DEIM Forum 2021 H34-2(day3 p35)

結合カーディナリティ推定の中間結果を利用した結合順最適化

川本孝太朗† 伊藤 竜一† 佐々木勇和† 肖 川† 鬼塚 真†

† 大阪大学大学院情報科学研究科 〒 565–0871 大阪府吹田市山田丘 1-5

E-mail: †{kotaro.kawamoto,ito.ryuichi,sasaki,chuanx,onizuka}@ist.osaka-u.ac.jp

あらまし 高速なクエリ処理を実現するためには結合順を最適化することが有効である．最適な結合順による実行プランの最適解を求めるには計算コストが大すぎる．また，強化学習によって結合順を選択する場合は学習に時間がかかる問題がある．本稿ではカーディナリティ推定モデルの内部で扱う部分クエリのカーディナリティを活用し最適な結合順を探索する手法を提案する．部分的クエリに関するカーディナリティを利用してコストを計算できる結合順のプランをまとめて探索することで，同じカーディナリティ推論回数でもより多くの候補解から最適な実行プランを選択できる．JOB の 113 クエリに対して PostgreSQL のデフォルトのクエリオプティマイザと提案手法を比較したところ，結合するテーブル数が多いクエリでは 10 15%程度実行性能が良くなることが確認できた．

キーワード 結合順最適化，クエリオプティマイザ，カーディナリティ推定

1 はじめに

近年，大規模なデータベースが扱われるようになりデータ分析のために多くのクエリが実行されている [1] [2]．特に大規模なものだと数千テーブルに渡るデータを扱う場合がある．このような大規模なデータベースでは単純なクエリの処理であっても数時間かかることがある．クエリ処理を高速に行うためには実行性能の高い実行プランを選択する必要がある．実行プランの中でも複数テーブルの結合順がクエリの実行性能に最も影響することが知られている [3]．結合順を最適化するためには候補となるジョインプランを列挙し最もコストの低い結合順序を特定する必要がある．このような最適な結合順を探索するアルゴリズムとして古くから Selinger のアルゴリズムが知られている [12]．サリンジャーのアルゴリズムは動的計画法により Left-Deep な結合順のうち最もコストの小さくなる結合順を求める．しかし，サリンジャーのアルゴリズムはテーブル数に対して指数オーダの計算時間がかかることが知られている．そのため数十テーブル程度の規模であっても実行に数時間から数日かかり，大規模なデータベースにおけるクエリ最適化には利用できない．また，一般的なクエリオプティマイザでは，テーブル数が多い場合は遺伝的アルゴリズムなどの発見的手法を用いて探索時間を削減している [14]．しかし，これらの一般的なクエリオプティマイザで実装されている発見的な探索手法は最適化時間が短いが実行性能の高い結合順を選択できない場合が多いことが問題として知られている．

さらには，最近の研究では深層強化学習による End to End のモデル [8] [9] [10] が成果を上げている．深層強化学習を用いた End to End のモデルはクエリを入力としてコストの低い結合順を出力するように学習する．しかし，1 つのモデルでカーディナリティ推定と結合順探索を両方行うためモデルは大規模になり，学習にかかる時間はカーディナリティ推定だけを行うモデルと比較して長くなる．そのためデータベースの更新に弱い問題がある．

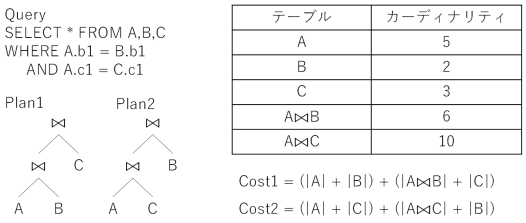

図 1 結合順によるコストの違い

本稿ではこれらの問題を解決するために機械学習を用いたカーディナリティ推定の特性を利用した結合順探索手法を提案する．提案手法では，使用するカーディナリティ推定モジュールは推定する際に内部で扱う部分クエリに対するカーディナリティを活用することを特徴とする．部分クエリに対するカーディナリティを活用することによって 1 回の推定で複数の近傍解のコストを計算できる．これにより同じ推論回数であっても従来手法より多くの解候補を探索できることから従来手法と同等の推論回数でより実行性能の高い結合順を選択できる可能性が高い．

