

点不相交划分的 分布式RDF系统

Vertex-disjoint Partitioning-based
Distributed RDF Systems



Peng Peng
Hunan University, China
hnu16pp@hnu.edu.cn



目录

01

背景

02

最小属性划分

03

最小模块划分

04

结论





湖南大学
HUNAN UNIVERSITY

第一部分

背景

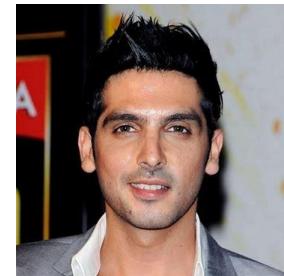


资源描述框架

这是一个被广泛用于知识库的数据模型

一切都是一个具有唯一名称的资源
可以定义资源的属性
可以定义与其他资源的关系

dbpedia:Zayed_Khan



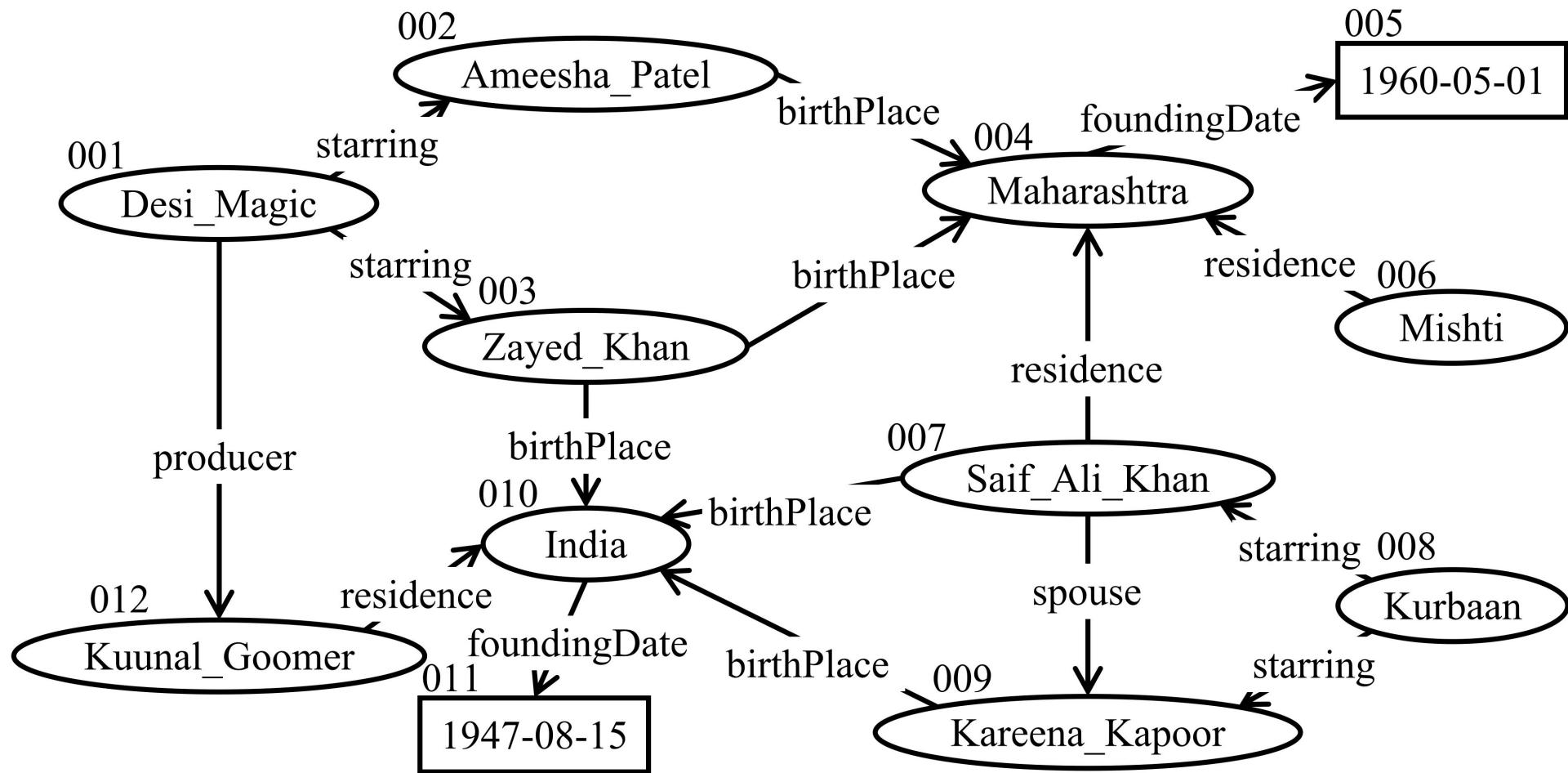
dbpedia:name "Zayed Khan"@en
dbpedia:dateOfBirth "1980-07-05"

dbpedia:birthPlace



dbpedia:Maharashtra

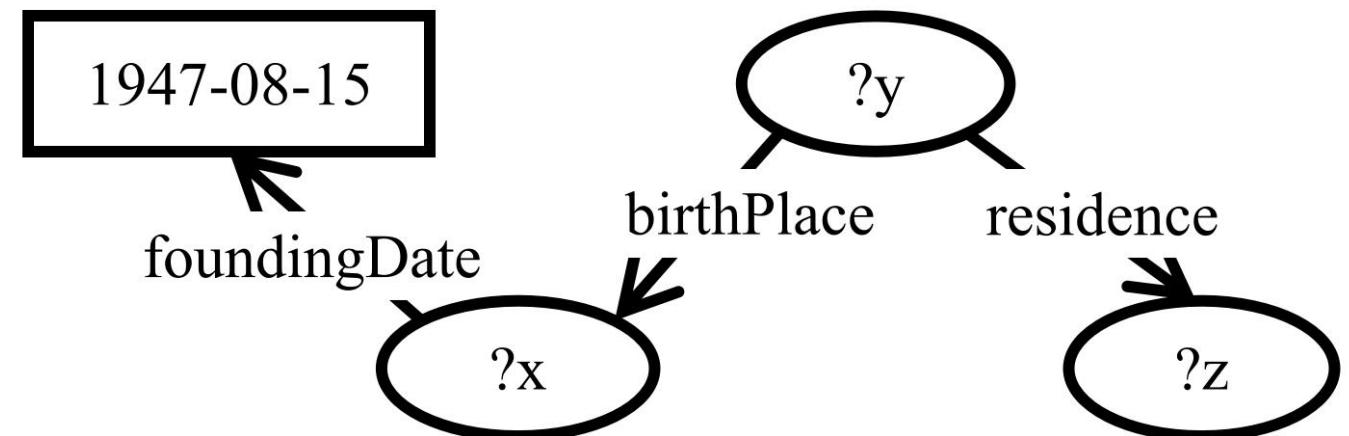
一个RDF数据集可以表示为一个图



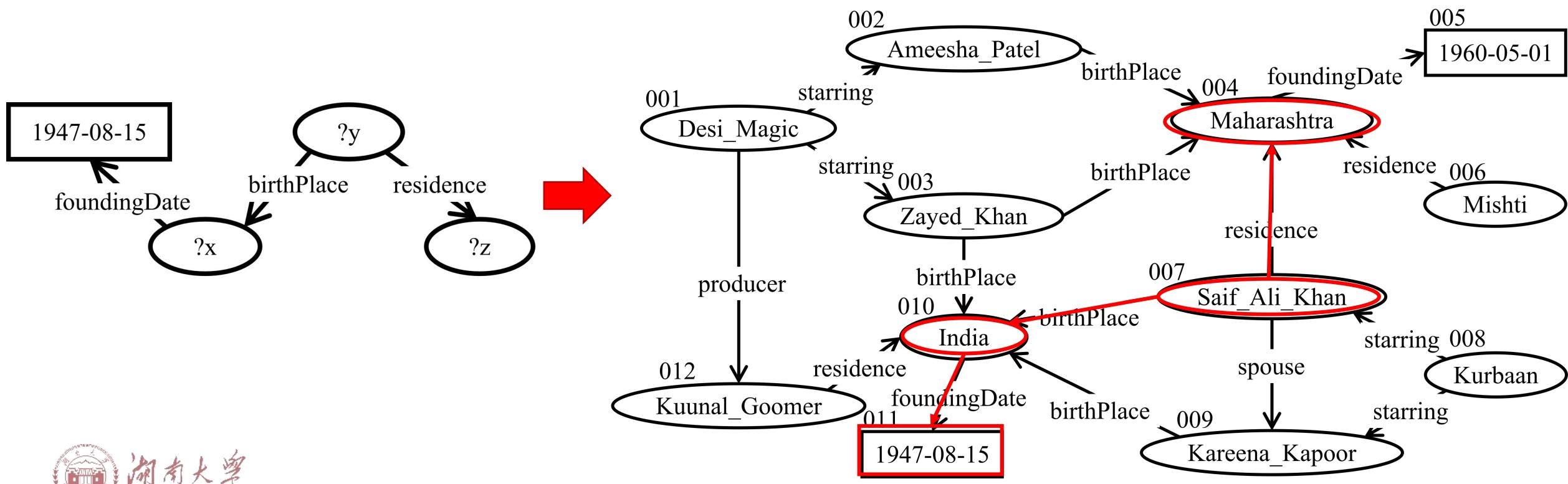
查询模型 - SPARQL协议和RDF查询语言

SPARQL查询是一组带有变量的三元模式

```
Select ?x where {  
    ?y      residence      ?z .  
    ?y      birthPlace     ?x .  
    ?x      foundingDate   1947-08-15 . }
```



回答SPARQL查询等同于使用同态子图匹配来进行子图匹配



当前，RDF数据集变得越来越大



Max-Planck-Institute

284 million triples



Metaweb Company
acquired by Google in 2010

2.4 billion triples



Leipzig University
University of Mannheim
OpenLink Software

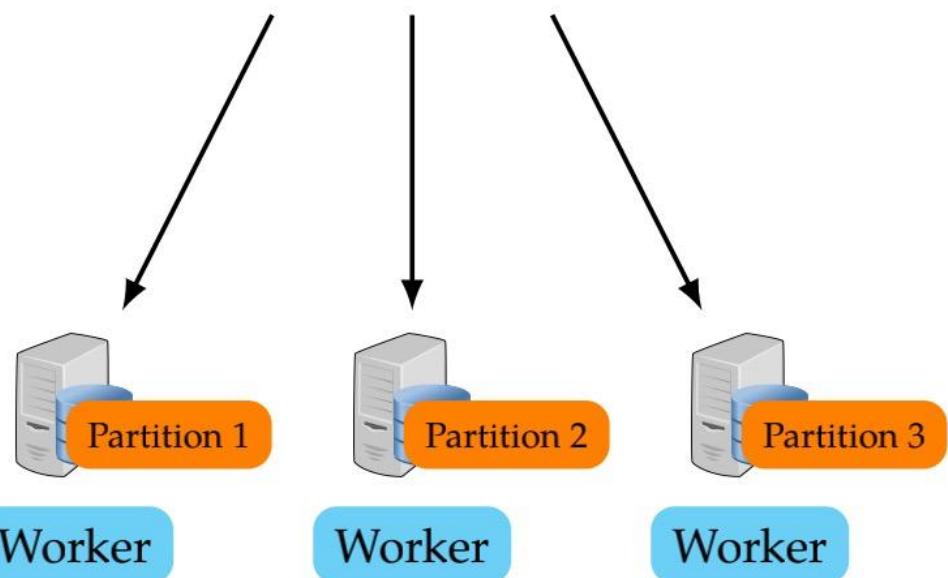
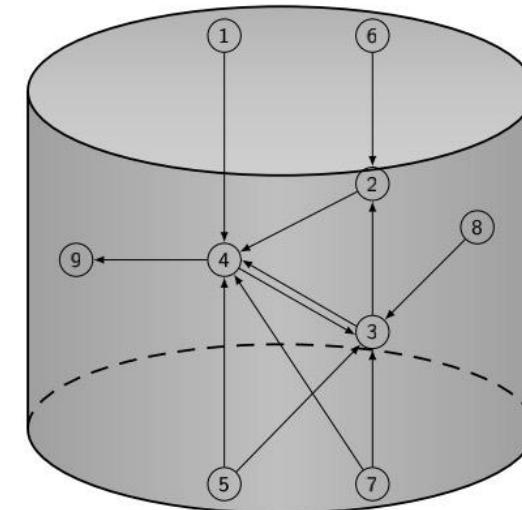
9.5 billion triples

设计一个分布式RDF系统来管理大型RDF数据集是非常重要的！

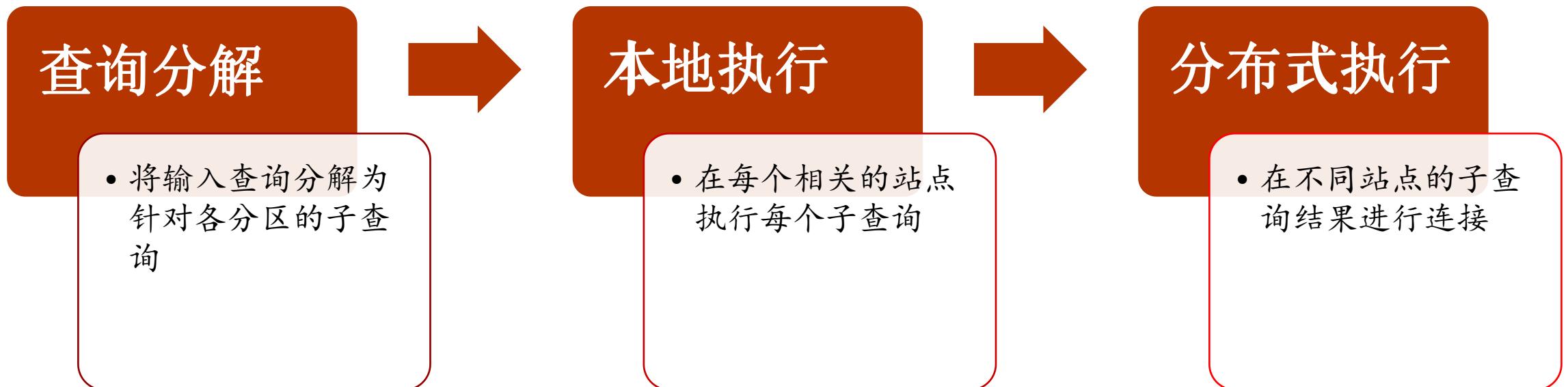
基于分区的架构

一个RDF图通过不同的站点进行分区

目标：尽可能减少站点间通信，实现并行化查询处理。



以尽量减少分区间连接的方式对数据和查询进行分区



RDF图划分方法可以分为三种类型：

- 点不相交：将每个顶点放入一个分区。现有方法的目的是最小化边切割，而不是最小化分区间连接
- 边不相交：将每条边放入一个分区。现有方法在许多基于云的系统中广泛使用，以剪枝不相关的分区并避免在云中进行过多的扫描，但并不专注于避免分区间连接
- 其他：考虑额外信息，如查询日志

