【SIGMOD/VLDB/ICDE20xx勉強会】

Session 18: Query Processing and Optimization II

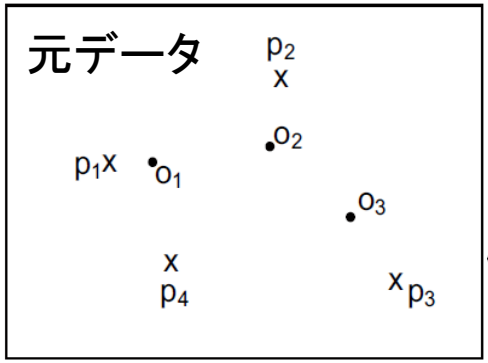
担当：渡辺陽介（東京工業大学）

[1本目]
MaxFirst for MaxBRkNN

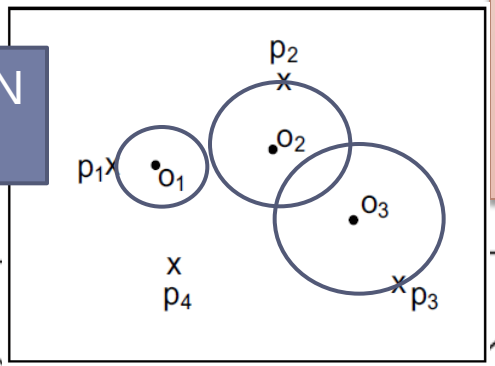
- 著者
  - Zenan Zhou (National University of Singapore)ら
- Contributions
  - Max Bichromatic Reverse k-Nearest Neighbor 問題 (略してMaxBRkNN) を効率よく解くためのMaxFirstアルゴリズム
    - MaxBRkNNについては次ページ参照
    - 探索空間を減らすために，部分空間を4分割していく方法を採用
  - MaxBRkNN問題に，ユーザごとの重みと，利用頻度の確率モデルを考慮できるように一般化
  - PVLDB2009で提案されたMaxOverlapとの比較実験

Max Bichromatic Reverse k-Nearest Neighbor (MaxBRkNN) 問題とは？

- 2次元平面にある2種類のオブジェクトの集合（oの集合とpの集合）
  - 各 o から k-最近傍にある p がすでに定まっている
- このとき，新たな p’ を配置して，p’ がoのk-最近傍に選ばれる回数が最も多くなるような領域を求める

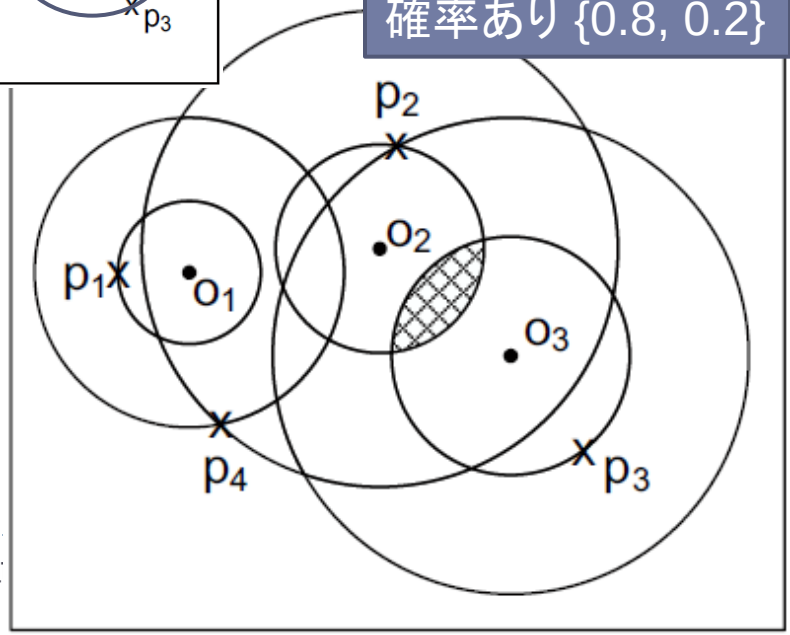


MaxBRNN (k=1)



k-最近傍といっても，上位と下位では使用頻度に差があるはずだ
→確率（重み？)の導入

MaxBRkNN (k=2) 確率あり {0.8, 0.2}



MaxBRkNN (k=2)