サリンジャーのアルゴリズムではテーブル数 N に対して $O(2^N)$ 回カーディナリティ推定を行う必要があるが，提案手法ではカーディナリティ推定の回数を $O(N^2)$ に削減する．Join Order Benchmark の 113 クエリに対して PostgreSQL のデフォルトのクエリオプティマイザで生成した実行プランと提案手法で生成した実行プランのそれぞれの実行時間を比較したところ，結合するテーブル数が少ないクエリでは PostgreSQL と提案手法は同等の実行性能を示し，結合するテーブル数が多いクエリでは 10〜15%程度実行性能が良くなることが確認できた．

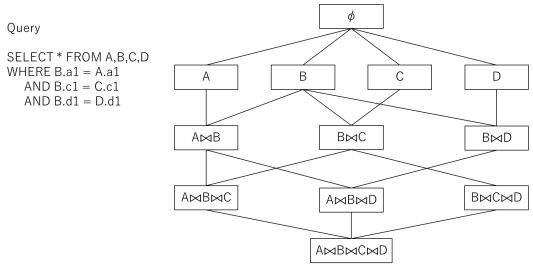

図 2　プラングラフの例

2　事 前 知 識

2.1　結合順のコスト計算

同じクエリであっても複数の結合順によるクエリプランを取ることができる．例えば図 1 に示すクエリを実行する場合にはテーブル A とテーブル B を先に結合してからテーブル C を結合する場合と，テーブル A とテーブル C を先に結合してからテーブル B を結合する場合の 2 通りの結合順が考えられる．このように，複数の結合順の中から最も実行性能が高くなる結合順を探索するアルゴリズムを，結合順最適化アルゴリズムと呼ぶ．結合にかかる時間は結合元のカーディナリティに比例することが知られており $A \bowtie B$ のコストは $|A| + |B|$ と計算する．3 テーブル結合する場合も同様に計算して $(A \bowtie B) \bowtie C$ のコストは $(|A| + |B|) + (|A \bowtie B| + |C|)$，$(A \bowtie C) \bowtie B$ のコストは $(|A| + |C|) + (|A \bowtie C| + |B|)$ となる．この式からも結合するテーブルの集合が同じ場合でも結合順によってコストが変わることがわかる．また結合していない状態のそれぞれのテーブルのカーディナリティは結合順によらず共通して 1 回ずつ足されることからコスト計算の際に無視して考えても良い．共通項を無視して考えた場合，図中に示すプラン 1 のコストは $|A \bowtie B|$, プラン 2 のコストは $|A \bowtie C|$ となり途中の結合状態のカーディナリティの総和が実行性能に影響することがわかる．図 1 に示す例では $|A \bowtie C|$ より $|A \bowtie B|$ の方が小さいため $(A \bowtie B) \bowtie C$ のように結合する方が実行性能が良くなる．

2.2　プラングラフ

プラングラフを考えることで結合順最適化の問題をグラフの問題に置き換えて考えることができる．プラングラフとはあるクエリに対してその任意の部分クエリをグラフの頂点，辺の重みを結合にかかるコストとしたグラフである．プラングラフで空集合から全テーブル結合状態までのパスが 1 つの結合順を示し，パスに含まれる辺の重みの総和がその結合順のコストとなる．プラングラフを考えることで結合順最適化を最短経路問題に置き換えて考えることができる．図 2 にクエリとそれに対応するプラングラフを示す．プラングラフの頂点は使用するテーブルをいくつか結合した状態となっていることが確認できる．辺の重みは結合にかかるコストを表し $A \bowtie B$ から $A \bowtie B \bowtie C$ への辺の重みは $|A \bowtie B| + |C|$ となる．また 2.1 で説明したように共通項を無視したコストを考えると結合順のコストはパスに含まれる結合状態のカーディナリティの総和となり見通しが良くなる．