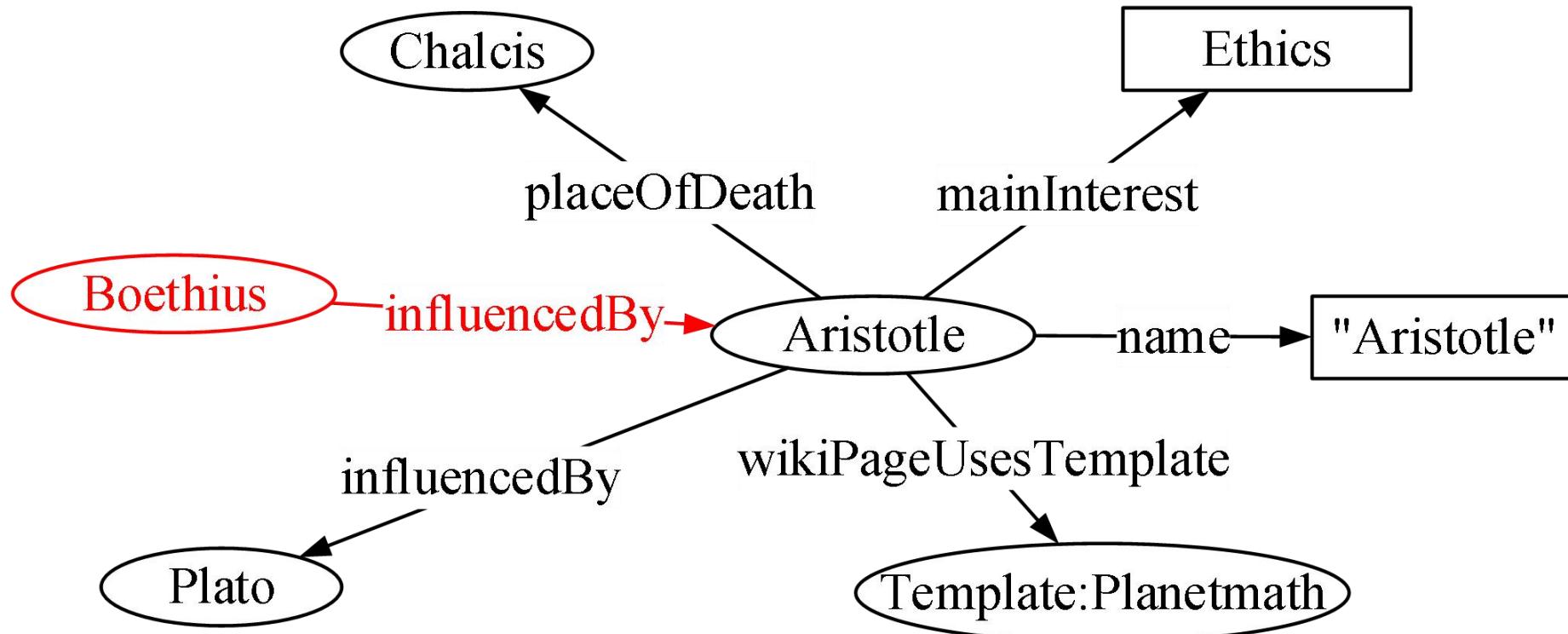
H-RDF-3X [Huang et al., VLDB 2011]

- 这个系统主要思想是将现有知识图谱数据图根据其结构信息进行水平划分
- 它对知识图谱的划分使用现有成熟工具METIS来实现。划分出来每个RDF数据块对应一个数据分区
- 在此过程中，为了提高数据的局部性，每个块除了保存自身所有的点之外，还保存了它们的n步邻居以内所有点和边的副本

SHAPE [Lee et al., VLDB 2013]

- SHAPE 中，在进行知识图谱数据划分的时候，SHAPE 划分的基本单元为一个三元组群（Triple Group）
- 所谓三元组群，其实就是图上的星状结构
- SHAPE 提出了三种三元组群：只含中心点出边的三元组群、只含中心点入边的三元组群、含中心点出边与入边的三元组群

SHAPE [Lee et al., VLDB 2013]



黑色部分是只含中心点出边的三元组群；红色部分是只含中心点入边的三元组群；整个就是含中心点出边与入边的三元组群

SHAPE [Lee et al., VLDB 2013]

- LUBM是基于大学领域的RDF基准测试数据集，其中所有IRI都是按照大学院系来组织，比如

<http://www.Department1.University2.edu/FullProfessor2/Publication14>

- 于是SHAPE可以将不同学校的不同实体放到不同分区



第二部分

最小属性划分



我们提出了一种新颖的顶点不相交的RDF图划分方案，即最小属性划分（Minimum Property-Cut, MPC），用于分布式SPARQL查询处理

动机

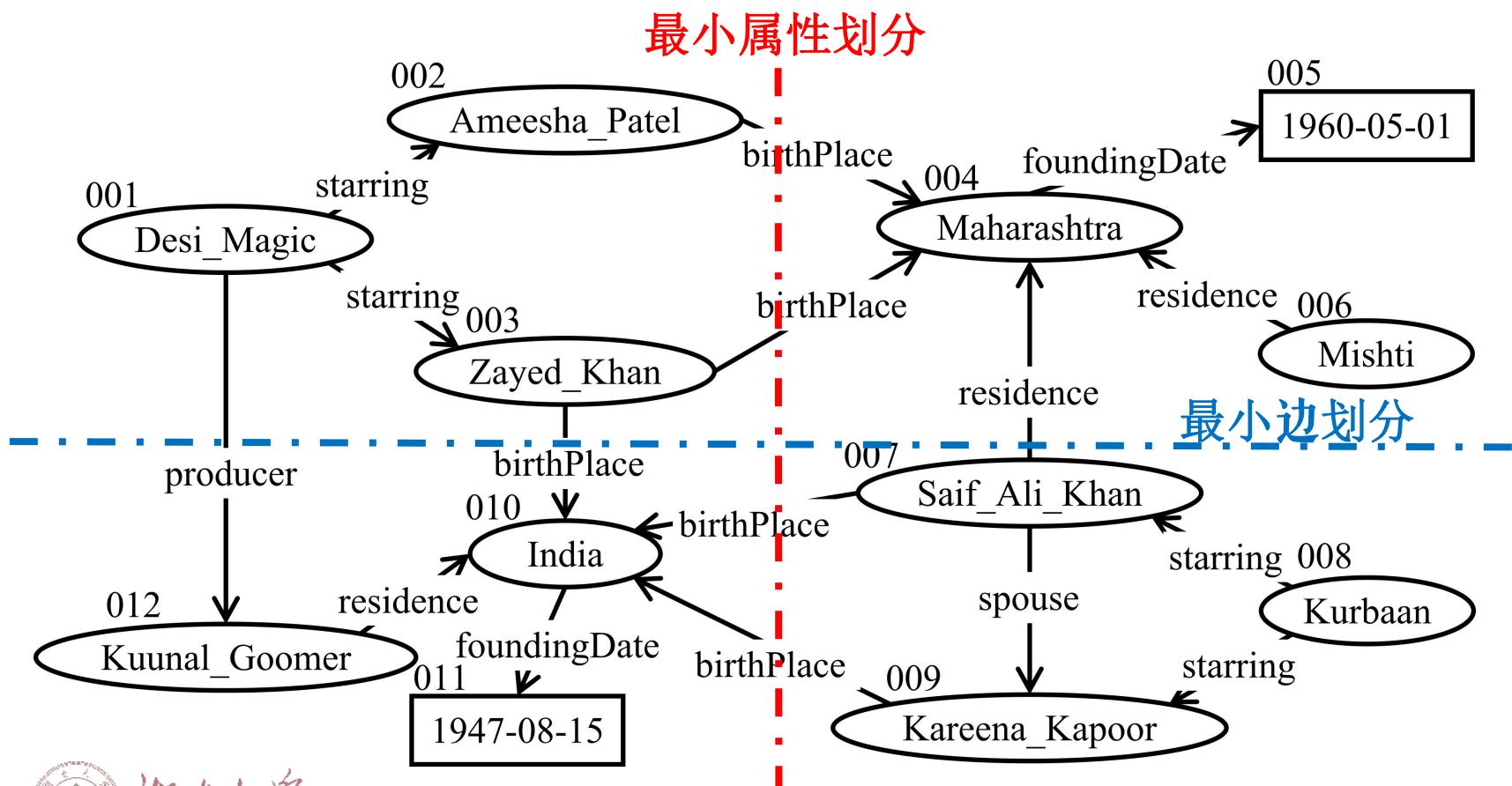
分布式SPARQL执行中的主要性能瓶颈是分区间连接

如果一个查询中的所有属性都是内部的 → 在每个分区上独立执行
而不进行分区间连接

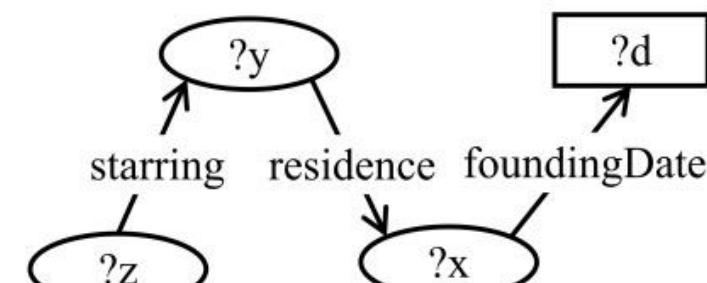
目标：分区以最大化内部属性的数量

可能会增加边切割，但最小化了唯一属性切割的数量

比较不同的划分方法



划分示例



查询示例

问题定义

给定一个RDF图G和一个正整数k，G的最小属性划分（MPC）分区

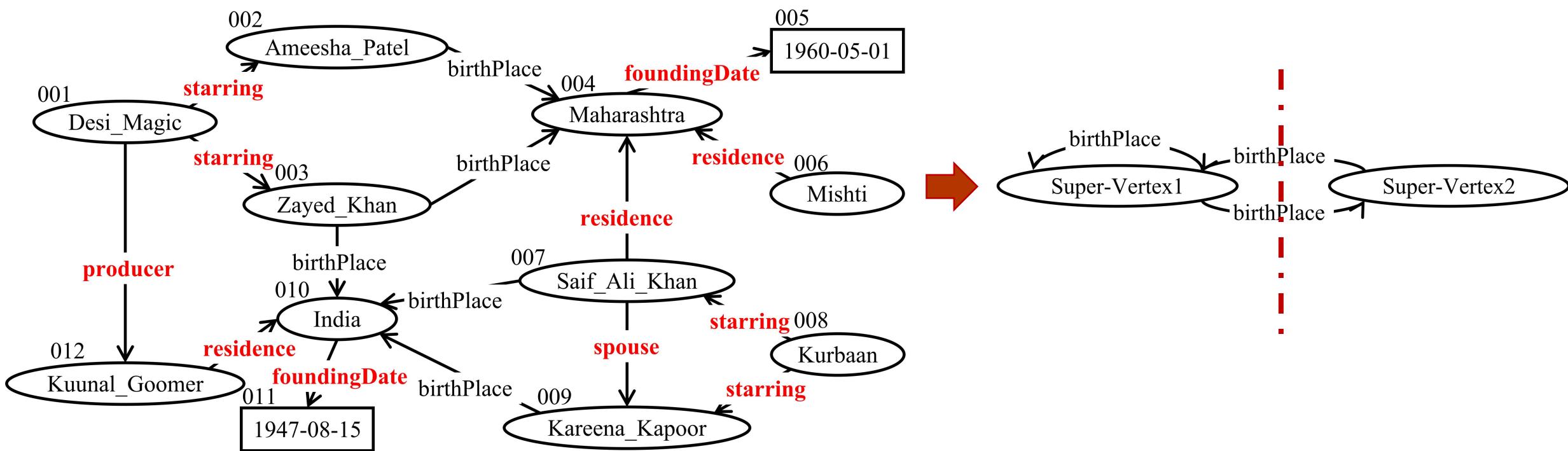
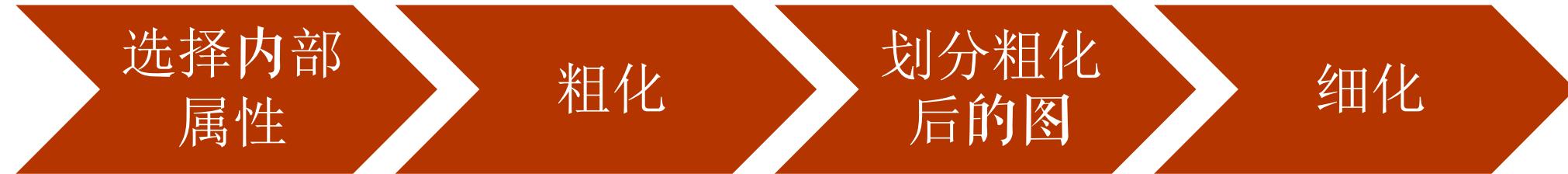
$F = \{F_1, F_2, \dots, F_k\}$ 是这样的一个分区，使得

- 交叉属性数 $|L_{cross}|$ 最小（即内部属性数 $|L_{in}|$ 最大）；
- 每个 F_i 的大小（即 $|V_i|$ ）不大于 $(1 + \varepsilon) \times |V|/k$ ，其中 ε 是用户定义的分区的最大不平衡比（即分区相对大小的差异程度）。

