询统计

	查询	Type	计数$_{BGP}$	深度	$\|[\![Q]\!]_D\|$
第1组	q1.1	U	6	2	153,325
	q1.2	UO	4	3	610,434
	q1.3	O	5	5	1,192
	q1.4	UO	7	5	92,041
	q1.5	UO	6	3	3,699,995
	q1.6	UO	10	4	176
第2组	q2.1	O	5	3	490,876
	q2.2	O	2	2	55,054
	q2.3	O	2	2	61,318
	q2.4	O	3	2	4,757
	q2.5	O	2	2	5,330
	q2.6	O	9	2	36

7.1 优化验证

本节中，我们验证第4.2节所提出优化的有效性，并评估以下四种方法：

1. 基线（简称为 基础），在原始 BE 树上调用基于 BGP 的查询评估方法（算法 1），类似于 Jena 和 gStore 中原始 SPARQL-UO 的实现；2. 树变换（简称为 TT ），通过算法 4 对原始 BE 树进行变换，然后在其上调用算法 1；

3. 候选剪枝（简称为 CP），在原始 BE 树上调用加入候选剪枝（第 6 节）的算法 1，使用数据库中三元组总数 1% 的固定阈值；

4. 协调树变换与候选剪枝的完整版（简称为 完整），通过算法 4 对原始 BE 树进行变换，然后调用加入候选剪枝的算法 1，并根据候选结果集大小使用自适应阈值。

据我们所知，目前没有专门针对 SPARQL-UO 查询的基准测试，因此我们构建了一个语义贴近实际、复杂度各异的小型基准测试，其中包含分别针对 LUBM 和 DBpedia 的六个查询，以下记作 q1.1-1.6，详见 [14] 的附录 A。令查询 为查询中最外层的组图模式。为衡量查询的复杂度，我们定义了两个指标：（1）BGP 计数（$Count_{BGP}(Q)$）；（2）嵌套组图模式的最大深度（深度(Q)）。

计数$_{BGP}(P)$图模式P的定义是递归的：

1. 如果 P 是一个基本图模式，则 $Count_{BGP}(P)=1$。2. 如果 $P=\{P_1\}$，则 $Count_{BGP}(P)=Count_{BGP}(P_1)$。3. 如果 $P=P_1$ AND P_2 或 P_1 UNION P_2 或 P_1 OPTIONAL P_2，$Count_{BGP}(P)=Count_{BGP}(P_1)+Count_{BGP}(P_2)$。

深度(P)图模式P 的定义如下：

1. 如果 P 是一个基本图模式，则 深度(P) = 0。2. 如果 $P=\{P_1\}$，则 深度(P) = 深度(P_1) + 1。3. 如果 $P=P_1$ AND P_2 或 P_1 UNION P_2 或 P_1 OPTIONAL P_2，深度(P) = 最大值(深度(P_1), 深度(P_2))。

假设P 是查询Q最外层的组图模式，则有$Count_{BGP}(Q)=Count_{BGP}(P)$，深度($Q$) = 深度($P$)。表3和表4中的第1组汇总了本小节所用查询的统计信息和结果大小。

我们通过查询执行时间来衡量性能。我们还报告执行 TT 和完整树变换所花费的时间。我们的方法在 LUBM 和 DBpedia 上的性能如图10所示。没有柱形表示该查询发生了内存溢出错误。如果执行时间超过 2×10^6 微秒，则认为该查询已超时。

在 gStore 和 Jena 上的结果趋势相似，表明无论底层采用何种 BGP执行引擎，我们的方法都具有适应性。我们的两种优化均被证明有效，因为TT、CP 和完整 在所有查询上的表现都优于基础 。在不同查询和数据集上，TT和CP各有优势。结合使用时，两者的优化效果会累积：完整 在几乎所有查询和数据集上的表现都最佳（gStore 上的 q1.2 除外，此时CP略胜完整 一筹），至少比基线快 2 倍，最高可超过一个数量级。我们的

