# BENU: Distributed Subgraph Enumeration with Backtracking-based Framework

|  |
| --- |
|  |
| Wang Z, Gu R, Hu W, et al. BENU: Distributed Subgraph Enumeration with Backtracking-Based Framework[C]//2019 IEEE 35th International Conference on Data Engineering (ICDE). IEEE, 2019: 136-147. |
| 提纲：  1.背景动机（子图匹配问题是np难的，匹配结果比最终结果大的多，BFS和DFS的困难，如何避免shuffle中间结果？减少甚至避免使用索引？  ,DFS框架） 1-2分钟 |  |

2.模型框架：on-demand shuffle，模型框架。（1分钟）

3.优化技术1，2，3（3分钟）

4.执行计划生成技术（1分钟）

5.实验 （2分钟）

6.总结和个人思考

回溯法下的分布式子图：

动机：

困难1：是子图枚举的核心操作即子图同构是np-hard的，这给问题带来了较高的计算复杂度。

困难2：部分匹配结果和最终匹配结果的大小可以比数据图本身大得多。

相关工作：

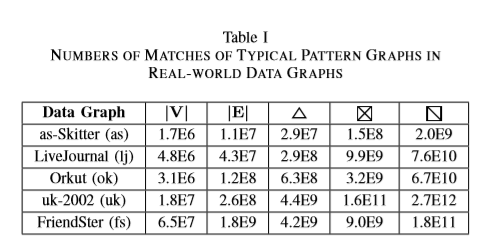
DFS-style和BFS-style

DFS:不需要shuffle中间结果。并行的广播数据图。QFrag[10]并行地广播数据图并枚举内存中的子图。机器的内存空间限制了它的可伸缩性。Afrati等人[11]采用MapReduce上的单轮多路连接进行枚举，但由于边缘复制量大，无法扩展到复杂的模式图，

BFS:BFS风格的算法递归地将模式图分解为一系列连接单元。它们首先枚举连接单元的匹配结果，然后通过一个或多个回合的连接将它们组合起来，以获得整个模式图的匹配结果。部分匹配结果(即中间结果)在连接过程中经过了shuffle。提出了多种连接单元(Edge [13]， Star [5] [14]， TwinTwig [12]， Clique [5]， Crystal[6])和连接框架(Left-deep join [12]， Bushy join [5]， Hash-assembly [6]， Generic join[13])来减小中间结果的大小。

主要缺点：1.中间结果很多（下图中结构是其它结构的基础组成部分），对中间结果的shuffle代价昂贵。2.索引需要额外空间。

动机：如何避免shuffle中间结果？减少甚至避免使用索引？

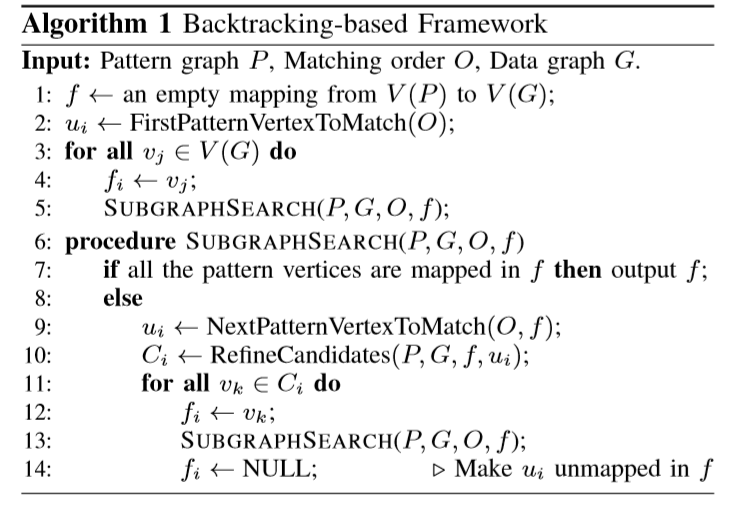


主要贡献：

1. 基于on-demand “shuffle”提出了子图迭代算法框架-BENU。Unlike the existing DFS-style algorithms that shufﬂe the data graph before enumeration in a one-round manner,当需要的时候才去访问。以key-value的形式存储在分布式数据库中进行，只需要去query需要的数据。
2. a search-based 方法去找在BENU下最小代价的最佳执行计划。
3. 提出了2个高效BENU实现方法。Cache缓存从分布式数据库获得的数据，充分利用了inter-task和intra-task的局部性从而减少交流代价。Task splitting 解决幂次分布。
4. 实验