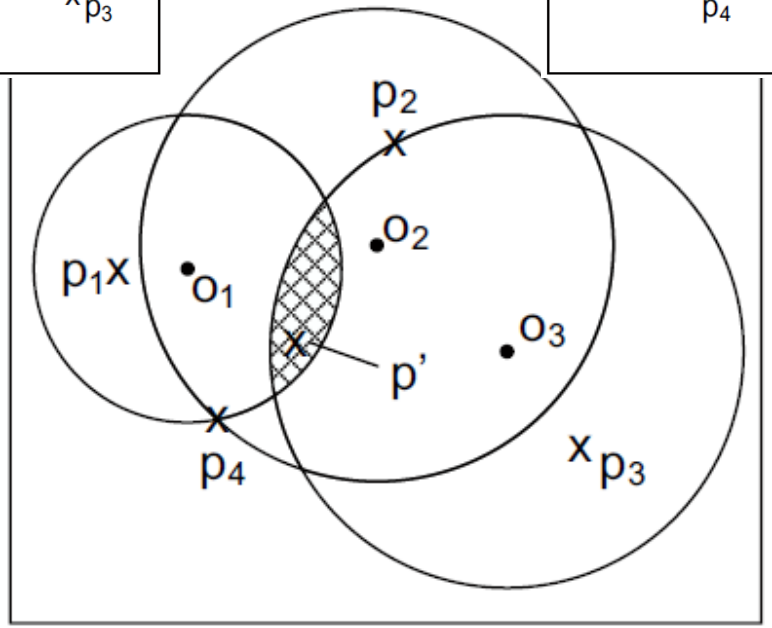
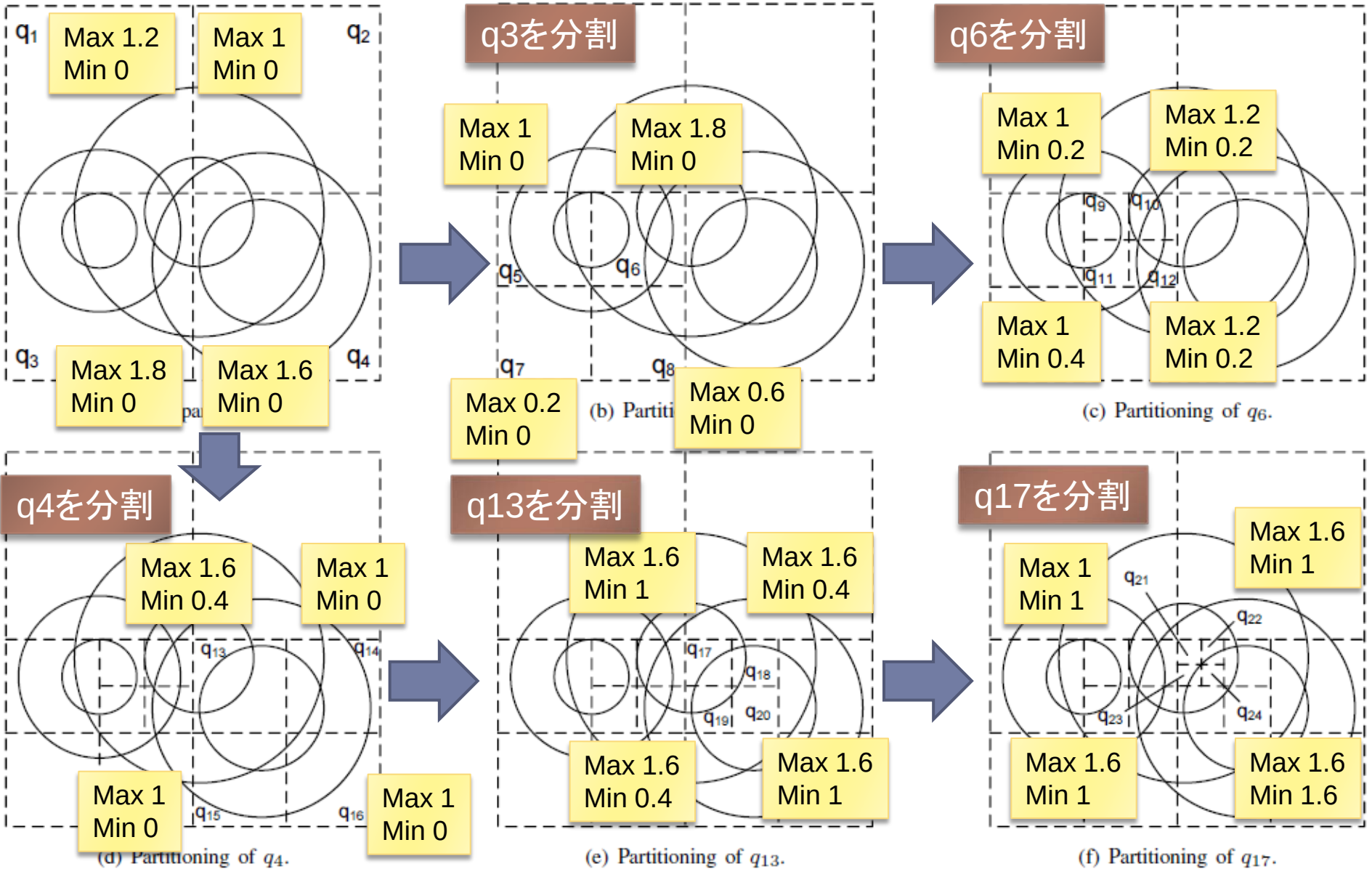