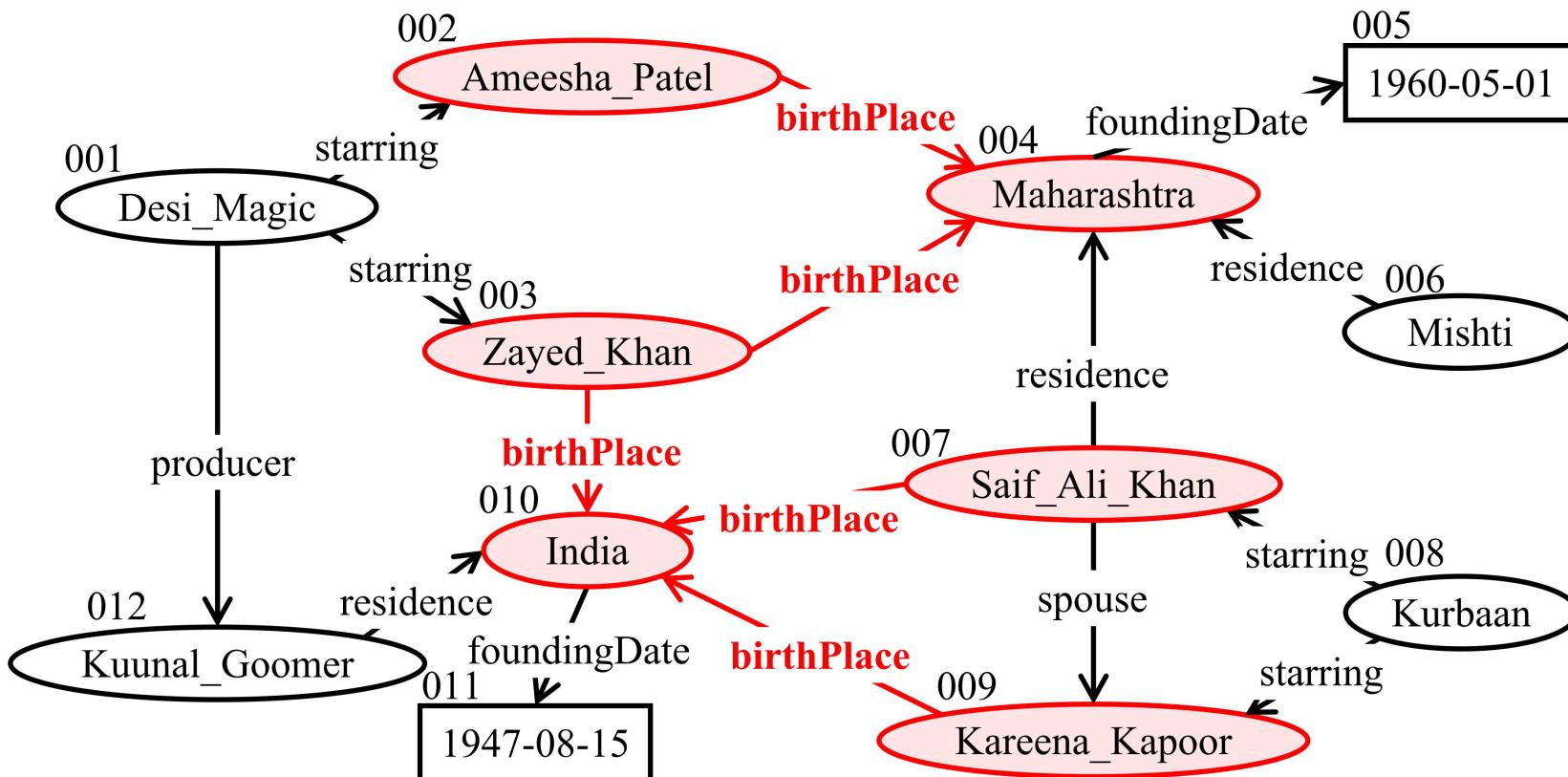
定理：MPC分区问题是NP-完全问题。

证明：我们将NP-完全的最小边切割问题归约为MPC问题，方法是为最小边切割图分区实例中的每条边分配一个独特的属性。

最小化谓词切割框架



如果属性p是内部属性，那么在其诱导子图的弱连通分量中的任意两个顶点应该在一个分区中。



如果我们希望
birthPlace成为一个
内部属性，那么所有
6个顶点都应该在一
个分区中。

给定一组属性 L' ，选择它们作为内部属性的成本定义为属性诱导子图 $G[L']$ 中最大 WCC 的大小，

$$Cost(L') = \max_{c \in WCC(G[L'])} |c|$$

其中 c 是 $WCC(G[L'])$ 中的一个弱连通分量， $|c|$ 表示 c 中顶点的数量。

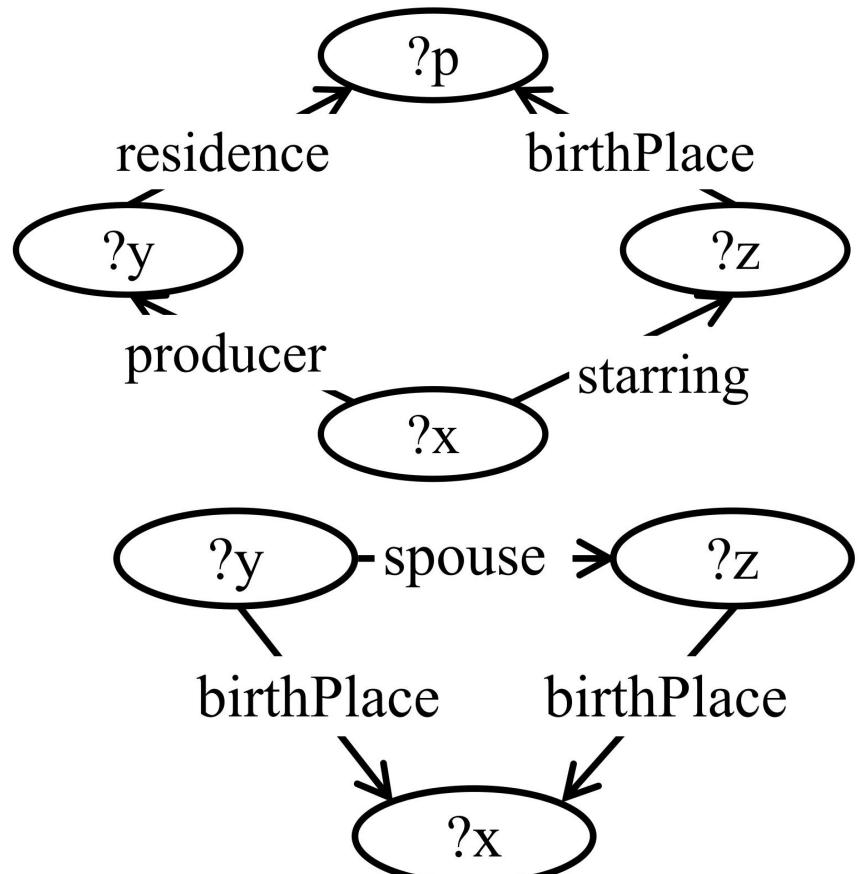
1. 首先，我们为内部属性初始化一个空集 L_{in} ，并为每个属性计算属性诱导子图的WCCs。
2. 我们迭代地选择一个属性 p ，使其最小化 $\text{Cost}(L_{in} \cup \{p\})$ 并将其插入到 L_{in} 中。这些步骤重复执行，直到无法再选择更多的属性为止。

为了计算和合并WCCs，我们建议使用不相交集合森林数据结构。

独立可执行的查询（可独立执行查询）是指在某个分区中是“局部”的查询，无需与其他分区进行连接就可以执行。

内部可独立执行查询s：不包含任何交叉属性的查询

扩展可独立执行查询s：移除交叉谓词边后连接的查询

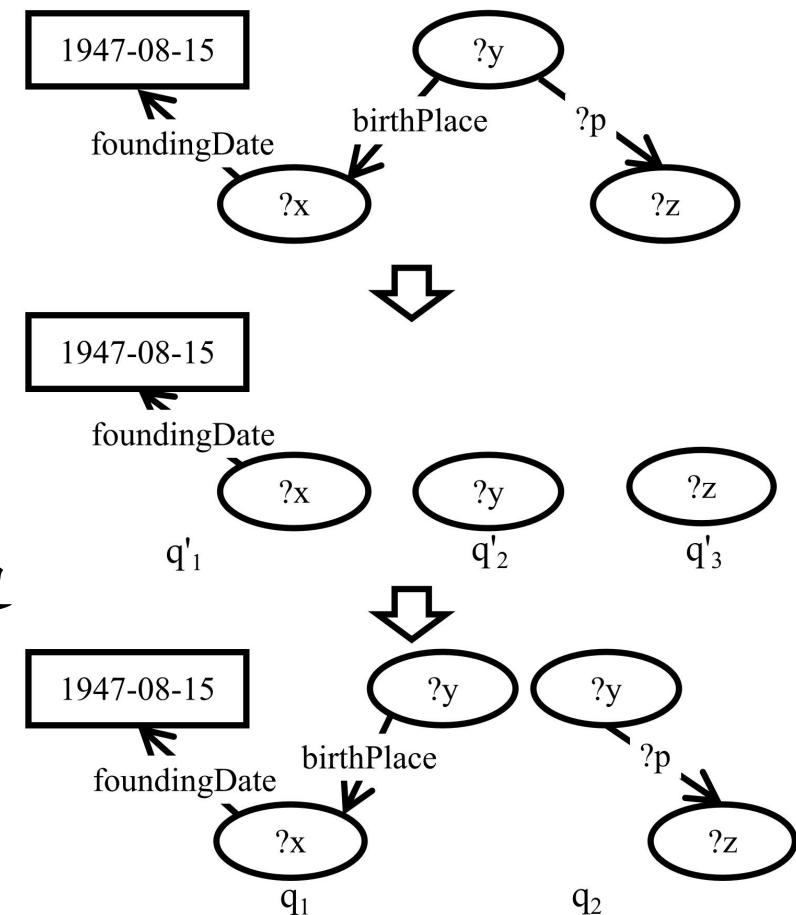


非可独立执行查询：我们分解成一组可独立执行查询并连接结果

□ 移除交叉属性边和带有变量的边

□ 形成子查询

□ 向子查询中添加交叉属性边和带有变量的边



- 当接收到一个非可独立执行查询查询Q时，它会被分解成一组子查询 $\{q_1, q_2, \dots, q_y\}$ ，并将这些子查询发送到每个分区进行独立执行。
- 然后将子查询的匹配项进行连接，以获得最终结果。

□ 数据集：

Dataset	#Entities	#Triples	#Properties
LUBM 100M	17,473,142	106,909,064	18
LUBM 1B	173,891,493	1,069,331,221	18
LUBM 10B	1,737,718,408	10,682,013,023	18
WatDiv 100M	5,212,745	108,997,714	86
WatDiv 1B	52,120,745	1,099,208,068	86
WatDiv 10B	521,200,745	10,987,996,562	86
YAGO2	21,073,153	284,417,966	98
Bio2RDF	804,671,979	4,426,591,829	1,581
DBpedia	139,493,254	1,111,481,066	124,034
LGD	311,153,753	1,292,933,812	33,348

- 竞争对手：Subject_Hash、METIS和VP
 □ 环境：在阿里云上运行Linux的8台机器

□ $|L_{cross}|$ 和 $|E^c|$:

Datasets	MPC		Subject_Hash		METIS	
	$ L_{cross} $	$ E^c $	$ L_{cross} $	$ E^c $	$ L_{cross} $	$ E^c $
LUBM	5	29,971,560	14	62,377,786	13	21,853,766
WatDiv	17	95,497,642	31	95,786,527	31	58,100,544
YAGO2	5	128,758,514	45	142,274,454	43	58,912,138
Bio2RDF	36	2,044,500,633	398	2,184,075,117	-*	-
DBpedia	64	504,897,118	33,966	681,734,289	17,807	171,944,344
LGD	6	518,035,967	2,012	676,620,154	2,010	296,443,817

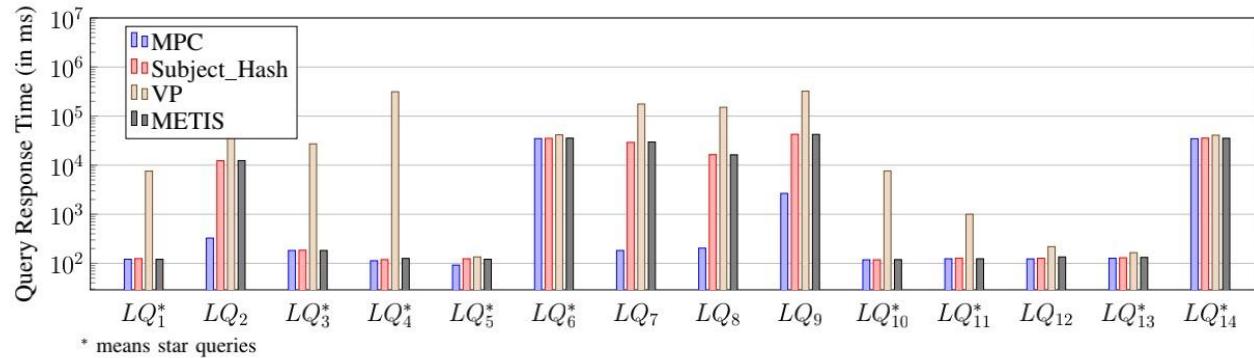
* - means that data size is beyond the capacity of METIS.

□ 可独立执行查询s的百分比:

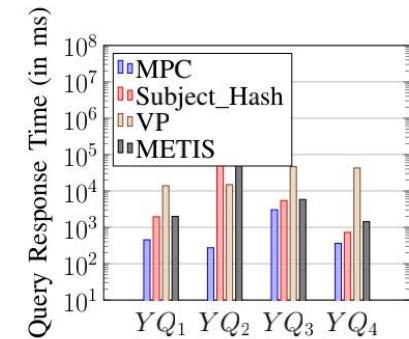
	MPC	VP	Subject_Hash / METIS	Subject_Hash+	METIS+
LUBM	100%	28.57%	71.43%	71.43%	71.43%
WatDiv	60%	0%	50%	50%	50%
YAGO2	100%	0%	0%	0%	0%
Bio2RDF	100%	40%	80%	80%	80%
DBpedia	75.19%	24.25%	46.87%	51.87%	51.90%
LGD	99.95%	83.51%	96.95%	96.98%	96.98%