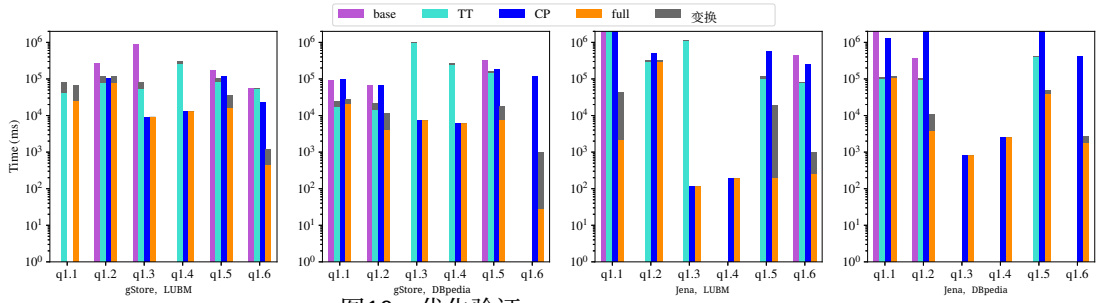


图10：优化验证。

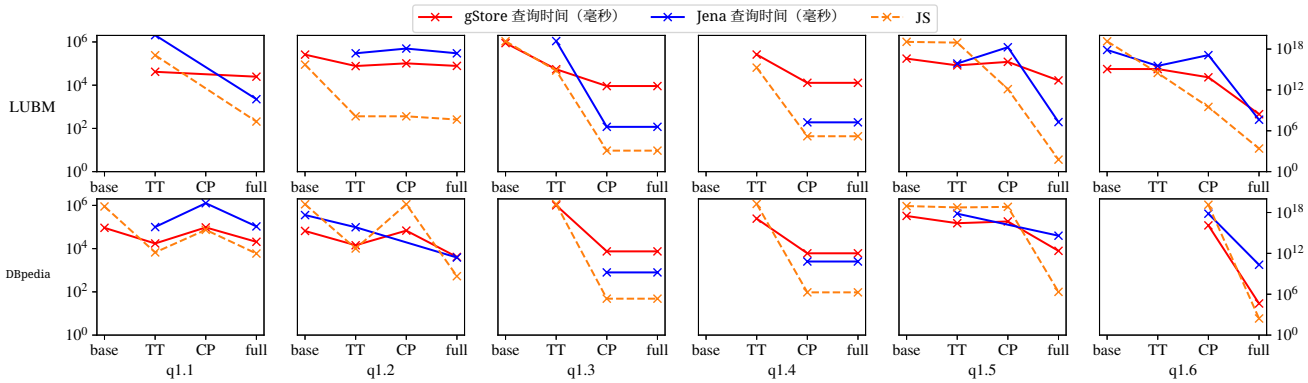


图11：查询的执行时间和连接空间。

优化方法还占用更少的内存。基础 在 24 个查询中有 13 个因内存不足而无法运行，而完整则成功运行了所有查询。

接下来，我们将通过分析基准查询以及优化方法在这些查询上的行为，尝试得出一些结论，说明我们的优化对不同 SPARQL-UO 查询的适用性。

当 TT 有效时。 DBpedia 上的 q1.1（ [14] 的附录A中的清单15）是一个适合使用TT、但不适合使用CP 的查询。该查询首先给出了两个UNION 子句（第2–3行），其子基本图模式的选择性都较低。它们之前没有高选择性的图模式来启用CP。然而，TT 可以将第5–8行中的高选择性基本图模式与第3行的UNION 子句合并，从而加快查询处理并减少内存开销，图10对此进行了验证。LUBM 上的 q1.2 和 DBpedia 上的 q1.2 也属于这一类。（请注意，LUBM 上的 q1.2 对应于第6节提到的特殊情形：在 OPTIONAL 子句之前只有一个基本图模式，因此TT 和 CP的效果相似。）

当 CP 有效时。 LUBM 上的 q1.3（ [14] 的附录A中的清单4）是一个 CP 有效、但 TT 无效的查询。在此查询中，第2行的基本图模式具有较高的选择性，其后是嵌套的低选择性子基本图模式 OPTIONAL。TT 可以将基本图模式注入最外层的 OPTIONAL，但无法触及内部的 OPTIONAL，因此效果有限。然而，CP 可以将少量的结果带入最内层的 OPTIONAL ，并将其设为候选项，以加速查询处理。LUBM 上的 q1.3-4 和 DBpedia 上的 q1.3-4 也属于这一类。

当 TT 和 CP 协同发挥作用时。 LUBM 上的 q1.6（ [14] 的附录A中的清单7）是一个TT和CP互为补充的查询，因此完整 的表现远优于TT 和CP。在该查询中，第2–3行的基本图模式具有高选择性，而第4行的基本图模式选择性相对较低。在获得规模相当大的结果后，CP对后续 UNION 子句的作用有限。然而，TT可以选取高选择性的基本图模式，与第5行的UNION 合并。执行完截至第6行的图模式后，CP 便可以加快后续OPTIONAL子句的处理。LUBM 上的 q1.1 和 q1.5，以及 DBpedia 上的 q1.5 和 q1.6，也属于这一类。