2.3　結合順最適化アルゴリズム

結合順最適化アルゴリズムとして古くからサリンジャーのアルゴリズムが知られている．結合順を全探索する場合，時間計算量はテーブル数を N として $O(N!)$ となる．サリンジャーのアルゴリズムでは最適な Left-Deep な結合順を動的計画法で探索する．実行性能が最も高い結合順は Left-Deep な結合順である確率が高いことが統計的に知られているため Left-Deep な結合順に限定してよい．結合済みのテーブルの集合を状態としてもち，各状態についてその状態に到達するための最小コストを保持する．初めは空集合以外の状態はコスト ∞，空集合はコスト 0 で初期化し．空集合から順に結合可能なテーブルを 1 つ結合する遷移を行なって各状態のコストを求める．計算量はテーブル数を N として $O(2^N)$ となる．

サリンジャーのアルゴリズムを用いることで Left-Deep な結合順のうち最適なものを計算できるが，テーブル数に対して指数オーダの計算時間がかかる．そのため大規模なデータベースでは遺伝的アルゴリズムなどの発見的な手法を用いる必要がある．発見的な手法を用いる場合，一般的に解空間の大きさと得られる解の精度がトレードオフの関係となる．結合順探索アルゴリズムでは少ない探索範囲で高い精度で解を求める必要がある．

2.4　カーディナリティ推定

クエリオプティマイザを構成する要素としてカーディナリティ推定がある．カーディナリティ推定とはあるクエリを実行した結果のカーディナリティを予測する技術である．テーブルを結合する操作にかかる時間は結合対象テーブルのカーディナリティに比例することが知られており，結合順探索の際のコスト計算においてカーディナリティ推定を利用する．一般的なデータベースエンジンではインデックスや統計情報を作成する際に自動で作成されるヒストグラムをもとにカーディナリティを推定している．単一属性のカーディナリティ推定には有効だが，ジョイン操作などによって生じる複数属性に跨るカーディナリティ推定の精度が高くない．この原因は，単一属性のカーディナリティを独立事象と捉えて，複数属性に跨るカーディナリティを推定するためである．　近年は機械学習を用いたカーディナリティ推定が注目されている．機械学習を用いたカーディナリティ推定にはクエリから直接カーディナリティを推定する手法と結合確率分布に基づいてカーディナリティを推定する手法がある [15]．クエリからカーディナリティを推定する手法では，クエリを特徴ベクトルに変換したものとカーディナリティ推定結果のペアを学習する．特徴ベクトルにはクエリの情報だけでなくいくつかの統計情報も含むことができる．推論の際には学習と同じようにクエリを特徴ベクトルに変換し，その特徴ベクトルをモデルに適用することで結果を得る．代表的なものとして MSCN [5], LW-XGB [4], LW-NN [4], DQM-Q [6] が知られている．結合確率分布に基づいてカーディナリティを推定する手法ではクエリに含まれるテーブルを単に結合したカー

ディナリティのうち条件を満たすカーディナリティがどれぐらいの確率で存在するかを計算してカーディナリティを推定する．

$$Cardinality = |A_1 \bowtie A_2 \bowtie \cdots \bowtie A_N| \times P(A_1, A_2, \cdots, A_n)$$

このとき確率の乗法定理により次のように書き換えることができる．

$$Cardinality = |A_1 \bowtie A_2 \bowtie \cdots \bowtie A_N| \times P(A_1)P(A_2|A_1)\cdots P(A_n|A_1, \cdots, A_{n-1})$$

結合確率分布に基づく手法では自己回帰モデルによって結合確率分布を学習し，上記の式に基づいてカーディナリティを計算する．代表的なものとして Naru [7] や DQM-D [4] が知られている．