在线性能比较

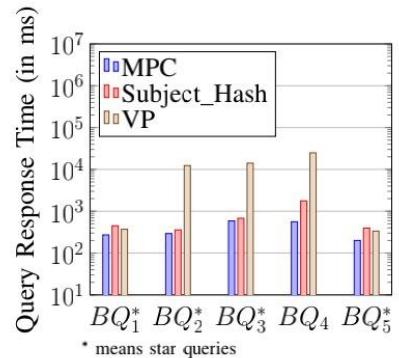
□ 在基准查询上：



(a) LUBM

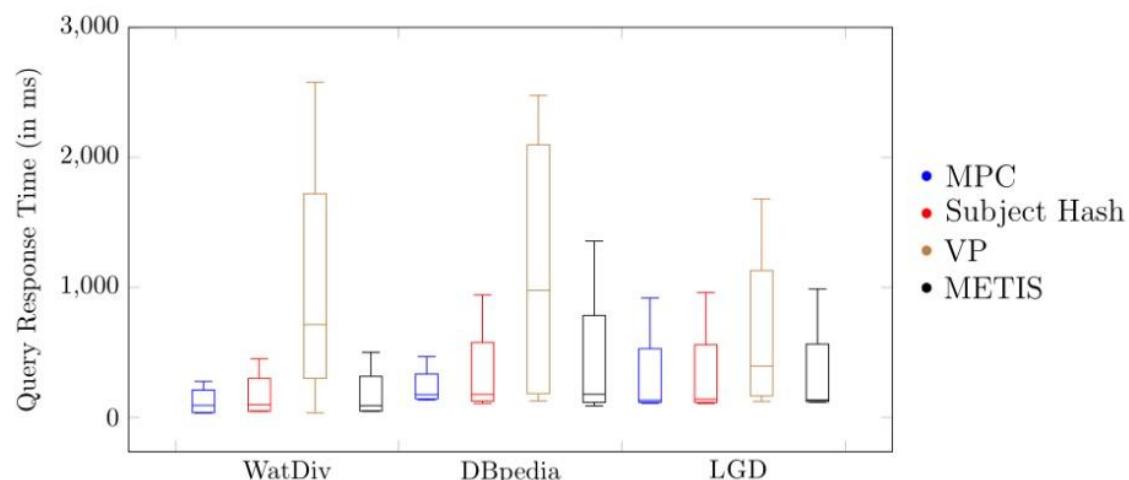


(b) YAGO2



(c) Bio2RDF

□ 在真实查询日志上：





第三部分

最小模块划分



本工作目标是考虑到查询日志特征和RDF图结构的分区技术

划分目标：最大限度地减少了给定查询日志中具有分区间连接的查询数量，进而提高查询执行性能。

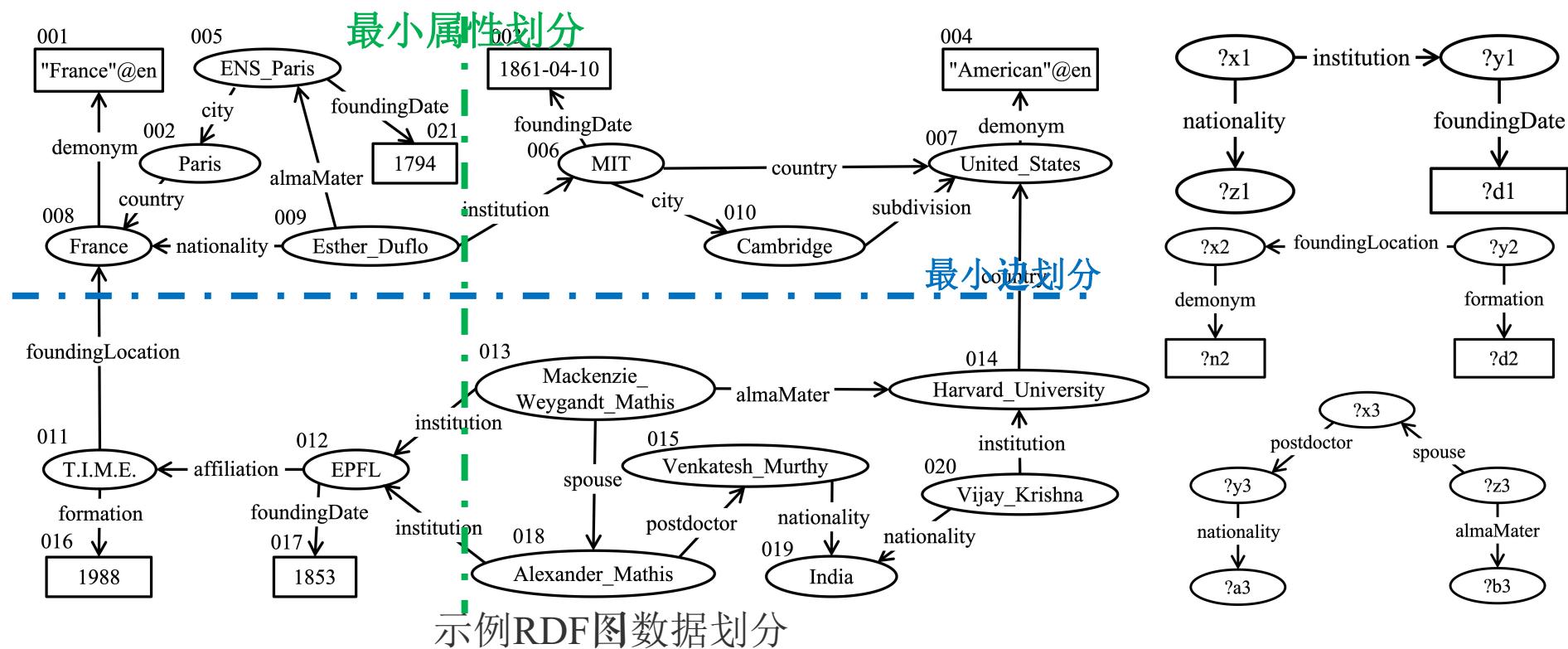
本划分也采用点不相交的划分方法，将每个顶点分配给单个分区。给定RDF图G上的点不相交划分，SPARQL查询的匹配分为两类：**内部匹配**和**跨界匹配**

内部匹配完全包含在单个分区中，而跨界匹配跨越多个分区

当用户提交查询时，如果我们可以确定它只有内部匹配，则可以在每个分区上独立地处理它；否则，处理该查询就会涉及分区间连接

背景和动机

当考虑到查询日志时，之前方法不一定能消除或减少对分区间连接的需求

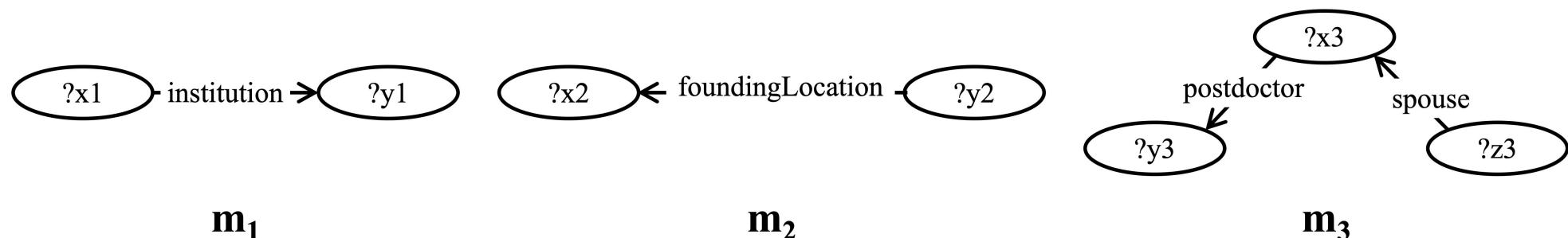


我们提出了一种最小模块划分 (Minimum Motif-Cut, MMC) 的划分策略，该策略考虑了查询日志特征，并最大化了查询日志中的可独立执行查询数量 $W = \{Q_1, \dots, Q_r\}$

所谓模块，就是通过移除查询日志查询中度为1的点并用变量替换所有常量（字符串和URI）所形成的结构

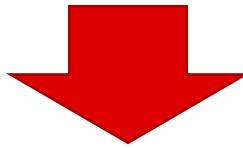
如果一个模块的匹配不涉及任何跨界边，则称该模块为**内部模块**；否则，称为**跨界模块**

如果一个查询对应模块是内部模块，那么这个查询不涉及跨界匹配



示例查询日志的示例模块

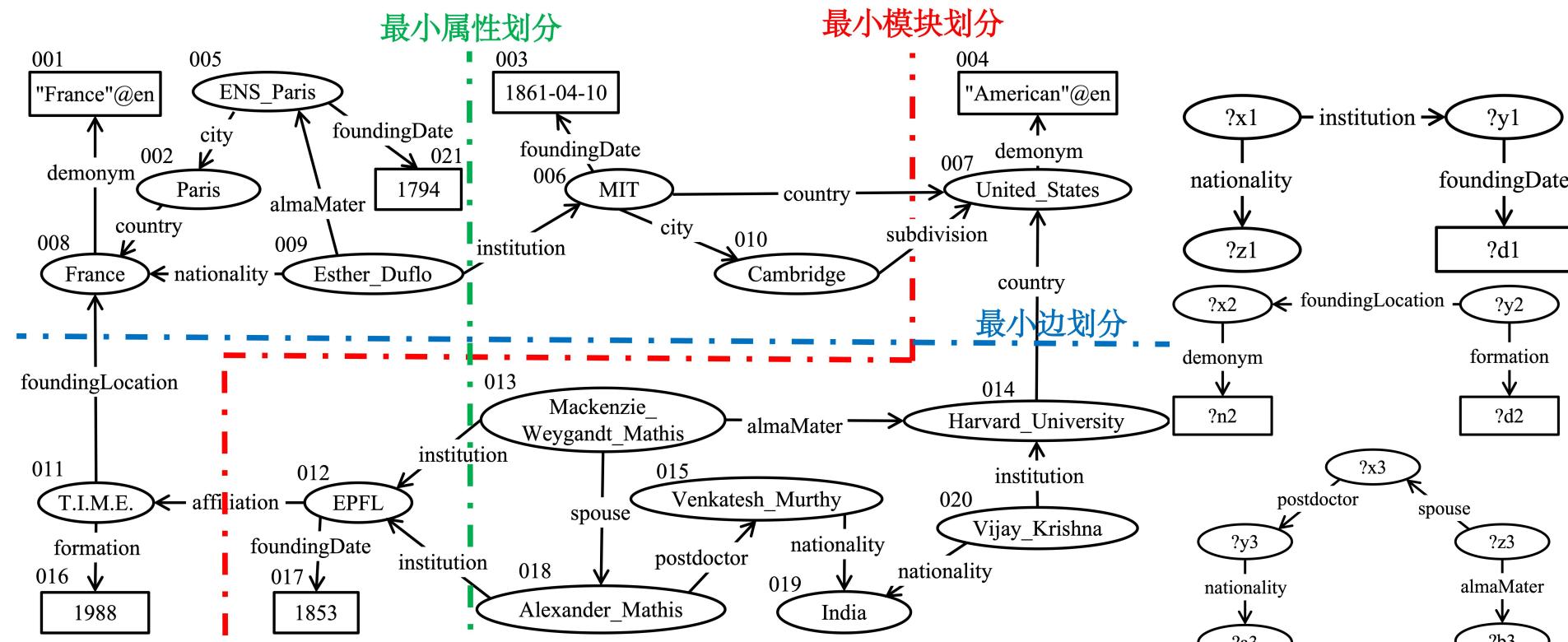
我们提出了一种最小模块划分 (Minimum Motif-Cut, MMC) 的划分策略，该策略考虑了查询日志特征，并最大化了查询日志中的可独立执行查询数量



我们提出了一种最小模块划分 (Minimum Motif-Cut, MMC) 的划分策略，该策略考虑了查询日志特征，并最小化了查询日志中跨界模块数量（等价于最大化内部模块数量）

最小模块划分 (Minimum Motif-Cut)

最小模块划分示例



示例RDF图数据划分

示例查询日志



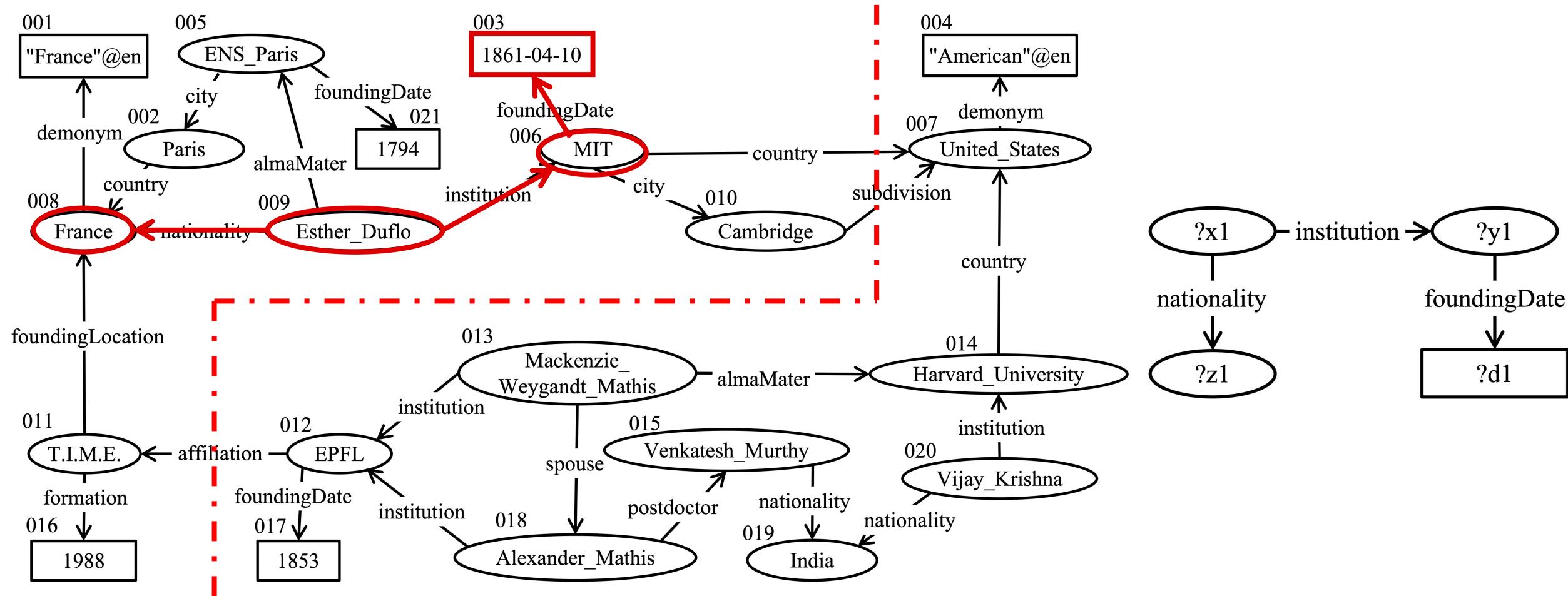
湖南大学
HUNAN UNIVERSITY

SPARQL查询的匹配分为两类：内部匹配和跨界匹配。内部匹配完全包含在单个分区中，而跨界匹配跨越多个分区

内部匹配可以进一步分为两类：给定一个内部匹配 μ ，如果存在一个分区 G_i ，并且 μ 中的所有边都是 G_i 中的内部边，则 μ 是一个**完全内部匹配**；否则， μ 会涉及一些跨界边，称为**扩展内部匹配**

给定一个可独立执行查询 Q ，如果 Q 的所有匹配都是完全内部匹配，则 Q 是一个**内部可独立执行查询**；否则， Q 的一些匹配是扩展的内部匹配， Q 被称为**扩展可独立执行查询**

完全内部匹配示例



定理1

给定一个查询 Q ，去掉所有的1度顶点后，剩下的部分记为 Q^* 。如果 Q^* 是一个内部可独立执行查询，则 Q 是一个可独立执行查询。

定义1（模块）

查询日志W中的一个查询Q对应于一个模块m，其中m是通过删除一度顶点并将Q中主语和宾语处的所有常量（字面和URI）替换为变量生成的。

因此，给定一个查询日志 W ， W 中的每个查询对应于一个模块，所有模块的集合表示为 M

通常，一个模块概括了一个特定的查询，是一个抽象的查询模板，而一个查询是一个模块的实例

定义2（内部/跨界模块）

给定一个RDF图G的划分和一个模块m，如果 $MS(m)$ 中的所有匹配都是完全内部匹配，那么我们称m是一个**内部模块**；否则，m是一个**跨界模块**

给定一组模块M和一个RDF图G的划分，让 M_{in} 和 M_{cross} 分别表示M中的所有内部模块和跨界模块，于是 $M = M_{in} \cup M_{cross}$ 且 $M_{in} \cap M_{cross} = \emptyset$