为了定量分析优化效果，我们将图模式 $JS(P)$的连接空间定义如下：

1. 如果 P 是基本图模式，则 $JS(P) = |[[P]]_D|$。2. 如果 $P = \{P_1\}$，则 $JS(P) = JS(P_1)$。3. 如果 $P = P_1$ AND P_2 或 P_1 OPTIONAL P_2，则 $JS(P) = JS(P_1) \times JS(P_2)$。4. 如果 $P = P_1$ UNION P_2，则 $JS(P) = JS(P_1) + JS(P_2)$。

查询的连接空间用于估算该查询执行期间物化的最大中间结果规模。因此，它可以反映查询的执行时间和内存开销。我们绘制了

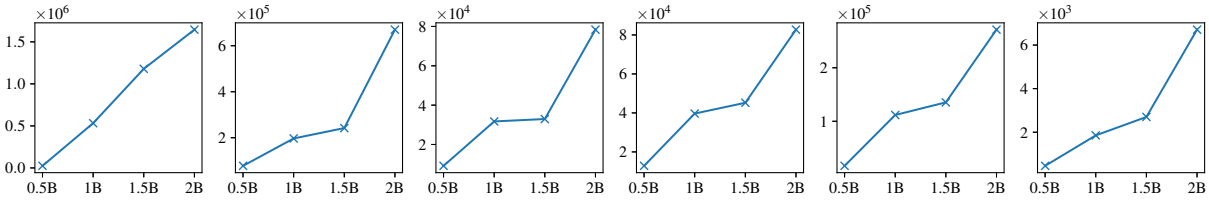


图 12：在不同规模的 LUBM 数据集上执行完整的查询执行时间（毫秒）（“B”代表十亿）

图 11 展示了 gStore 和 Jena 上所有查询的执行时间（左侧 y 轴）及其对应的连接空间（右侧 y 轴）。在测试的各种方法中，这三项指标呈现出相似的趋势。对于所有查询，TT 和 CP 的连接空间都小于基础，而完整 的总体连接空间最小，这印证了上述定性分析。

7.2 与最先进方法的比较

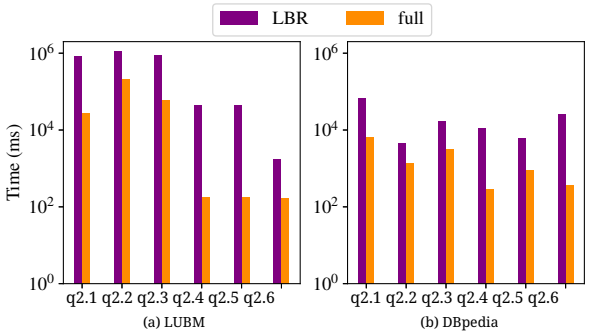


图13：在 LUBM 上与最先进方法的比较。

唯一考虑 SPARQL OPTIONAL查询优化的工作是 LBR [7]。因此，我们将完整 方法与 LBR 进行比较。我们也用 C++ 实现了 LBR。我们在 LBR [7] 提供的 LUBM 和 DBpedia 查询上进行实验，这些查询列为 q2.1-2.6，见 [14] 的附录A。这些查询的统计数据列于表3和表4的第二组中。q2.1-2.3 较为复杂，包含多个嵌套组图模式，每个模式都包含一个低选择率的基本图模式，后接一个只有单个低选择率子基本图模式的 OPTIONAL。与此同时，q2.4-2.6 较为简单，不包含嵌套组图模式，其最外层组图模式包含一个高选择率的基本图模式，后接一个OPTIONAL。

图13展示了完整 方法和 LBR 的总响应时间。在所有查询中，完整 方法都显著快于 LBR；在 q2.4-2.6 上的性能提升比在 q2.1-2.3 上更为显著。这是因为候选剪枝能够利用 q2.4-2.6 中的高选择性基本图模式，而 q2.1-2.3 不包含高选择性基本图模式。