Fig1,2,3より引用

MaxFirstアルゴリズムによる空間分割

Fig6より引用



(b) Partiti

(c) Partitioning of q_6.

(d) Partitioning of q_4.

(e) Partitioning of q_{13}.

(f) Partitioning of q_{17}.

[2本目]
SQPR: Stream Query Planning with Reuse

- 著者
  - Evangelia Kalyvianaki (Imperial College)ら
- Contributions
  - 分散ストリーム処理システムにおける複数問合せ最適化問題
    - 問合せ間で共通な演算結果の共有を行う
    - 計算リソースの制約，ネットワーク帯域の制約，ループ禁止の制約
    - 問合せの追加に対するインクリメンタルな最適化
  - 最適化問題の定式化
    - 目的関数，制約を数式化し，整数計画問題として解けるようにした
  - 個人的には2005年ごろからあった分散ストリームの問合せ最適化問題を，きれいにまとめ直したという印象

分散ストリーム処理システムにおける 複数問合せ最適化問題

- 例: (図2) Query1と2で共通な演算o3の結果を共有する
  - ホストh1は別の問合せの演算を2つ処理している
  - ホストh2のCPUはidleだが，ネットワーク帯域が狭い
- 何をもって最適とするか
  - 処理できる問合せ数，全体のリソース使用量，負荷の均衡