定义3（最小模块划分）

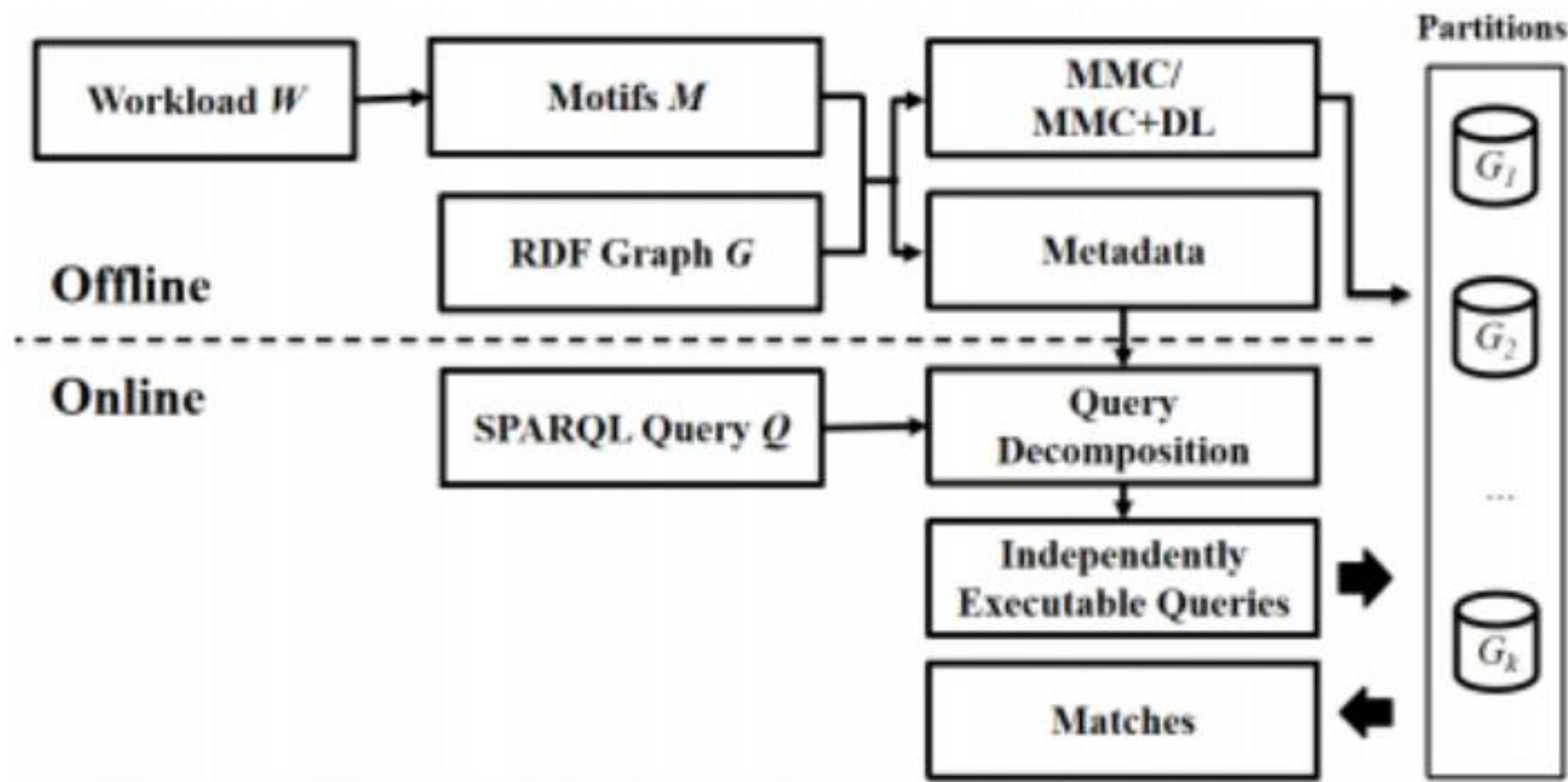
给定一个RDF图G，来自查询日志W的一组模块M和一个正整数k，G的最小模块切割分区是一个分区 $P = \{G_1, G_2, \dots, G_k\}$ ，使得：

- (1) 内部模块的数量 $|M_{in}|$ 最大化（即，跨界模块的数量 $|M_{cross}|$ 最小化）；
- (2) 分区的大小大致平衡： $|V_i| \leq (1+\varepsilon) \times |V|/k$ 对于每个 G_i ，其中 ε 是用户定义的最大分区不平衡比例（即，分区相对大小可以有多少差异）。

定理2 MMC分区问题是NP完全的

证明：我们将NP完全最小边切割问题简化为最小模块切割问题。

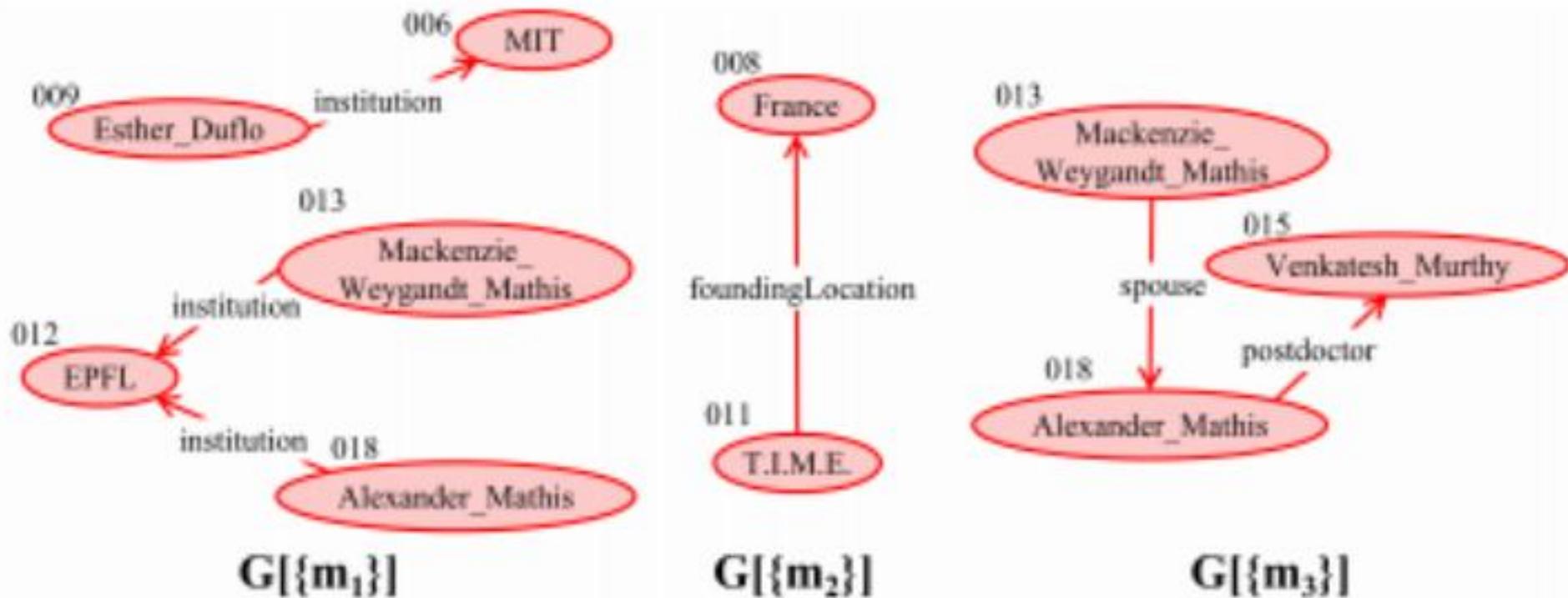
方法概述图



定义4（模块诱导子图）

给定一组模块 $M' \subseteq M$ ，由 M' 诱导的 G 的模块诱导子图，记为 $G[M']$ ，是由至少一个 $MS(m)$ 中的匹配中的边形成的子图，其中 $m \in M'$ ，即 $G[M']$ 是由 $U_{m \in M'}$ 中的所有边形成的（它是 $(\bigcup_{\mu \in MS(M)} E(\mu))$ ，其中 μ 是模块 m 的一个匹配， $E(\mu)$ 表示 μ 中的所有边）。

示例模块诱导子图



定理3

设 M_{in} 为内部模块的集合， $G[M_{in}]$ 为由 M_{in} 诱导的子图。 $G[M_{in}]$ 的 WCC 中的任意两个顶点必须在同一个分区中。

证明：假设 $G[M_{in}]$ 的 WCC 中的两个顶点 u 和 v 不在同一个分区中。由于 M_{in} 是内部模块的集合， $G[M_{in}]$ 中的所有边都是内部边。另一方面，由于它们在同一 WCC 中，至少存在一条连接它们的弱连通路径 π 。由于它们不在同一个分区中，路径 π 中至少有一条边跨越了两个分区，即它是跨界边。这与路径 π 中的所有边都是内部边的事实相矛盾。

定义5(选择内部模块成本)

给定一组模块 $M' \subseteq M$, 选择 M' 作为内部模块的成本为:

$$Cost(M') = \max_{c \in WCC(G[M'])} |c|$$

其中 $WCC(G[M'])$ 是 $G[M']$ 中的 WCC 集合, c 是 $WCC(G[M'])$ 中的一个 WCC , $|c|$ 表示 c 中的顶点数量。

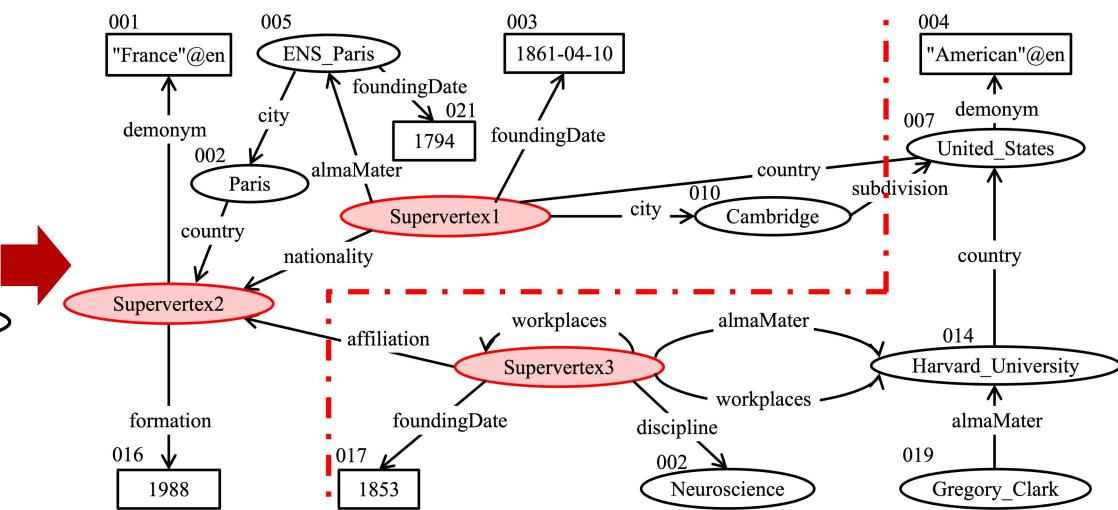
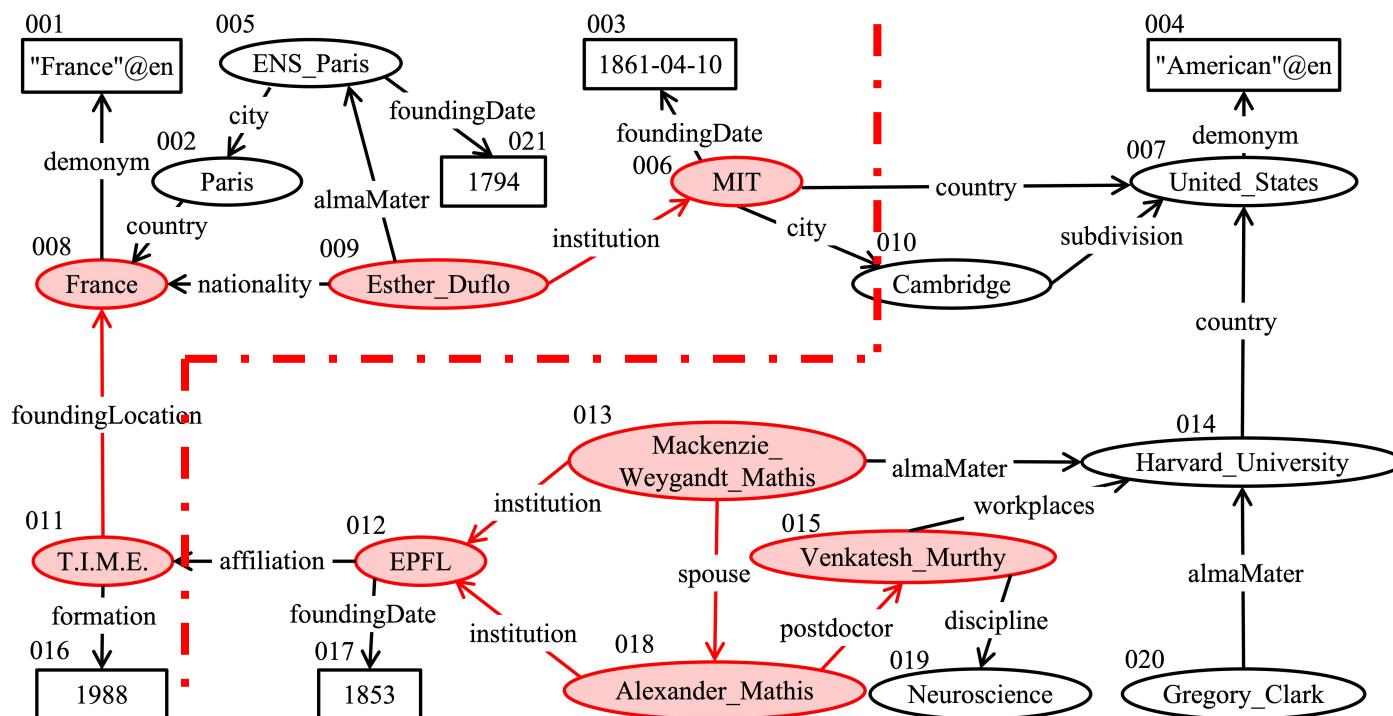
最小模块划分