2.5 カーディナリティ推定モジュール

提案手法では主キー・外部キーによって結合されるテーブルとの関係からカーディナリティの計算をするモジュールを利用する．例えば R, S, T の 3 テーブルを結合した結果のカーディナリティを推定するとき，モジュールの内部では次のような式に沿って計算される．

$$Cardinality(\theta) = |R \bowtie S| \cdot P(R, S) \cdot F(S \leftarrow T) \cdot P(T|R, S)$$

$F(S \leftarrow T)$ は S に T を結合する時にカーディナリティがどれだけ変化するかを示すベクトルである．$|R \bowtie S| \cdot P(R, S)$ は部分クエリのカーディナリティである．右辺に着目すると部分的クエリのカーディナリティに $F(S \leftarrow T)$ を掛けていることから、カーディナリティ推定モジュール内部ではまず部分クエリに対するカーディナリティを計算し，テーブルを結合する際のカーディナリティの変化量から目的となるカーディナリティを推定していることがわかる．部分クエリのカーディナリティを推定する順番は複数考えられるが，その中で最も推定精度の高くなる順序でカーディナリティが計算される．

3 提 案 手 法

カーディナリティ推定の際に同時に得られる部分クエリに対する推定結果を利用して結合順を探索する手法を提案する．提案手法ではカーディナリティ推定の際に同時に得られる部分クエリのカーディナリティを利用することで 1 回の推論で従来より多くの結合順を評価する．よってカーディナリティ推定を実行する回数を増やさずにより多くの候補解を探索することが可能になる．最終的により多くの候補解から解を選択できることから，少ない推論回数でも性能の高い解が選択できることを期待している．

3.1 探索アルゴリズム

提案手法はシミュレーテッド・アニーリングによって最適解の探索を行う．シミュレーテッド・アニーリングとは大域的最適化問題への汎用の乱択アルゴリズムであり，初期解から解の改善を繰り返すことによって最適解へと近い解を出力することを目指す．山登り法と異なり局所的な最適解にとらわれにくい性質がある．

3.2 初 期 解

まずクエリ全体のカーディナリティを推定し，その時にモジュール内部で扱われる推定順を初期解とする．例えば R，S，T を結合した状態のカーディナリティを計算する際に内部で $Cardinality(\theta) = |R \bowtie S| \cdot P(R, S) \cdot F(S \leftarrow T) \cdot P(T|R, S)$ と計算されていたとする．$(R \bowtie S) \bowtie T$ のコストは $(|R| + |S|) + (|R \bowtie S| + |T|)$ となるが $|R|, |S|, |T|$ はクエリによらず一定であるため，カーディナリティ推定モジュール内部で扱われる $|R \bowtie S|$ を利用することでコストが計算できる．テーブル数が多い場合でも同様にクエリ全体のカーディナリティを計算する際の途中結果を利用することで 1 つの結合順のコストが計算できる．この時コストが計算できる結合順はカーディナリティ推定モジュール内部推定精度が高くなる順序であり，コストが小さくなる順序とは限らないため解の精度を改善する必要がある．

3.3 候補解の改善

候補解の結合順から隣接するテーブルの結合順を 1 箇所だけ入れ替えた結合順を考える．このような結合順は候補解とコストの計算式がほぼ同じになるため，候補解から変化する部分だけに着目すると 1 つ部分クエリのカーディナリティを計算するだけでコストが計算できる．便宜上隣接するテーブルの結合順を 1 箇所だけ入れ替えた結合順を近傍と定義する．候補解を改善するには候補解の近傍からランダムに 1 つの結合順を選択しそのコストを計算する．コストが小さくなるようなら新たな候補解として採用，コストが大きくなるようなら確率的に採用する．コストが小さくなる場合のみ新たな候補解として採用する場合は局所解にとらわれる可能性があるが，コストが大きくなる場合も確率的に採用することによって局所解にとらわれにくくなる．コストが大きくなる解も採用すると解が収束しない可能性がある．この対策として探索が進むにつれてコストが大きくなる解を選択する確率を小さくし，最終的にはコストが小さくなる解しか選択しないようにすることで対策する．