问题定义

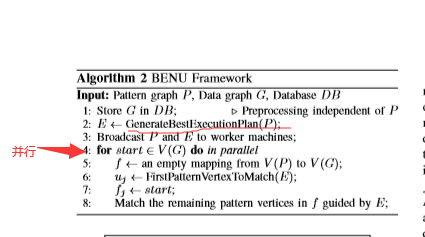
回溯法的框架：



这种dfs很有为题。考虑：查询图中包含一个三角形，但是数据图中没有这个三角形结构，那么一种更有效的方式是去先看有没有三角形再去做子图匹配。

on-demand shufﬂe: 主要思想是将数据图存储在分布式数据库中，在RefineCandidates需要这条边的时候，去请求。

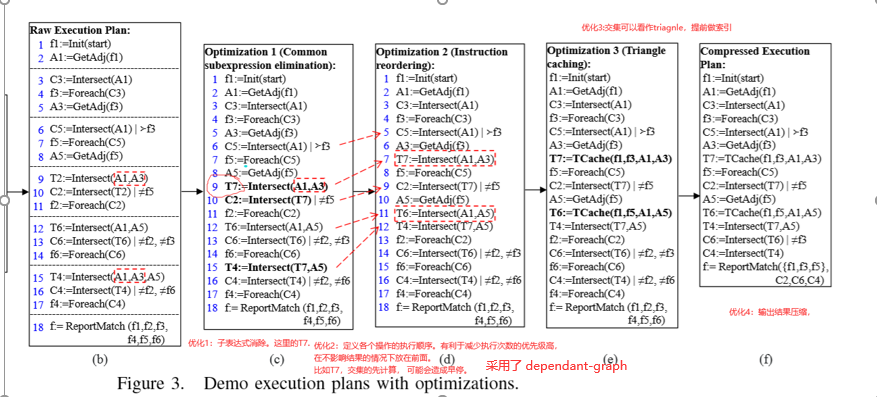
本文框架



贡献点1：

在匹配顺序给定的情况下，先定义了

1. Raw Execution Plan Generation(就是在匹配顺序给定的情况下，在这个框架下一个个匹配),做优化



优化1：

优化2：

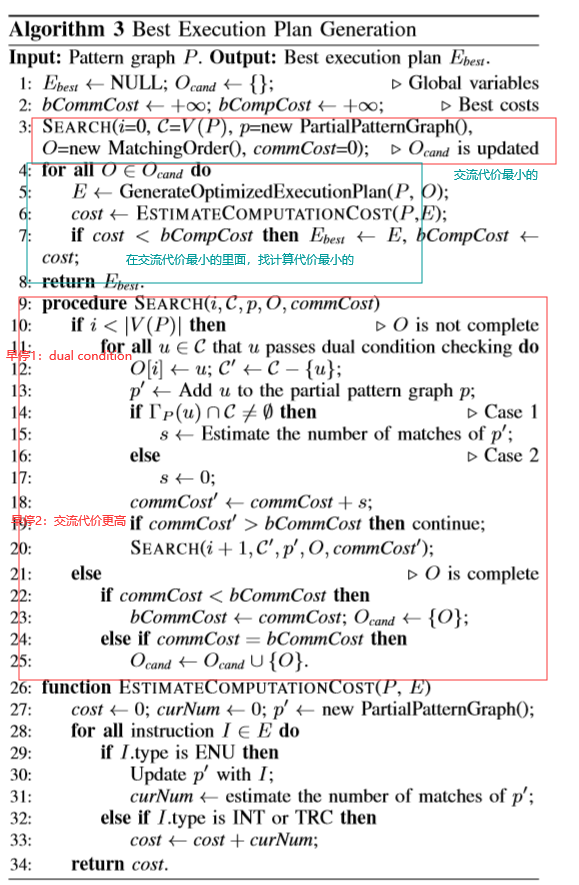
优化3：

1. Execution Plan Cost Estimation

计算的代价看作 INT/TRC的次数。交流的代价看作DBQ的次数。而这些每一个次数，都与前面的foreach有关(Loop层数。)

Estimate Computation Cost:

Best Execution Plan Generation:





用了一个算法，去估计匹配p’的次数。并且认为P’的匹配次数=交流的次数。

1. EFFICIENT IMPLEMENTATION

利用task内部的局部性和task之间的局部性，采用cache。

Task splitting:

实验