选择内部模块

粗化

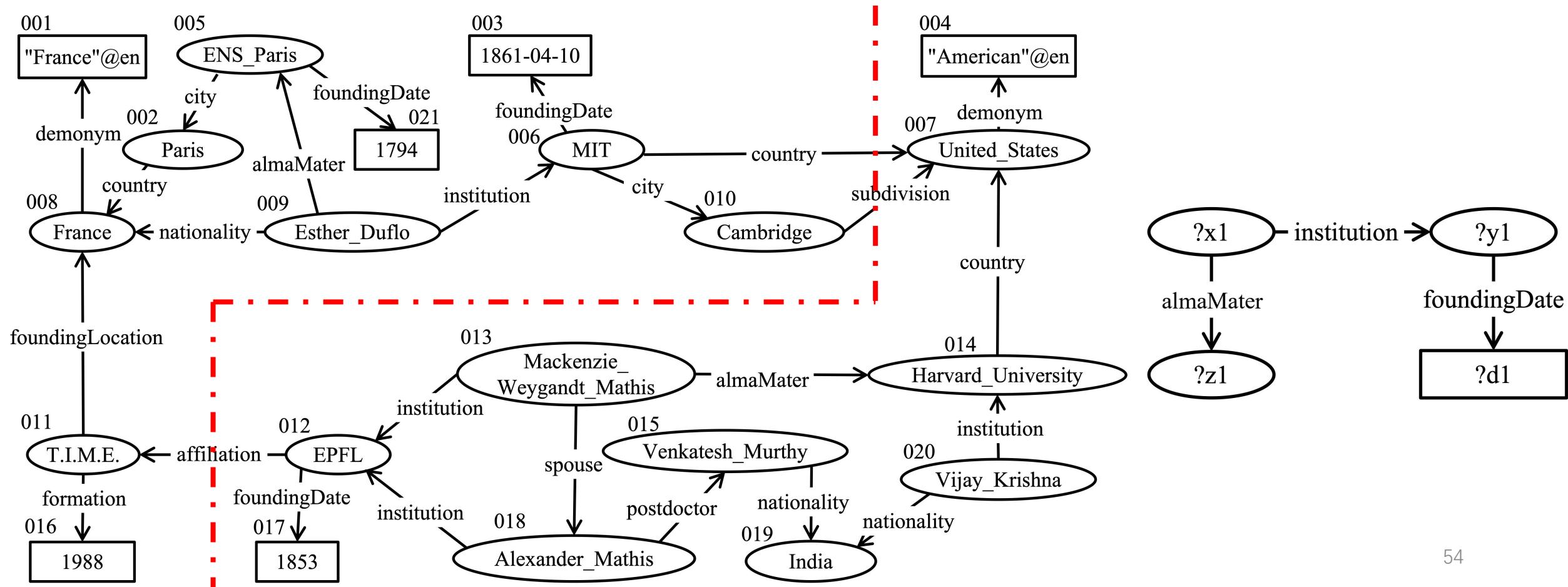
划分粗化图

不粗化

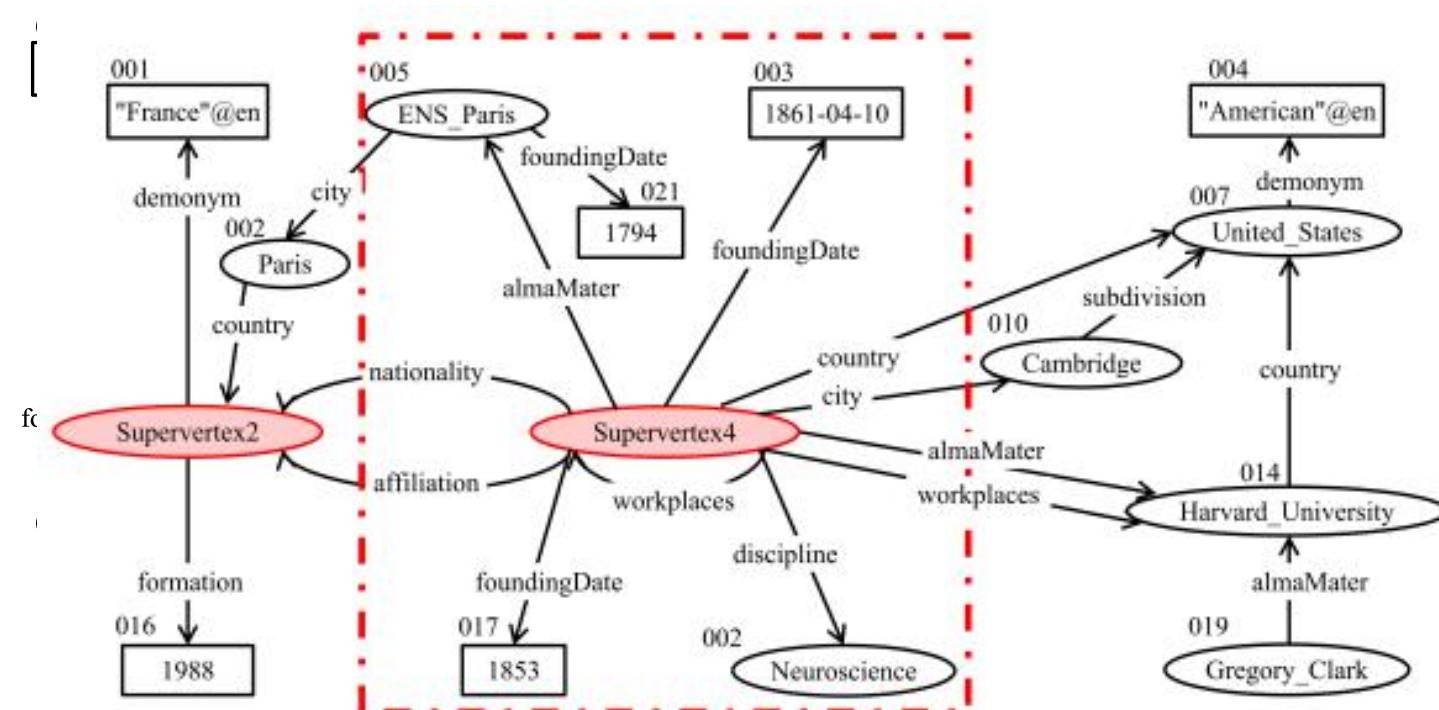


数据本地化

上面描述的基本方法的一个问题是，它没有考虑内部模块在不同分区之间的匹配分布



在粗化过程中，可以将包含相同内部模块匹配的WCCs进一步粗化为更大的超顶点



对高级粗化图的划分进行解粗化，以获得原始图的划分

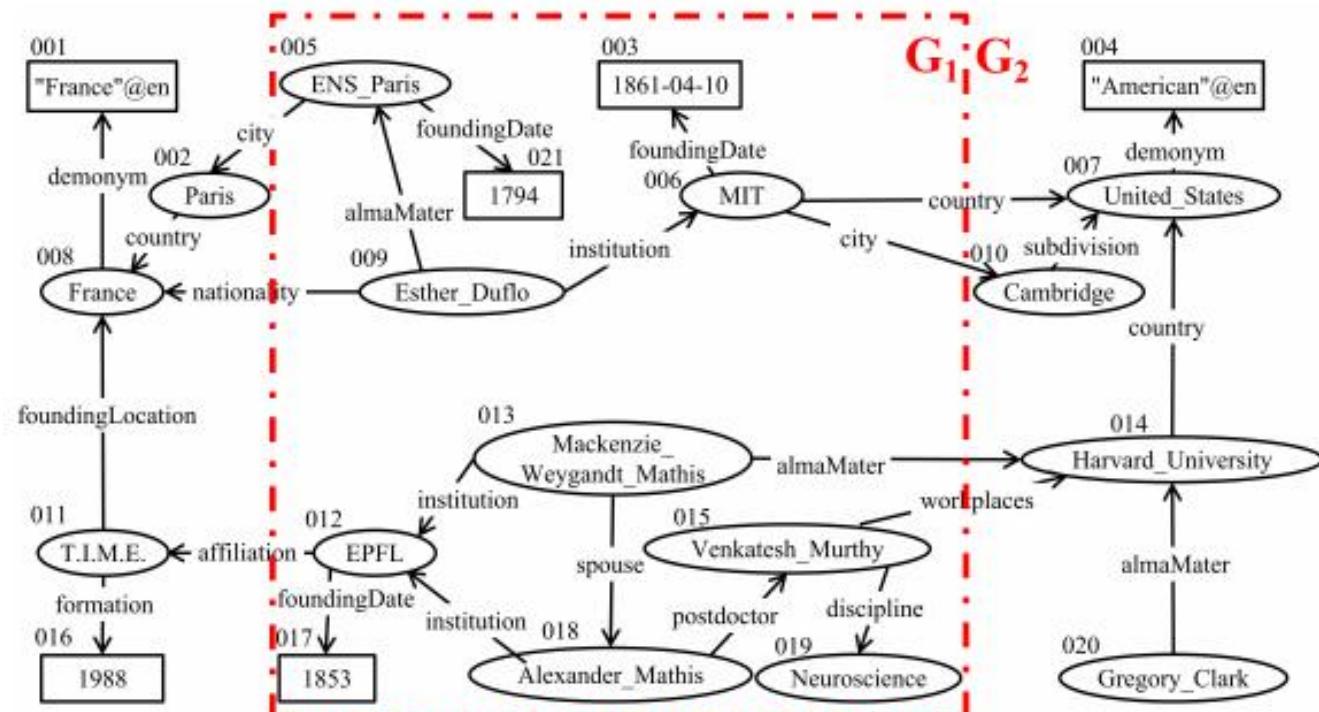


图9 进一步粗化后的分区结果



一个关键问题是哪些超顶点可以一起进一步粗化。直觉上，来自相似的内部模块集合且在满足大小约束的情况下可以一起粗化的超顶点可以被一起粗化

给定 $G[M_{in}]$ 中的两个超顶点 s_1 和 s_2 ，它们在 $G[M_{in}]$ 中的相应 WCC 分别表示为 $wcc(s_1)$ 和 $wcc(s_2)$ ，它们的内部模块集合分别表示为 $IM(s_1)$ 和 $IM(s_2)$

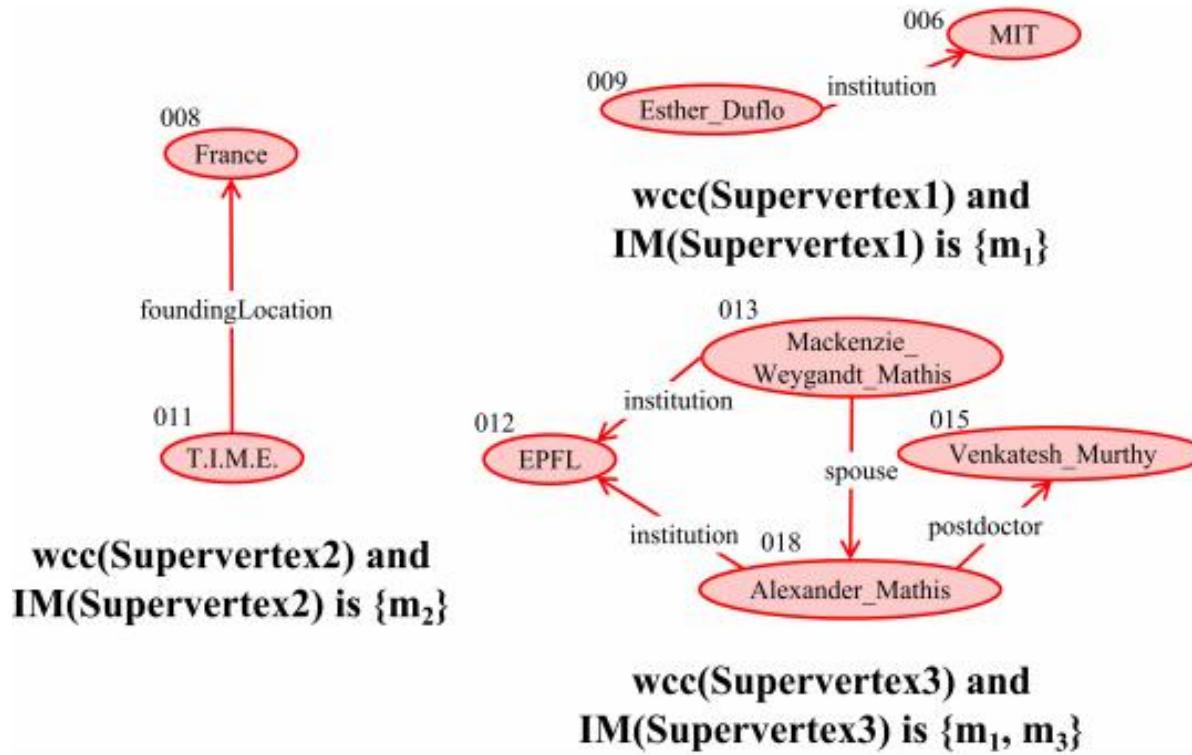
然后，如果它们的内部模块集合 $IM(s_1)$ 和 $IM(s_2)$ 彼此相似，并且结合 $wcc(s_1)$ 和 $wcc(s_2)$ 的大小小于自安置，则两个超顶点 s_1 和 s_2 可以一起粗化。

在这篇论文中，我们使用 $IM(s_1)$ 和 $IM(s_2)$ 之间的Jaccard相似性来衡量 s_1 和 s_2 之间的相似性

将两个超顶点 s_1 和 s_2 一起粗化的收益可以定义如下：

$$Benefit(s1, s2) = \frac{|IM(s1) \cap IM(s2)|}{|IM(s1) \cup IM(s2)|}$$

示例模块诱导子图中的wcc及其对应的内部模块集



基于上述直觉，我们提出了一种优化方法，即使用超顶点对应的内部模块集将其分组成簇

聚类的结果对应一种粗化策略，将同一聚类中的超顶点进一步粗化为一个更大的超顶点

一般情况下，我们迭代选择两个超顶点 s_1, s_2 ，如果它们具有最大的相似度，并且它们对应的WCC大小之和仍然满足大小约束，则可以将它们粗化在一起(即 $|wcc(s_1)| + |wcc(s_2)| \leq (1 + \epsilon) \times |V|/k$)。

为了实现数据本地化，我们需要存储和维护关于哪些分区与哪些模块相关的元数据

我们构建一个哈希表(HT)来实现上述目标。我们还使用DFS编码将内部模块翻译成序列。利用内部模块的DFS代码，我们可以通过散列其规范标签将任何内部模块映射为整数

然后，我们使用HT定位内部模块并检索其相关分区。给定一个内部模块 m ，其相关分区集表示为 $P(m)$ 。

每个内部图案的相关分区， $P(m_1) = \{G_1\}$, $P(m_2) = \{G_2\}$ 和 $P(m_3) = \{G_1\}$ 。

Motifs	Relevant Partitions
 m₁	{G ₁ }
 m₂	{G ₂ }
 m₃	{G ₁ }



MMC分区图上的查询处理首先确定输入查询是否是可独立执行查询(可独立执行查询)

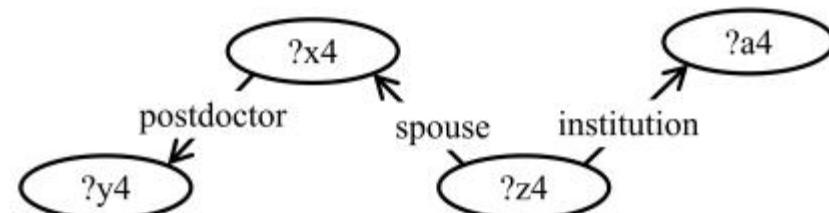
如果是可独立执行查询，我们本地化它的相关分区，将查询发送到这些分区，然后在所有相关分区中合并匹配

如果输入查询不是可独立执行查询，则将其分解为几个独立可执行的子查询，并且每个分解的子查询都在其相关分区上进行本地化。然后，在相关分区上执行分解的子查询，并将它们的匹配项连接在一起以形成最终匹配

MMC中独立可执行的查询

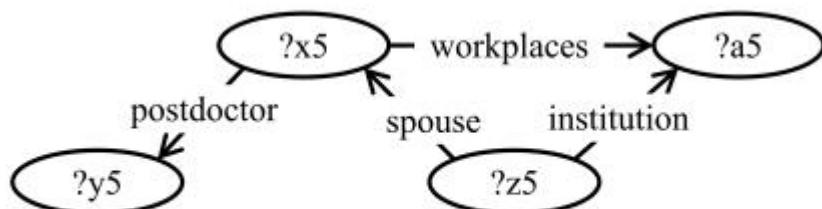
定义6（第一类可独立执行查询）

考虑到RDF图G上的MMC分区，让 M_{in} 代表所有内部模块。给定一个SPARQL查询Q，忽略顶点常量，如果Q的内部模块诱导子图（表示为 $Q[M_{in}]$ ）与Q同构，那么Q就是第一类可独立执行查询。



定义7（第二类可独立执行查询）

如果一个SPARQL查询Q满足以下两个条件，那么它就是第二类可独立执行查询：（1）存在一个子图 Q' （是Q的一部分），它是第一类可独立执行查询；（2）对于Q中所有与 Q' 关联的边e，e的两个端点都位于 $V(Q')$ 中，其中 $E(Q')$ 和 $V(Q')$ 分别表示 Q' 中所有的边和顶点。