新しい候補解のコストを計算する際にはカーディナリティ推定を行うが，このとき同時に得られる部分クエリに対するカーディナリティからコストが計算できる結合順も候補解に加える．これによって 1 回の推定で複数の解を探索することが可能になる．

例えば図 3 のプラングラフを考えたときに赤のパスで示す結合順が候補解であるとする．候補解から結合順を 1 つ入れ替えた $((A \bowtie B) \bowtie D) \bowtie C$ のコストを計算する際に $A \bowtie B \bowtie D$ のカーディナリティを推定する．それと同時に図中に青で示す部分クエリのカーディナリティが得られたと仮定する．同時に得られる部分クエリのカーディナリティを使うことで $((B \bowtie A) \bowtie D) \bowtie C$ のコストも計算することができるため 1 度のカーディナリティ推定で複数の結合順のコストが計算

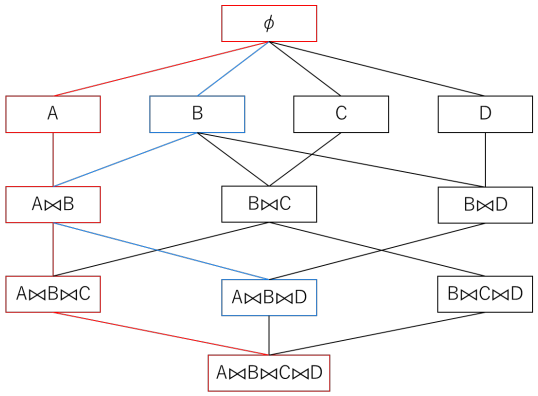

図 3　候補解の改善

できることがわかる．したがって同じ推論回数の場合はより多くの候補解から最終的な解を選ぶことになり，実行性能の高い結合順を選択する確率が高くなる．

3.4　終 了 条 件

提案手法では反復して候補解の改善を行うため長い時間をかければいずれ最適解にたどり着く．しかし最適化に長い時間をかけてしまうと全体としてもパフォーマンスは下がってしまうため高速に解を出力する必要がある．一般的には解が収束するまで反復するが，一定時間で終了することを保証するためにループ回数を条件に終了することにした．テーブル数 N に対して N^2 回のループで終了することにした．最適な結合順を求めるためにはテーブル数 N に対して $O(2^N)$ の時間計算量がかかることに対して十分に高速に動作するループ回数を設定した．

4　実　　験

本章では，一般的な DBMS である PostgreSQL と比較して提案手法の有効性を評価する．

4.1　ベンチマーク

実験には Join Order Benchmark（JOB）のクエリを使用して従来手法との精度の違いを比較する．JOB は Internet Movie Database（IMDB）のデータに対して 3〜16 テーブルの結合を含む 113 クエリからなるベンチマークである [11]．クエリはランダムに生成されており，カーディナリティ推定や結合順最適化アルゴリズムの評価に広く利用されていることから本実験のワークロードとして適している．

4.2　実 験 環 境

JOB のクエリを実行した際の実行時間の評価実験を行った．具体的な実験環境は以下である．マシンは MacBook Pro (Monterey,　15-inch,　2019)．cpu は 2.3 GHz オクタコア Intel Core i9．メモリは 32GB 2400 MHz DDR4．クエリの実行には PostgreSQL を使用する．

4.3　実 験 方 法

従来手法の代表として PostgreSQL と比較して性能を検証する．JOB の 113 クエリに対して提案手法を用いて生成した実行プランと PostgreSQL で生成した実行プランとで実行速度を比較した．クエリ最適化にかかる時間は除いて実行時間を計測した．計測する際のマシンの状態による誤差を小さくするためにそれぞれ 10 回ずつ実行し平均実行時間を調べた．