（请注意，由于 q2.1-2.6 中的所有组图模式都包含一个基本图模式，后接一个OPTIONAL 子句，因此它们对应于第6节中提到的特殊情况，在此情况下，树变换与候选剪枝等价，所以只执行候选剪枝。）结果表明，候选剪枝生效时，其效率高于 LBR 复杂繁重的剪枝策略。在 q2.1-2.3 上，完整 方法仍然快于 LBR，因为基于基本图模式的求值方案比 LBR 分别处理三元组模式的方式更高效。

总而言之，尽管 LBR 针对OPTIONAL 查询进行了优化，我们的方法在OPTIONAL查询上的表现仍优于 LBR。

7.3 可扩展性研究

最后，我们评估了我们的方法扩展到更大数据集的能力。通过设置 LUBM 的扩展因子（即大学数量），我们分别生成了三个规模为 10 亿、15 亿和 20 亿个三元组的 LUBM 数据集。我们在这些数据集上运行 q1.1-q1.6 的完整 方法，并在图12中绘制了每个查询的执行时间随数据集大小的变化情况。

这些图展示了我们方法的经验复杂度曲线。我们的方法对于数据集中的三元组数量几乎呈线性扩展。查询执行时间的增长率与各查询的结果大小相关：结果较大的查询，其执行时间随数据集大小增长得更快。（较大 LUBM 数据集上 q1.3-1.6 的结果大小与表3中所示相同，而 q1.1-1.2 的结果大小则呈线性增长。）

8 结论

知识图谱应用的蓬勃发展带来了日益增多的资源描述框架数据管理问题。本文关注如何优化包含UNION 和 OPTIONAL子句的 SPARQL 查询（简称 SPARQL-UO）。利用 SPARQL 引擎中现有的基本图模式查询评估模块，我们提出了一系列针对基于BGP的评估树（BE 树）的代价驱动变换。这些优化能够显著缩小搜索空间并减少中间结果大小，从而

提高 SPARQL-UO 查询求值在时间和空间上的效率。我们通过实验验证了这些优化的有效性，并在包含数百万个三元组的大规模合成和真实资源描述框架数据集上，将优化方法的性能与最先进方法进行了比较。实验结果证实，我们的 SPARQL-UO 查询求值方法比现有方法高效数个数量级。