定理4

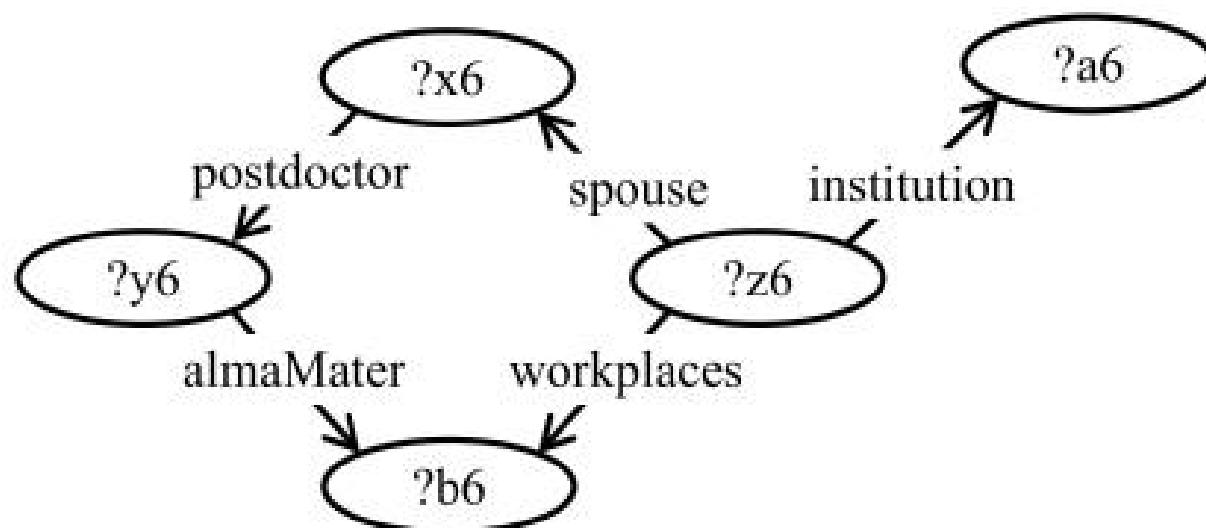
给定一个查询 Q 是上述两类可独立执行查询， Q 的任何匹配都不包含任何跨界边。换句话说， Q 是一个内部可独立执行查询，可以独立执行。

证明：详见论文。

除了上述可独立执行查询之外，还有另一类独立可执行的查询，它们的匹配可能涉及每个分区上跨界边的副本，这些被称为 **Type-III 可独立执行查询**

定义8 (Type-III可独立执行查询)

SPARQL查询Q是Type-III 可独立执行查询当且仅当(1)存在一个(Q的)子图 Q' 是type-I 可独立执行查询;(2) $\forall e \in E(Q) \setminus E(Q')$, e至少有一个端点位于 $V(Q')$; (3) $\exists e^* \in E(Q) \setminus E(Q')$, e^* 的一个端点位于 $V(Q')$, 另一个端点不在 $V(Q')$ 。



定理5 Type-III 可独立执行查询查询Q可以独立执行。

证明： Type-III 可独立执行查询 Q除其模块诱导子图外的多余边集记为 E^{Q_c} ，去掉 E^{Q_c} 中的所有边后记为 Q' 。根据 iii型可独立执行查询的定义， Q' 是 i型或 ii型可独立执行查询，并且 Q' 的任何匹配中的所有顶点必须是同一分区的内部顶点。

由于 E^{Q_c} 中的每条边都与 Q' 中的一个顶点相邻，而 Q' 只与内部顶点匹配，因此由于跨界边复制，内部顶点的所有1跳邻居必须在同一分区中。因此， Q 也可以独立执行。

MMC分区也遵循切边策略，并允许跨界边的副本。因此，在MMC分区中，任何星型查询都可以独立执行

显然，与传统的边切分区(如最小切策略)相比，我们在给定的查询日志中增加了可独立执行查询的数量，因为除了星型查询外，MMC分区中所有类型I、II和III的可独立执行查询都是独立可执行的。

给定查询查询日志W及其相应的MMC分区，如第3节所述，让 W_{in} 表示该查询日志中的所有可独立执行查询， W_{star} 、 W_I 、 W_{II} 和 W_{III} 分别表示W的所有星型查询和类型I、II和III 可独立执行查询

研究几个真实的SPARQL查询日志可以得出两个观察结果

$$\frac{|W_{in}|}{|W|} \approx 99.67\%; \frac{|W_{star}|}{|W|} \approx 46.87\% \quad (1)$$

$$\frac{|W_{star} \cup W_{typeI} \cup W_{typeII} \cup W_{typeIII}|}{|W_{in}|} \approx 99.01\% \quad (2)$$

1. 等式1证实了MMC分区的有效性，这使得更多的SPARQL查询能够独立执行。在传统的边缘切割分区（如最小切割）中，只有星形查询是可独立执行查询。
2. 请注意， $|W_{in}|$ 是在查询处理之前是未知的，因为它依赖于查询结果MS(Q, G)。等式2显示我们的方法（查找类型I、II、III和星形查询）足够接近 W_{in}

$$\frac{|W_{in}|}{|W|} \approx 99.67\%; \frac{|W_{star}|}{|W|} \approx 46.87\% \quad (1)$$

$$\frac{|W_{star} \cup W_{typeI} \cup W_{typeII} \cup W_{typeIII}|}{|W_{in}|} \approx 99.01\% \quad (2)$$

面向可独立执行查询的分布式查询处理

由于我们可以保证可独立执行查询 Q 不涉及任何跨界匹配，因此可以直接将其发送到相应的分区并在那里执行

我们将 Q 的相关分区集表示为 $P(Q)$ ，相关分区处的查询表示为 $Q @ P(Q)$ 。于是，整个 RDF 图 G 上 Q 的匹配集 $MS(Q, G)$ 可以按如下方式获得

$$MS(Q, G) = \bigcup_{G_i \in P(Q)} MS(Q, G_i)$$

评估可独立执行查询 Q 的关键问题是如何将 Q 定位到其相关分区集

我们首先使用 Q 的 DFS 代码检索前文提出的哈希表，并确定它包含的内部模块集 $IM(Q)$ 。然后， Q 的相关分区可以确定为它对应的内部模块的相应分区集的交集

$$P(Q) = \bigcap_{m \in IM(Q)} P(m)$$

定理6

给定一个可独立执行查询 Q 及其内部模块集 $IM(q)$ ，以及 $m_1, m_2 \in IM(q)$ ，如果一个分区 G_i 与 m_1 相关但与 m_2 无关，那么 G_i 不可能有 Q 的匹配。

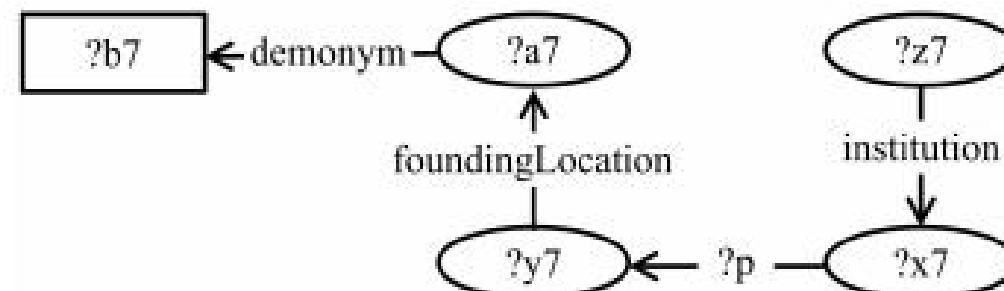
证明：因为可独立执行查询 Q 是可独立执行的，所以 Q 的所有匹配都应该是内部匹配。另一方面， Q 是 m_1 和 m_2 的组合，所以 Q 的每个匹配也是 m_1 和 m_2 的匹配的组合。因此， m_1 和 m_2 的两个匹配应该在同一个分区中。由于分区 G_i 只与 m_1 相关而与 m_2 无关， G_i 不能是 m_1 和 m_2 的两个匹配的分区。因此， G_i 不可能有 Q 的匹配。

非可独立执行查询的分布式查询处理

我们首先将其分解为一组可独立执行查询子查询

然后，我们对每个可独立执行查询子查询在其相关分区进行定位和评估

最后，通过将非可独立执行查询的所有子查询的结果连接起来，形成最终匹配



查询分解

为了分解查询 Q ，我们对它的DFS代码检索哈希表，确定 Q 包含的内部模块集

基于内部模块集，我们首先删除不在 M_{in}^Q 的任何内部模块中的边集 $E_{crossing}^Q$ ，得到

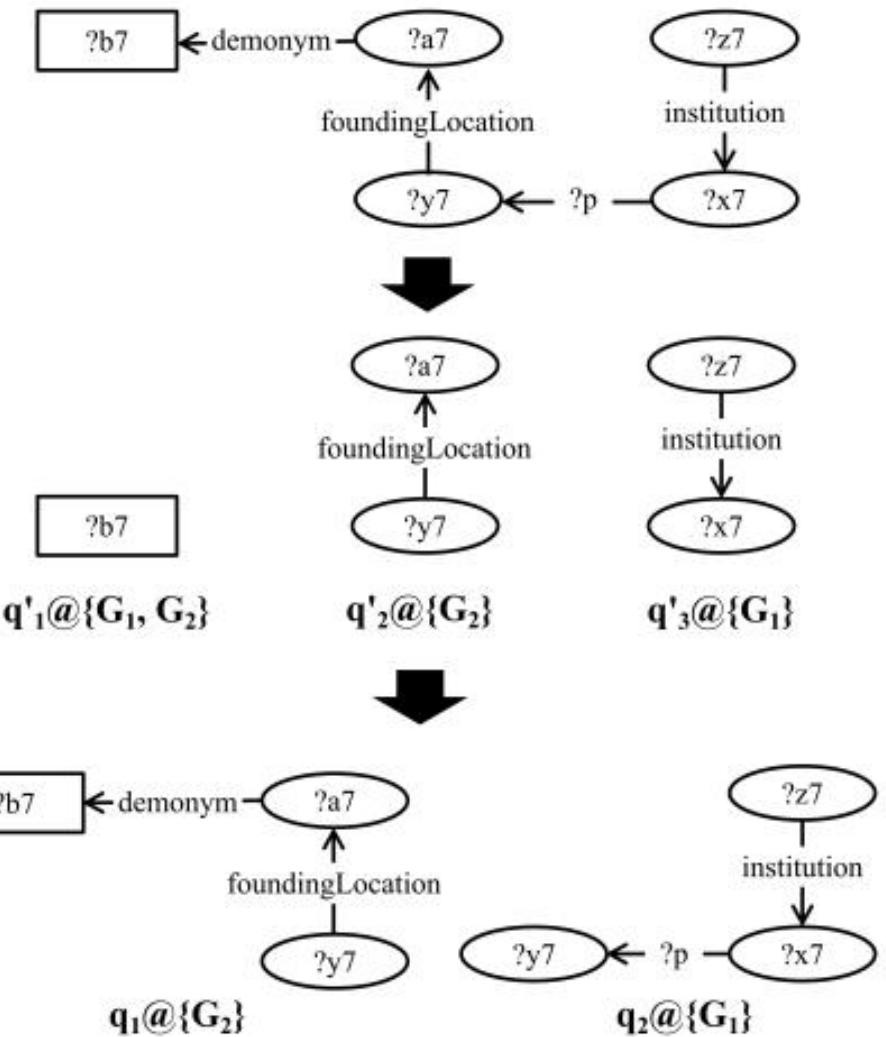
一组WCCs， $\{q'_1, q'_2, \dots, q'_x\}$

删除 $E_{crossing}^Q$ 中的所有边后，我们可以得到只包含内部模块中的边的WCCs，它们中的每一个都是内部可独立执行查询。然后，对于每个WCC q'_i ，我们根据其对应的内部模块集 $IM(q'_i)$ 来确定其相关分区

最后，对于 $E_{crossing}^Q$ 中的边，我们考虑将它们添加到WCCs中

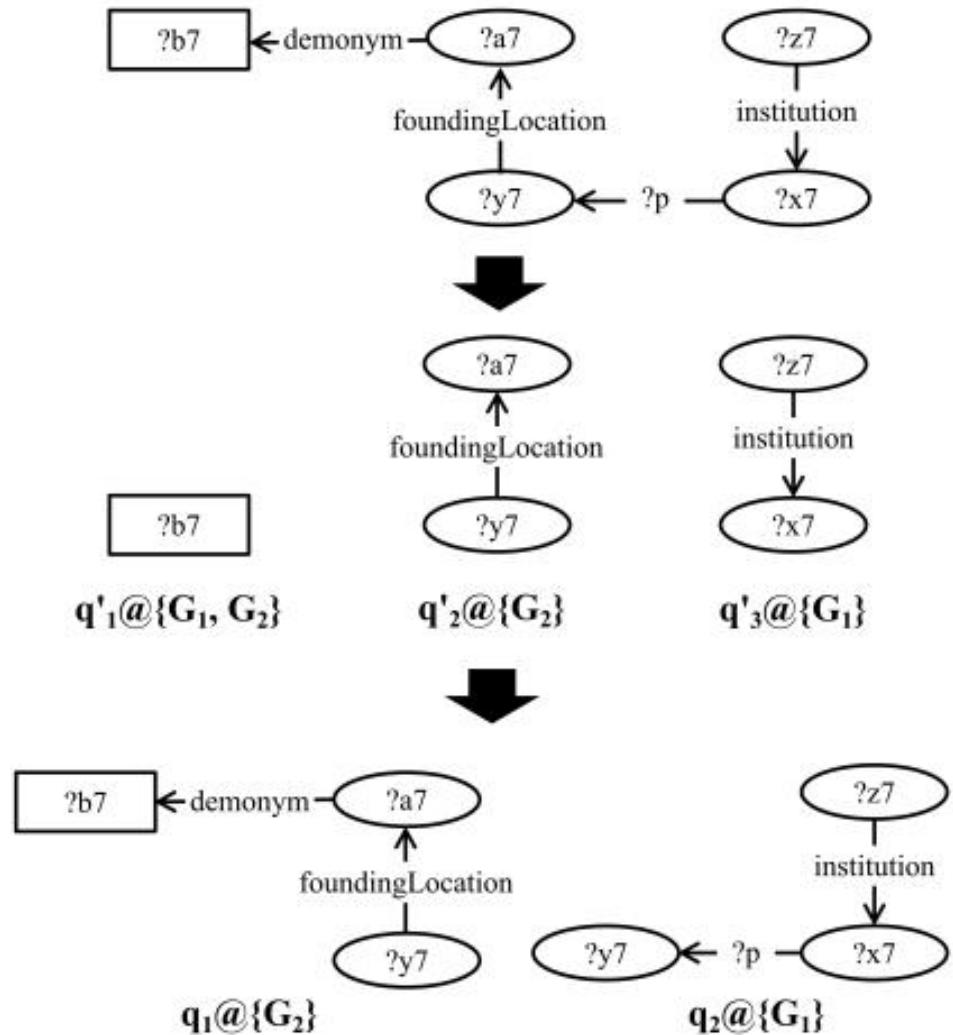
我们可以首先检测到 m_1 和 m_2 是 Q_7 的子图，
并获取不在任何内部模块中的边的集合

$$E_{crossing}^Q, \text{ 即 } \{\overrightarrow{?a7?b7}, \overrightarrow{?x7?y7}\}$$