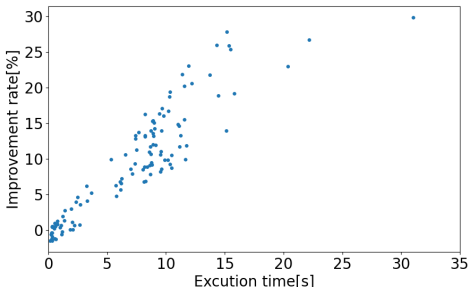

図 4　実行時間の改善率

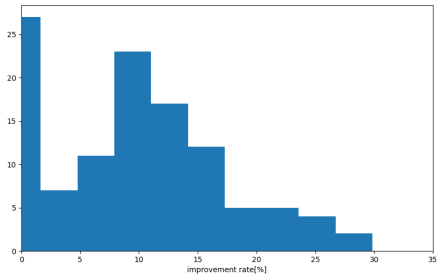

図 5　実行時間の改善率の統計結果

4.4　実 験 結 果

JOB の 113 クエリの実行時間を散布図にまとめた．実験結果を図 4 に示す．横軸は PostgreSQL での実行時間，縦軸は提案手法での改善率を示す．全体的な傾向としては PostgreSQL での実行時間におおよそ比例して改善率が上がることがわかる．クエリの実行時間に着目すると 3 秒未満の部分とと実行時間が 10 秒程度の部分に結果が集中していることがわかる．実行時間が 3 秒未満のクエリでは実行速度の改善はほとんど見られなかった．改善率は-2% 2%程度と中には実行速度が悪化しているクエリもあることが確認された．実行速度が 10 秒程度のクエリでは平均して 10%程度実行速度が改善されていることが分かった．最も改善率の高いクエリでは 30 %ほどの高速化に成功していることが確認できる．

クエリ全体の傾向を評価するために改善率をヒストグラムにまとめた．その結果を図 5 に示す．従来手法と比較してほぼ改善されていないクエリが最も多く，27 クエリで改善がみられないことが確認できた．10〜15 %程度実行時間が高速化されたクエリが合計で 52 クエリあり約半数のクエリでは 10 %程度実行時間が改善されていることがわかる．20 %以上実行時間が高速化されたクエリは 11 クエリ存在することが確認できた．

図 4 によると改善率が低いクエリは実行時間が短いクエリに多くみられることから，その原因を確かめるために PostgreSQL で実行時間をまとめた．PostgreSQL で実行時間が短い上位 10 クエリの実験結果を表 1 に示す．PostgreSQL で実行時間が短

表 1　実行時間の短い 10 クエリの結果

クエリ番号	テーブル数	PostgreSQL[s]	提案手法 [s]	改善率 [%]
15	5	0.134	0.136	-1.49
12	5	0.228	0.230	-0.878
2	5	0.266	0.267	-0.376
40	8	0.291	0.294	-1.03
3	5	0.294	0.292	0.680
108	6	0.296	0.294	0.676
1	5	0.296	0.297	-0.338
13	5	0.302	0.300	0.662
109	6	0.333	0.338	-1.50
39	8	0.367	0.365	0.545

表 2　実行時間の長い 10 クエリの結果

クエリ番号	テーブル数	PostgreSQL[s]	提案手法 [s]	改善率 [%]
95	12	14.312	11.364	25.9
59	7	14.467	11.737	18.9
28	7	15.136	13.020	14.0
20	5	15.162	10.947	27.8
56	8	15.333	11.364	25.9
71	10	15.481	11.558	25.3
25	8	15.811	12.783	19.1
83	11	20.381	15.702	22.9
82	11	22.165	16.247	26.7
84	11	31.017	21.758	29.9

いクエリと提案手法の実行時間の差は大きいものでも 5ms 程度と誤差程度の変化しか見られないことが確認できた．それぞれ生成された実行プランを表示すると結合順が同じことが確認できた．