参考文献

1. DBpedia。网址https://wiki.dbpedia.org/2. LUBM。网址 http://swat.cse.lehigh.edu/projects/ lubm/3. Virtuoso。网址 https://virtuoso.openlinksw.com/ 4. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.：SW-store：面向语义网数据的垂直分区数据库管理系统。VLDB Journal **18**(2), 385–406 (2009) 5. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.：使用垂直分区实现可扩展的语义网数据管理。载于：VLDB，第411–422页。ACM (2007) 6. Al-Kateb, M., Sinclair, P., Crolotte, A., Ma, L., Au, G., Nair, S.：优化 Teradata 中的 UNION ALL 连接查询。载于：2017 IEEE 第33届国际数据工程会议 (ICDE)，第1209–1212页。IEEE (2017) 7. Atre, M.：*LeftBit Right*：针对含 OPTIONAL 模式（左外连接）的 SPARQL 连接查询。载于：SIGMOD，第1793–1808页。ACM (2015) 8. Bernstein, P.A., Chiu, D.W.：使用半连接求解关系查询。J. ACM **28**(1), 25–40 (1981) 9. Bonifati, A., Martens, W., Timm, T.：大型 SPARQL 查询日志的分析研究。VLDB Journal**29**(2), 655–679 (2020) 10. Bornea, M.A., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., Bhattacharjee, B.：在关系数据库之上构建高效的资源描述框架存储。载于：SIGMOD，第121–132页。ACM (2013) 11. Chebotko, A., Lu, S., Fotouhi, F.：保留语义的 SPARQL 到 SQL 转换。Data & Knowledge Engineering **68**(10), 973–1000 (2009) 12. Hogan, A., Riveros, C., Rojas, C., Soto, A.：一种针对 SPARQL 的最坏情况最优连接算法。载于：C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. Cruz, A. Hogan, J. Song, M. Lefrançois, F. Gandon (编)，《语义网——ISWC 2019》，第258–275页。Springer International Publishing，Cham (2019) 13. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer：DBpedia——从维基百科提取的大规模多语言知识库。Semantic Web **6**(2), 167–195 (2015) 14. Lei Zou Yue Pang, M.T.Ö., Chen, J.：使用 OPTIONAL 和 UNION 表达式高效执行 SPARQL 查询（完整版本）。技术报告 (2022)。网址 https://anonymous.4open.science/r/gStore-UO/full_version.pdf15. Letelier, A., Pérez, J., Pichler, R., Skritek, S.：语义网查询的静态分析与优化。ACM 数据库系统汇刊 **38**(4) (2013)。DOI 10.1145/2500130。网址 https://doi.org/10.1145/250013016. Mhedhbi, A., Salihoglu, S.：结合二元连接与最坏情况最优连接优化子图查询。Proc. VLDB Endow. **12**(11), 1692–1704 (2019) 17. Neumann, T., Weikum, G.：用于可扩展 RDF 数据管理的 RDF-3X 引擎。VLDB Journal**19**(1), 91–113 (2009) 18. Pérez, J., Arenas, M., Gutiérrez, C.：SPARQL 的语义与复杂度。ACM 数据库系统汇刊 **34**(3), 16:1–16:45 (2009) 19. Prud'hommeaux, E., Bertails, A.：将 SPARQL 映射到传统 SQL (2008)。网址 https://www.w3.org/2008/07/MappingRules/StemMapping#sqlOpt 20. Rao, J., Pirahesh, H., Zuzarte, C.：用于外连接优化的规范抽象。载于：SIGMOD，第671–682页。ACM (2004) 21. Stefanoni, G., Motik, B., Kostylev, E.V.：使用图摘要估算 RDF 数据上合取查询的基数。载于：2018年万维网会议论文集，WWW '18，第1043–1052页。国际万维网会议指导委员会，日内瓦共和国及州，瑞士 (2018)。DOI 10.1145/3178876.3186003。网址 https://doi.org/10.1145/3178876.3186003 22. Wilkinson, K., Sayers, C., Kuno, H.A., Reynolds, D.：jena2 中高效的 RDF 存储与检索。载于：首届 Semantic Web 与数据库国际研讨会论文集，第 131–150 页（2003） 23. Yuan, P., Liu, P., Wu, B., Jin, H., Zhang, W., Liu, L.：Triplebit：一个面向大规模 RDF 数据的快速紧凑系统。VLDB 基金会会刊**6**(7), 517–528 (2013) 24. Zou, L., Mo, J., Chen, L., Özsu, M.T., Zhao, D.：gstore：通过子图匹配回答 SPARQL 查询。VLDB 基金会会刊 **4**(8), 482–493 (2011)

附录

A 实验中使用的查询

A.1 LUBM 上的查询

清单 1：LUBM 查询的前缀

```
前缀 ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>前缀 rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

清单 2：LUBM 上的 q1.1

```
查询 条件 {{
?v 2 1 } { 2 1 } ub:负责人 ?v .
3 2 3
UNION ?v ub:任职于 ?v . ?v ub:本科毕业于 ?v . 4 ?v4 ub:博士毕业于 ?v3 . 5 ?v5 ub:出版物作
者 ?v2 . 6 { 6 1 } { 6 1 } ?v ub:负责人 ?
7 { 2 7 } { 2 7
v . UNION ?v ub:任职于 ?v . ?v ub:负责人 ?v . UNION ?v ub:任职于 ?y .
8 < // 0 0 / http://www.
Department.University.edu UndergraduateStudent91> ub:成员属于 ?v1 . 9 ?v7 ub:名称 ?
v8 . }
```

列表 3：LUBM 上的 q1.2

```
1 { 查询 条件 ?v ub: 电子邮件地址 " 本科生91@系0.大学0.
2 3
edu ". 3 ?v2 ub: 电子邮件地址 ?v1 . 4 OPTIONAL {?v2 ub: 教授课程 ?v4 . ?v3
ub:选修课程 ?v4 . }}
```

清单4：LUBM 上的 q1.3

```
{
1 查询< 条件 http://www.系1.大学0.edu/本科生
363> ub:选修课程 ?v1 . 3 OPTIONAL {?v2 ub:担任助教 ?v1 .
4 OPTIONAL {?v2 ub:成员属于 ?v3 . 5 ?v4 ub:下属组织 ?v3 . 6 ?
v4 ub:下属组织 ?v5 . 7 ?v4 资源描述框架:类型 ?v6 . 8
OPTIONAL {?v5 ub:下属组织 ?v7 . } } } }
```

清单5：LUBM 上的 q1.4