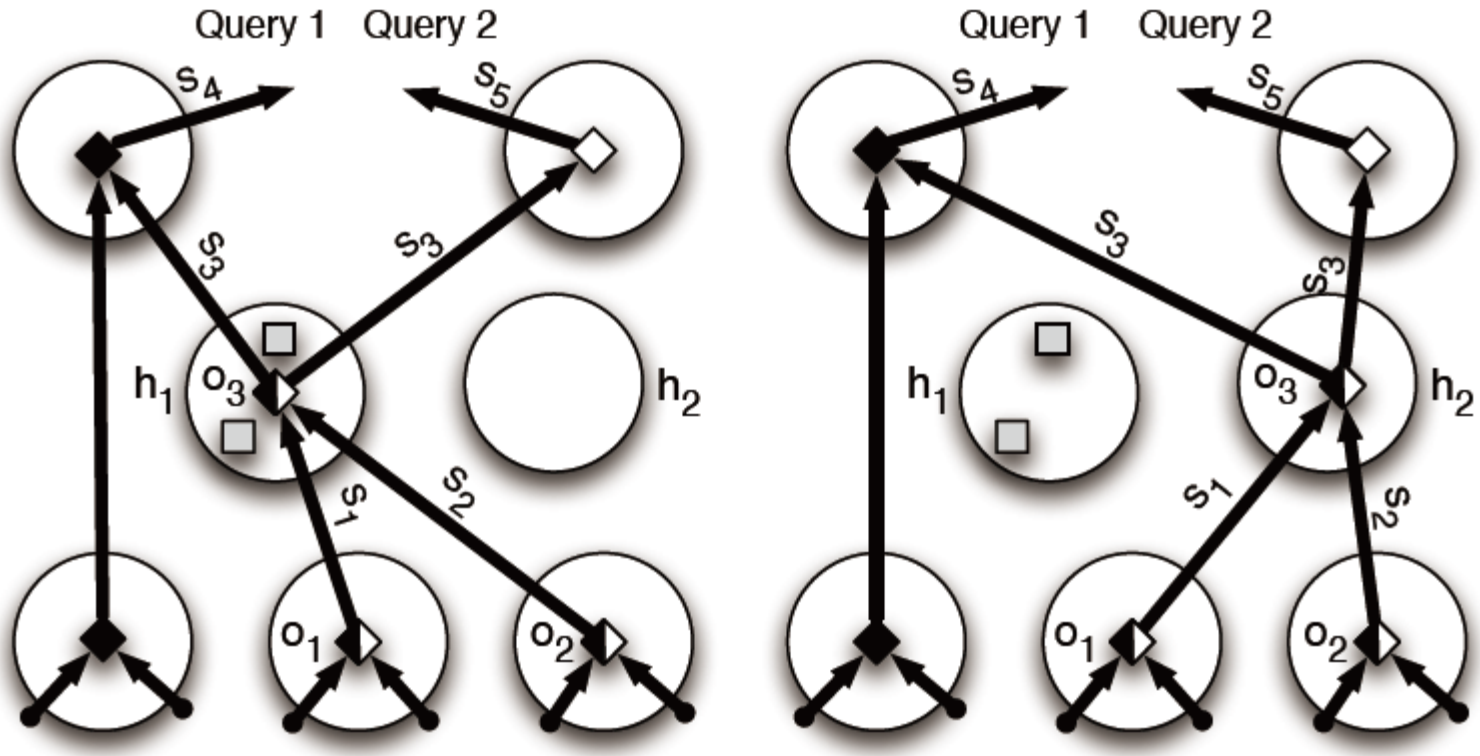


Fig2より引用

最適化問題の定式化

- O1: 要求を満たせる問合せ数（多いほど良い）
- O2: ネットワーク全体の利用量（少ないほど良い）
- O3: 計算資源の使用量(CPUコア数)（少ないほど良い）
- O4: ホスト間の負荷のばらつき （少ないほど良い）

$$O_1 := \sum_{\substack{s\in\mathcal{S},\\ h\in\mathcal{H}}} d_{hs}\,, \qquad O_2 := \sum_{\substack{s\in\mathcal{S},\\ h,m\in\mathcal{H}}} \varrho_s x_{hms}\,,$$

$$O_3 := \sum_{\substack{h\in\mathcal{H},\\ o\in\mathcal{O}}} \gamma_o z_{ho}\,, \qquad O_4 := \max_{h\in\mathcal{H}} \sum_{o\in\mathcal{O}} \gamma_o z_{ho}\,. \tag{III.2}$$

変数d_{hs}: ホストhでストリームsが使えるか
変数x_{hms}: ホストhからmへストリームsを転送
変数z_{ho}: ホストhに演算oを置くか

最適化問題

目的関数:

$$\begin{aligned} &\underset{d,x,y,z,p}{\text{maximise}} && \lambda_1 O_1(d) - \lambda_2 O_2(x) - \lambda_3 O_3(z) - \lambda_4 O_4(z) \\ &\text{subject to} && \text{(III.4)–(III.7).} \end{aligned} \tag{III.8}$$

制約:

- Demand constraint(式III.4): ストリームのデータが目的のホストに届いている
- Availability constraint(式III.5): ホスト間通信が各ホストの帯域上限を超えない
- Resource constraint(式III.6): 各ホストの計算資源を上回る演算が配置されない
- Acyclicity constraint(式III.7): ストリームの転送経路が循環してはならない