然后，查询被分解为三个子查询：
 q'_1 、 q'_2 和 q'_3 ，其中 q'_1 是一个没有任何边的平凡子查询

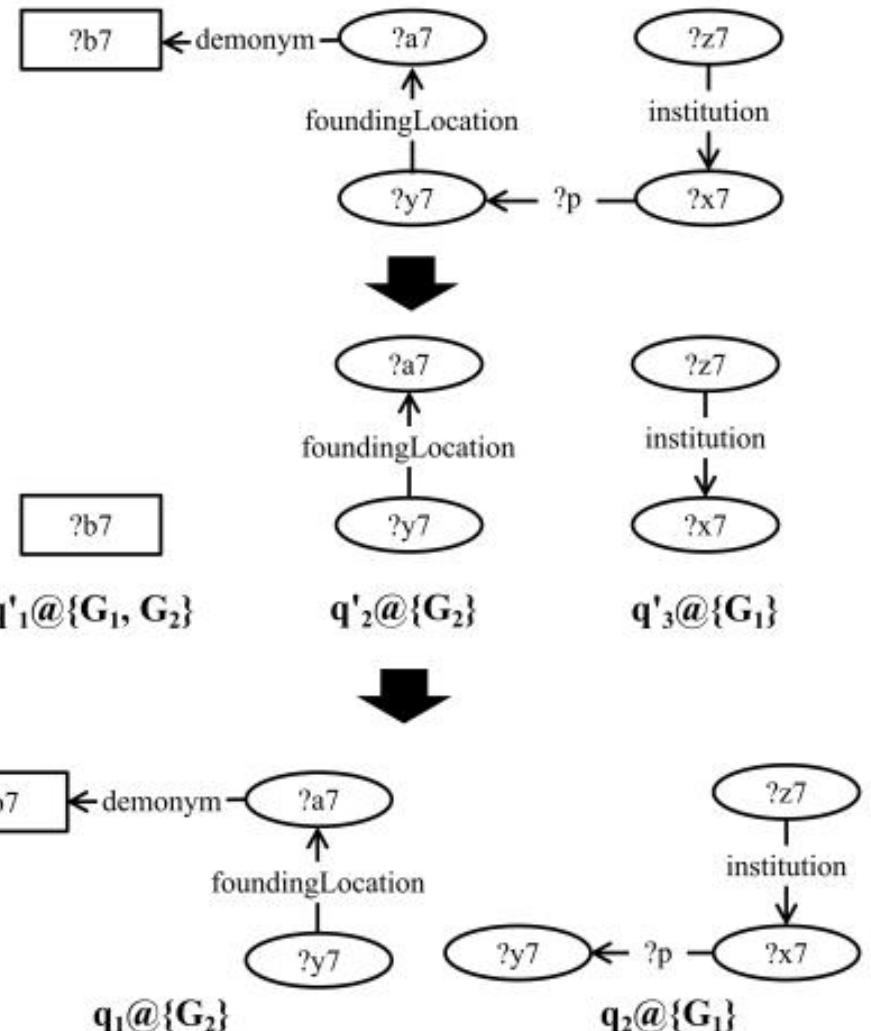
对于数据本地化， q'_1 只有一个顶点而没有边，所以它可以与两个分区 G_1 和 G_2 相关； q'_2 对应于与分区 G_2 相关的内部模块 m_2 ； q'_3 对应于与分区 G_1 相关的 m_1



因此，数据本地化结果是 $q'_1 @ \{G_1, G_2\}$ 、
 $q'_2 @ \{G_2\}$ 和 $q'_3 @ \{G_1\}$

然后，边 $\overrightarrow{x7?y7}$ 可以添加到 $q'_2 @ \{G_2\}$ 或
 $q'_3 @ \{G_1\}$ ，这里，我们假设 $\overrightarrow{x7?y7}$ 被添加到
 $q'_3 @ \{G_1\}$ 以形成 $q'_2 @ \{G_1\}$

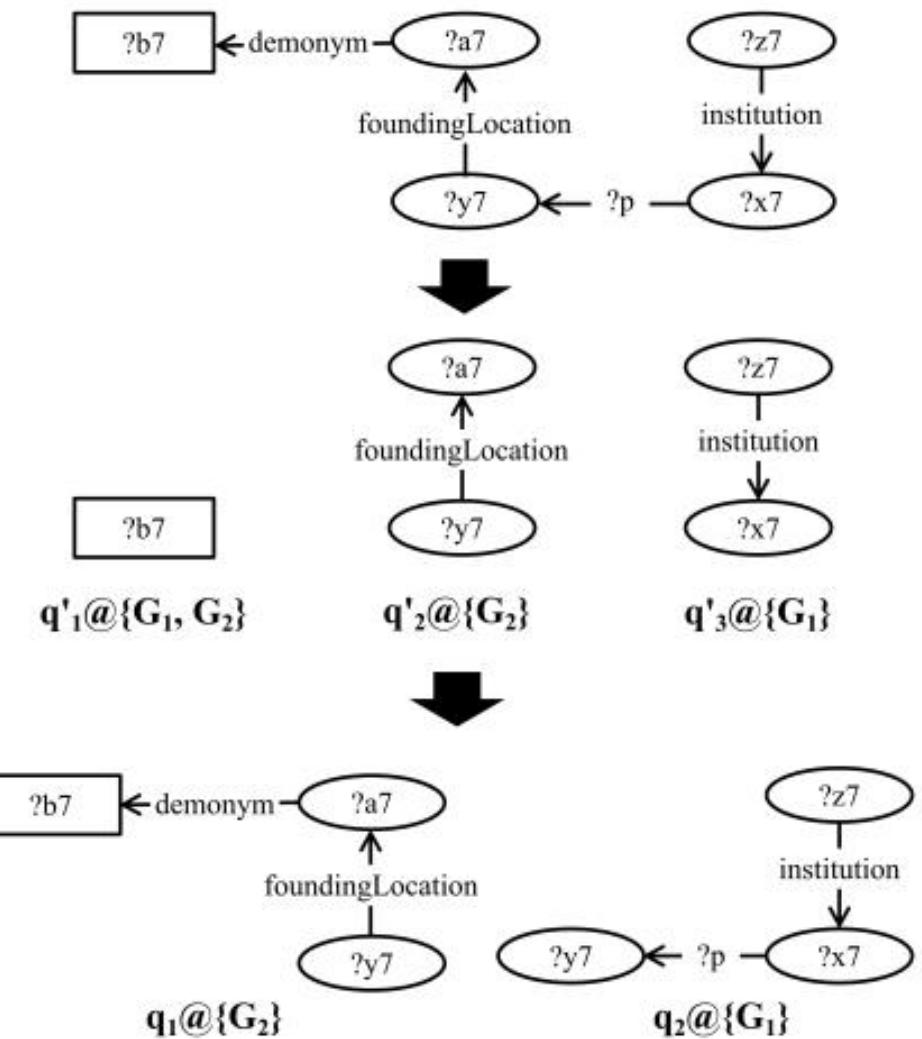
同时，边 $\overrightarrow{a7?b7}$ 与 $q'_1 @ \{G_1, G_2\}$ 和 $q'_2 @ \{G_2\}$
 相邻，应该被添加到 $q'_2 @ \{G_2\}$ 以形成 $q'_1 @ \{G_2\}$ 。



然后，我们可以发现在 $q'_1 @ \{G_1, G_2\}$ 中有一个顶点，所以 $q'_1 @ \{G_1, G_2\}$ 可以被移除。

因此，查询分解的结果是两个子查询：

$q'_1 @ \{G_2\}$ 和 $q'_2 @ \{G_1\}$ 。



查询执行

当接收到非可独立执行查询 Q 时，将其分解为一组子查询 $\{q_1, q_2, \dots, q_y\}$ 它们被发送到相关分区执行

每个 q_i 都可以在其相关分区上独立执行，从而允许在每个分区站点上并发执行，也允许在工作机器上并发执行。这些执行的结果是每个子查询的一组匹配，即 $MS(q_i)$

子查询的匹配需要连接起来得到最终的结果，即 $MS(Q) = \bowtie_{i=1}^y MS(q_i)$ 。

□ 数据集：

Dataset	#Entities	#Triples	#Properties
LUBM 100M	17,473,142	106,909,064	18
LUBM 1B	173,891,493	1,069,331,221	18
LUBM 10B	1,737,718,408	10,682,013,023	18
WatDiv 100M	5,212,745	108,997,714	86
WatDiv 1B	52,120,745	1,099,208,068	86
WatDiv 10B	521,200,745	10,987,996,562	86
YAGO2	21,073,153	284,417,966	98
Bio2RDF	804,671,979	4,426,591,829	1,581
DBpedia	139,493,254	1,111,481,066	124,034
LGD	311,153,753	1,292,933,812	33,348

- 竞争对手：Subject_Hash、METIS、VP、MF、WARP
- 环境：在阿里云上运行Linux的8台机器

□ 可独立查询比例与复制比：

Datasets	MMC+DL		MMC		WARP		MF	
	α^1	γ^2	α	γ	α	γ	α	γ
LUBM	100%	1.23	100%	1.23	100%	1.27	100%	1.33
WatDiv	80%	1.72	80%	1.74	70%	1.62	40%	1.24
YAGO2	100%	1.38	100%	1.38	100%	1.46	100%	1.44
Bio2RDF	100%	1.43	100%	1.43	-	-	100%	1.27
DBpedia	99.74%	1.57	99.74%	1.56	83.11%	1.74	50.38%	1.95
LGD	99.98%	1.44	99.98%	1.46	98.40%	1.57	91.11%	1.80

¹ α means the percentage of IEQs.

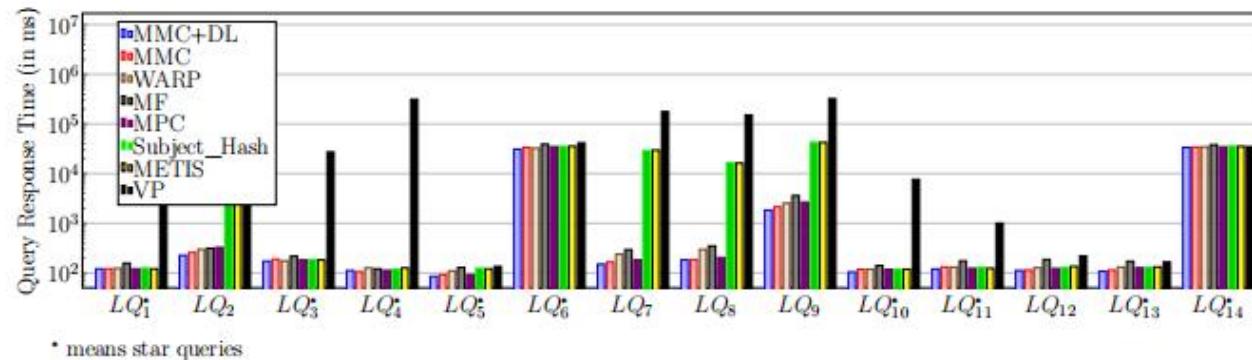
² γ means the redundancy rate.

³ - means that data size is beyond the capacity.

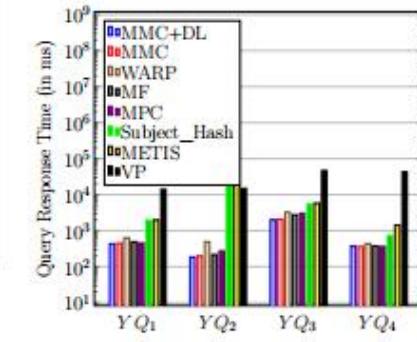
Datasets	MPC		Subject_Hash		METIS		VP	
	α	γ	α	γ	α	γ	α	γ
LUBM	100%	1.28	71.43%	1.58	71.43%	1.20	28.57%	1.00
WatDiv	60%	1.88	50%	1.89	50%	1.53	0%	1.00
YAGO2	100%	1.45	0%	1.50	0%	1.21	0%	1.00
Bio2RDF	100%	1.46	80%	1.49	-	-	40%	1.00
DBpedia	75.19%	1.45	46.87%	1.61	46.87%	1.15	24.25%	1.00
LGD	99.95%	1.40	96.95%	1.52	96.95%	1.23	83.51%	1.00

在线性能比较

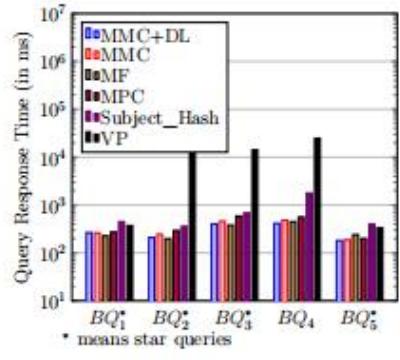
□ 在基准查询上：



(a) LUBM

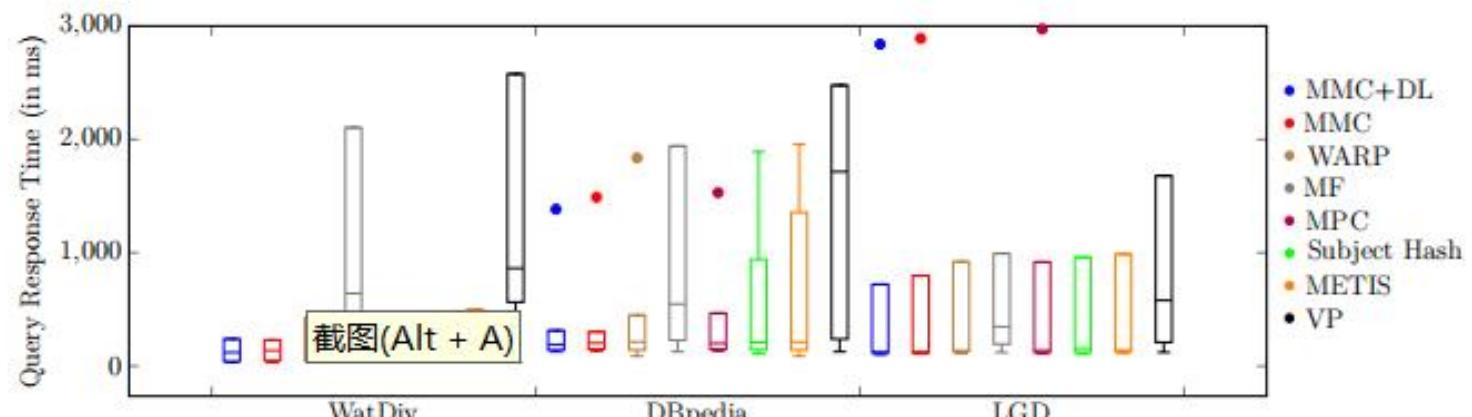


(b) YAGO2



(c) Bio2RDF

□ 在真实查询日志上：





湖南大学
HUNAN UNIVERSITY

第四部分

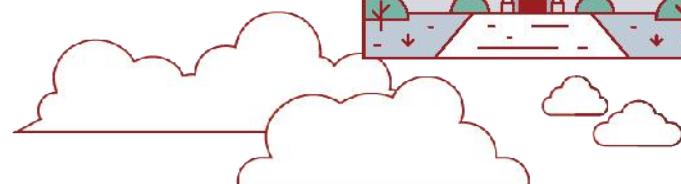
结论



- 我们介绍了两种基于顶点不相交分区的分布式RDF系统：**MPC**和**MMC**。
- 我们的实验表明，MPC和MMC在合成和实际的RDF图上都表现出高效的性能。

谢谢观看

thank you for watching



1. Jiewen Huang, Daniel J. Abadi, Kun Ren. Scalable SPARQL Querying of Large RDF Graphs. Proc. VLDB Endow. 4(11): 1123-1134 (2011)
2. Kisung Lee, Ling Liu. Scaling Queries over Big RDF Graphs with Semantic Hash Partitioning. Proc. VLDB Endow. 6(14): 1894-1905 (2013)
3. Yuanbo Guo, Zhengxiang Pan, Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. J. Web Semant. 3(2-3): 158-182 (2005)
4. Peng Peng, M. Tamer Özsü, Lei Zou, Cen Yan, Chengjun Liu. Accelerating Partial Evaluation in Distributed SPARQL Query Evaluation. Accepted in ICDE 2022
5. Katja Hose, Ralf Schenkel. WARP: Workload-aware replication and partitioning for RDF. ICDE Workshops 2013: 1-6