```
1 查询 条件 {2 ?v1 ub:电子邮件地址 " 本科生309@系12.大学0.edu ". 3
OPTIONAL{?v1 ub:成员属于 ?v2 . ?v2 ub:名称 ?v3 . 4 OPTIONAL{?v5 ub:出
版物作者 ?v4 . ?v4 ub:任职于 ?v2 . 5 OPTIONAL{?v6 ub:出版物作者 ?v4 .
} } } }
```

清单6：q1.5（LUBM）

```
1 { 查询 条件 ?v http://www.w3.org/1999/02/22-资源描
2 { 2 < // 4 }
述框架-syntax-ns#type> ?v3 . }3 UNION { 2 } ?v
5 < //
ub:名称 ?v . http://www.系0.大学0.edu/UndergraduateStudent356> ub:
成员属于 ?v1 . 6 ?v2 ub:任职于 ?v1 . 7 OPTIONAL{?v5 ub:导师 ?v2 .
8 { 5 6 } } OPTIONAL?v ub:
9 { 7 2 } }
担任助教 ?v .OPTIONAL?v ub:导师 ?v .
```

清单7：q1.6（LUBM）

```
1 { 查询 条件 ?v ub:负责人 ?v1 .3 <http://www.系1.大学0.edu/
2 4
UndergraduateStudent256> ub:成员属于 ?v1 . 4 ?v3 ub:下属组织 ?v5 .
5 { 2 1 } { 2 1 } ?v ub:任职于 ?
6 { 2 3 } { 2 3
v . UNION ?v ub:负责人 ?v . ?v ub:任职于 ?v . UNION ?v ub:负责人 ?y .
7 { 6 2 OPTIONAL ?v
}
ub:出版物作者 ?v . 8 { { 7 1 } {
OPTIONAL ?v ub:负责人 ?v . UNION ?v7 ub:任职于 ?v1 . } } }
```

清单8：q2.1（LUBM）

```
1 查询 条件 {2
{? s t ub:担任助教 ? course .OPTIONAL {? s t ub:选修课程 ?
3 course2 . ?pub1 ub:出版物作者 ? s t . } }{? prof ub:教授课程 ?
4 course . ? s t ub:导师 ? prof .OPTIONAL {? prof ub:研究兴趣 ? r e s i
5 n t . ?pub2 ub:出版物作者 ? prof . } } }
```

清单9：q2.2（LUBM）

```
1 { 查询 条件 ?pub rdf:类型 ub:出版物 . ?pub ub:出版物
2 {
作者 ? s t . ?pub ub:出版物作者 ? prof . 3 OPTIONAL {? s t ub:电子邮件地址 ?
ste . ? s t ub:电话 ? s t t e l . 4 { } ? s t ub:本科毕业于 ? univ . ?
dept ub:下属组织 ? univ . 5 OPTIONAL {?head ub:负责人 ?dept . ? others
ub:任职于 ?dept . 6 { ? s t ub:成员属于 ?dept . ? prof ub:任职于 ?
dept . 7 OPTIONAL {? prof ub:博士毕业于 ?univ1 . ? prof ub:r e s e a r c h
I n t e r e s t ? resint1 . } } }
```

列表 10：LUBM 上的 q2.3

```
1 { 查询 条件 ?pub ub:出版物作者 ? s t . ?pub ub:出版物作
2 {
者 ? prof . 3 ? s t rdf:类型 ub:研究生 . 4 OPTIONAL {? s t ub:本科毕业于 ?
univ1 . ? s t ub:电话 ? s t t e l . 5 { } ? s t ub:导
师 ? prof . 6 OPTIONAL {? prof ub:博士毕业于 ? univ . ? prof ub:r e s e a r
c h I n t e r e s t ? r e s i n t . 7 { } ? s
t ub:成员属于 ?dept . ? prof ub:任职于 ?dept . ? prof rdf:类型 ub:F u l l P r
o f e s s o r . 8 OPTIONAL {?head ub:负责人 ?dept . ? others ub:任职于 ?
dept . } } }
```

列表 11：LUBM 上的 q2.4

```
1 查询{?x ub:条件职于 <http://www.系0.大学0.edu>. ?x rdf:type ub:
2 F u l l P r o f e s s o r .OPTIONAL {?y ub:导师 ?x . ?x ub:教授课程 ?z . ?
3 y ub:选修课程 ?z . } }
4
```

列表 12：LUBM 上的 q2.5