[3本目] Memory-Constrained Aggregate Computation over Data Streams

- 著者
  - K.V.M. Naidu (Bell Labs India Research)ら
- Contributions
  - 同じデータストリームに対する複数の問合せの集約演算をまとめて計算するintermediate aggregates処理手法
    - 元は参考文献 15 (SIGMOD2005)で提案されたアイデア
  - Hash-tableモデルの提案
    - コストモデル(ハッシュ操作の回数を見積もる)
    - コストの低い実行プランを構築するためのgreedyアルゴリズム
    - 実行プランにメモリ割り当てを行うアルゴリズム FPTAS
      - Fully polynomial-time approximation scheme

Intermediate aggregates

- ストリームに対する集約演算
  - 例：ネットワークのパケットの統計
  - データ量が多く，メモリ上で高速に処理する必要がある
  - 無限にデータが到着する
    - メモリ量は有限なため，すべてのデータを記憶することができない
- 複数の集約演算の集計値を個別に保持するのは冗長

```
SELECT  srcIP, dstIP, SUM(byte)
FROM  stream
GROUP By  srcIP, dstIP
EVERY 15 minutes
```

```
SELECT  dstIP, dstPort, SUM(byte)
FROM  stream
GROUP By  dstIP, dstPort
EVERY 15 minutes
```

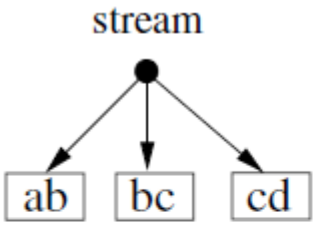
dstIPでグルーピングする部分は共通
{ srcIP, dstIP, dstPort } の中間的なグループ表を作るのはどうか？

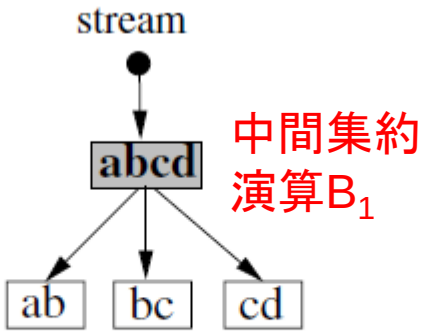
Intermediate aggregate
コスト見積り例

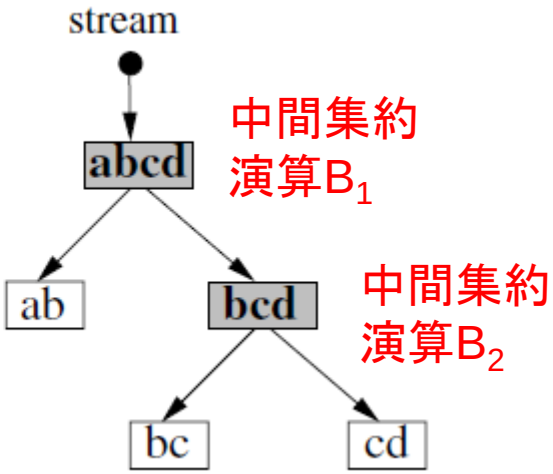
- Intermediate aggregateによる集約の共通化
  - ただし，各演算にはハッシュ表に使用可能なメモリ量の上限がある
  - それを超えた場合は，収まらない分の集約値を下位の演算に出力する

問合せ集合　Q1:属性a,bで集約，Q2:属性b,cで集約，Q3:属性c,dで集約

実行プラン例







コスト式
(ハッシュ表の操作回数)

$$3 \cdot n_R$$

$$n_R + 3 \cdot (b_1 + f_1 \cdot n_R)$$

$$n_R + 2 \cdot (b'_1 + f'_1 \cdot n_R) + 2 \cdot (b'_2 + f'_2 \cdot b'_1 + f'_2 \cdot f'_1 \cdot n_R)$$

n_R : 単位時間に到着するタプル数
b : 演算Bが作るハッシュ表のバケット数
f : 演算Bでメモリ制限を超えて追い出されるタプル数

bは，ユニークなグループが何種類存在するか，で決まる → Zipf分布，一様分布での分析
fは，各演算Bにどれだけメモリが割り当てられるか，で決まる → FPTASアルゴリズム

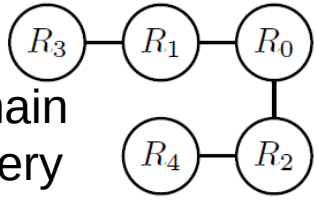
[4本目] A New, Highly Efficient and Easy To Implement Top-Down Join Enumeration Algorithm

- **ICDE2011 Best paper**
- 著者
  - Pit Fender (University of Mannheim)ら
- Contribution
  - 複数リレーションのJoinを効率的に行える実行プランを探すためのアルゴリズム
  - SIGMOD2007で提案されたMINCUTLAZYアルゴリズムの分析と問題点の指摘
    - 特定条件のクエリ(clique)のときに探索コストが$O(n^2)$になる
  - MINCUTBRANCHの提案
    - 上記条件のクエリのときの探索コストはO(1)
    - タイトルの”Easy to implement”は，集合演算だけで実現できるの意

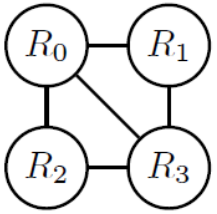
Join最適化について

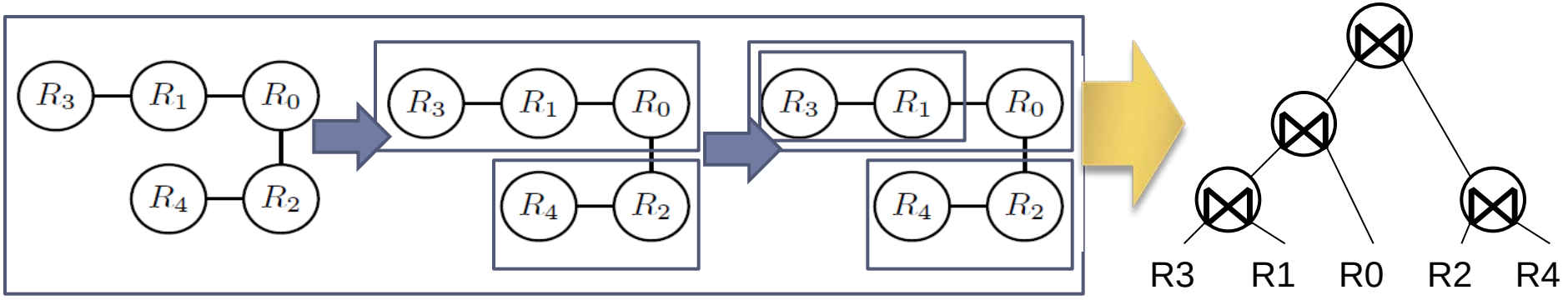
- Binary-Joinの木構造で表される実行プランのうち，コストが最も低いものを求めたい
  - 可能な木を全列挙するとリレーション数に対して指数オーダー
- Query graph　G=(V, E)
  - V(頂点): 問合せに出現するリレーションの集合
  - E(辺): リレーション間の結合条件の有無を表す
- Top-down join enumeration
  - Query graphを部分graphに分割していく問題
    - ただし，部分graph同士をつなぐ辺が必ず存在するように分割

Chain query



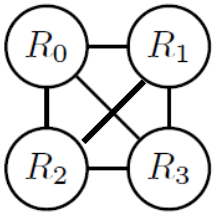
Cycle query





MINCUTBRANCHアルゴリズム

Clique query



- 既存のMINCUTLAZYアルゴリズム
  - Biconnection tree（ノードが2種類: vertex nodeとset node）
  - query graphがクリーク構造の場合，計算時間が$O(|V|^2)$になる（Appendix Bに記述）
- MINCUTBRANCHアルゴリズム
  - Query graphと集合演算で実現
  - 重複した分割をしない工夫
    - 直前に探索したノードLの情報を覚えておき，Lからたどれる近傍ノードを次の探索候補にする

Fig9より引用