```
1 { 查询 条件 ?x ub:任职于 http://www.系0.大学12.
2 <
edu>. 3 ?x rdf:type ub:F u l l P r o f e s s o r . 4 OPTIONAL {?y ub:导师 ?
x . ?x ub:教授课程 ?z . ?y ub:选修课程 ?z . } }
```

代码清单 13：LUBM 上的 q2.6

```
1 { 查询 条件 ?x ub:任职于 http://www.系0.大学12.
2 <
edu>. 3 ?x 资源描述框架:type ub:F u l l P r o f e s s o r . 4 OPTIONAL {?x
ub:电子邮件地址 ?y1 . ?x ub:电话 ?y2 . ?x ub:名称 ?y3 . } }
```

A.2 DBpedia 查询

清单14：DBpedia 查询的前缀

```
前缀 rdf : <http : / /www.w3. org /1999/02/22—rdf—syntax—ns#> 前缀
rdfs :  <http : / /www.w3. org /2000/01/ rdf—schema#> 前缀 foaf :  <
http : / / xmlns .com / foaf /0.1/ > 前缀 purl :  <http : / / purl . org /dc /
terms /> 前缀 skos :  <http : / /www.w3. org /2004/02/ skos / core#> 前缀
nsprov :  <http : / /www.w3. org /ns /prov#> 前缀 owl :  <http : / /
www.w3. org /2002/07/owl#> 前缀 dbo :  <http : / / dbpedia . org /
ontology /> 前缀 dbr :  <http : / / dbpedia . org / resource /> 前缀 dbp : <
http : / / dbpedia . org / property />
```

清单15：DBpedia 上的 q1.1

```
查询  条件 {
  { ?v3 rdfs : l abel ?v7 .  } UNION  { ?v3 foaf :名称 ?v7 .  }
{  1  3  }  {  3  1  } ?v
3  4
purl : 主语 ?v . UNION ?v skos : 主语 ?v . ?v rdfs : labe l ?v . ?v5 nsprov :
wasDerivedFrom ?v2 . ?v1 owl : 相同实体 ?v6 . 7 ?v1 dbo : 维基页面链接 dbr :
Economic_system . ?v1 nsprov : wasDerivedFrom ?v2 .  }
```

列表 16：DBpedia 上的 q1.2

```
{ 查询 条件 ?v purl : 主语 ?v5 . OPTIONAL{ ?v5 rdfs :
{  3
labe l ?v6  }  }3 UNION
{  5  3  {  5  6  }  } ?v
1
skos : 主语 ?v . OPTIONAL?v foaf :名称 ?v ?v dbo : 维基页面链接 dbr :
Economic_system . 6 ?v1 nsprov : wasDerivedFrom ?v2 . 7 ?v3 dbo : 维基页
面链接 ?v4 . 8 ?v3 nsprov : wasDerivedFrom ?v2 .  }
```

列表 17：DBpedia 上的 q1.3

```
查询  条件  dbr : Air_masses foaf : 是其主要主题的页面 ?v1 .
3 ?v2 foaf : 是其主要主题的页面 ?v1 .4  { OPTIONAL ?v dbo:
5  2
wikiPageRedirects ?v3 . ?v4 foaf : 主要主题 ?v2 .6  {
7  5
OPTIONAL?v dbo : 维基页面链接 ?v3 . 8 OPTIONAL{ ?v6 dbo :
wikiPageRedirects ?v5 . 9 OPTIONAL{ ?v6 dbo : 维基页面链接 ?v7 .
} } } } }
```

列表 18：DBpedia 上的 q1.4

```
{ 查询 条件 dbr : Functional_neuroimaging purl : 主语 ?
v1 .3  { OPTIONAL?v owl :相同实体 ?v2 . ?v1 资源描述框架 : 类型 ?v3 .
4  1
?v4 owl : 相同实体 ?v2 . ? v5 skos : related ?v4 . 5 OPTIONAL{ ?v6 skos :
related ?v4 .  }6  { OPTIONAL?v purl : 主语 ?v1 .  } UNION  { ?v1
7  {  7
skos : 主语 ?v7 .  }8  { OPTIONAL?v purl : 主语 ?v8 .  } UNION  { ?
9  {  7
v8 skos : 主语 ?v7 .  } } } }
```

列表 19：DBpedia 上的 q1.5

```
{ 查询 条件 ?v purl : 主语 ?v3 .  } UNION  { ?v2 dbo: 维
{  2
基页面链接 ?v  4  } . ?v dbo: 维基页面链接 dbr : Abdul_Rahim_
3  1
Wardak . 4 ?v2 dbo: 维基页面链接 ?v1 . 5 OPTIONAL{ ?v5 owl : 相同实体 ?v2 .
{  5  6  } } OPTIONAL?v dbo :
{  2  7  } }
wikiPageLength ?v . OPTIONAL?v skos : prefLabel ?v .
```

列表 20：DBpedia 上的 q1.6

```
查询  条件 {
{ ?v2 foaf : 主要主题 ?v1.  } UNION  { ?v1 foaf :
是其主要主题的页面 ?v2 .  }
{ ?v2 foaf : 主要主题 ?v3.  } UNION  { ?v3 foaf :
是其主要主题的页面 ?v2 .  }
?v1 dbo: 维基页面链接 dbr : Category : Cell_biology .
?v3 dbo: 维基页面链接 ?v1 .
OPTIONAL{
{ ?v2 foaf : 主要主题 ?v4.  } UNION  { ?v4 foaf :
是其主要主题的页面 ?v2 .  } }
OPTIONAL{ ?v5 dbo: 门 ?v3 . ?v6 dbo: 门 ?v3 .
OPTIONAL{
{ ?v7 foaf : 主要主题 ?v5.  } UNION  { ?v5 foaf :
是其主要主题的页面 ?v7 .  } } } }
```

列表 21：DBpedia 上的 q2.1

```
查询6 a dbo : 聚居地 . ?v6 dbo : a b s t r ac t ?v1 . ?v6 rdfs : l a be l ?v2 . ?v6 geo : l
a t ?v3 . ?v6 geo : long ?v4 . 4 OPTIONAL  { ?v6 foaf : 描绘图像 ?v8 .  } OPTIONAL
条件
{  6  10  } ?v foaf : 主页 ?v . OPTIONAL ?v dbo : 总
{  6
人口 ?v  12  } . OPTIONAL ?v dbo : 缩略
7  {  6  14  } }
图 ?v .
```

列表 22：DBpedia 上的 q2.2

```
查询3 foaf 条件 页 ?v0 . ?v3 a dbo : 足球运动员 . ?v3 dbp : 位置 ?v6 . ?v3 dbp :
俱乐部 ?v8 . ?v8 dbo : 容量 ?v1 . ?v3 dbo : 出生地 ?v5 . 4 OPTIONAL  { ?v3
dbo :number ?v9 .  } }
```

列表 23：DBpedia 上的 q2.3

```
{
SELECT  WHERE  ?v dbo : 缩略图 ?v4 . ?v5 资源描述框架 : 类型
dbo : 人物 . ?v5 rdfs : l abe l ?v . 3 ?v5 foaf : 主页 ?v8 . 4 OPTIONAL  { ?v5
foaf : 主页 ?v10 .  } }
```

列表 24：DBpedia 上的 q2.4

```
查询  条件 {
  { ?v2 a dbo : 居民点 . ?v2 rdfs : l ab el ?v . ?v6 a dbo : 机场 . ?v6
dbo : c i t y ?v2 . ?v6 dbp : i a t a ?v5 . 4 OPTIONAL  { ?v6 foaf : 主页 ?
v7 .  }  }OPTIONAL  { ?v6 dbp : 本地名称 ?v8 .  } }
```

列表 25：DBpedia 上的 q2.5

```
查询 条件 {?v4 skos : 主语 ?v . ?v4 foaf :名称 ?v6 .
OPTIONAL  { ?v4 rdfs :注释 ?v8 .  } }
```

列表 26：DBpedia 上的 q2.6

```
{ 查询 条件 ?v rdfs : 注释 ?v1 . ?v0 foaf : 页面 ?v .
0
{  0  6  } OPTIONAL ?v skos : 主语 ?v . OPTIONAL ?v dbp : 行业 ?v .
{  0  5  }
{  0  2  } OPTIONAL ?v dbp : 位置 ?v .
}
{  0  OPTIONAL ?v dbp : 所在国家 ?v
3  } . OPTIONAL ?v dbp : lo c a t i o n Ci t y ?v . ?a dbp : 制造
{  0  9
商 ? v  0  } . OPTIONAL  {  0  11  0  } ?v
8  9  {  0  10
dbp : 产品 ?v . ?b dbp : 型号 ?v . OPTIONAL ?v georss : 坐标点 ?v .
{  0  7  } }  OPTIONAL ?v rdf : 类型 ?v .
```