最も実行時間の短かったクエリは 5 テーブルの結合を含むクエリであったが，結合条件が厳しくテーブルを結合した結果のカーディナリティは 4 と極端に少なかった．実行時間が短いクエリはこのように簡単なクエリが多く，従来手法でも最適な実行プランが選択できていたと考えられる．PostgreSQL と提案手法の両方で最適な結合順が選択できていたことにより実行時間の改善が見られなかったと考えられる．-2 %〜1 %程度の性能変化が見られたが元の実行時間が短いため時間に直すと数ミリ秒程度となり誤差と考える．

PostgreSQL で実行時間が長かった上位 10 クエリの実験結果を表 2 に示す．実行時間の長いクエリでは PostgreSQL と比較して提案手法で生成した実行プランの方が明らかに実行時間が短くなっていることが確認できた．改善率の低いものでも 15 %，最も改善率の高いもので 30 %近く高速化できていることが確認できた．生成された実行プランを表示すると，提案手法では中間状態のカーディナリティが小さいことが確認できた．これほど性能に差が出た原因としては PostgreSQL のカーディナリティ推定の精度が低いことが考えられる．PostgreSQL で利用されているカーディナリティ推定手法は高速に動作するが機械学習を用いた手法と比較して推定精度が低い．一部のクエリでは実際のカーディナリティと比較して 10 倍以上のカーディナリティを予測しているものもあり，これが原因で実行性能の低い結合順を選択していたと考えられる．

またテーブル数の多いクエリで高い改善率が得られた要因としては，テーブル数が多いクエリほどカーディナリティ推定の際に得られる部分クエリの情報が多くなることも挙げられる．カーディナリティ推定モジュール内部では精度が高くなる推定順に従って順に結合したカーディナリティを計算する．そのため結合したテーブルの数と同じ数の部分クエリのカーディナリティも同時に計算している．よって結合するテーブル数におおよそ比例して探索する結合順の候補が増えることから PostgreSQL での実行時間が長いクエリほど顕著に差が現れたと考えられる．

5 結　　論

本稿ではデータベースにおける結合順最適化問題に取り組み，カーディナリティ推定モジュール内部で扱う部分クエリのカーディナリティを利用した探索手法を提案した．従来手法であるサリンジャーのアルゴリズムではテーブル数 N に対して $O(2^N)$ 回カーディナリティ推定を実行する必要があることに対して，提案手法では推論回数を N^2 に抑えることに成功した．また推論回数を少なくした上で従来手法と比較して 10 15%実行性能が高い結合順を選択できることを確認した．また強化学習を利用した End to End のモデルではカーディナリティ推定と結合順最適化の両方をモデルに学習させていることに対して提案手法ではカーディナリティ推定部分にしか機械学習を利用していないことから，更新に強いことも期待できる．今後の課題点としては以下の点が挙げられる

- 大規模なクエリに対して提案手法の性能を評価する
- 動的な環境において実験を行い更新にに対する強さを確認する
- 機械学習によるカーディナリティ推定を利用した他の手法と比較する
- カーディナリティ推定モジュールの推論順の特性を利用する

サリンジャーのアルゴリズムでは時間計算量が大きすぎる問題を解決するために効率的な探索手法を提案したが，JOB に含まれるクエリは 3 テーブル〜16 テーブルの結合クエリであり大規模とは言えない．多くのクエリオプティマイザではテーブル数が少ない場合はサリンジャーのアルゴリズムが利用されており，PostgreSQL では結合するテーブルが 12 テーブル未満の場合サリンジャーのアルゴリズムが用いられる．よって大規模なデータベースに対する性能を評価するためには最低でも 12 テーブル以上の結合クエリに対して実験する必要がある．

また強化学習を用いた End to End のモデルが更新に弱い問題に対して，機械学習を使う部分をカーディナリティ推定だけに絞ることで対策したが，この対策が本当に有効かどうかの検証は行えていない．そのため動的な環境において実験を行いデータベースの更新に対してどれだけ強いかを検証する必要がある．

また今回は従来手法として PostgreSQL と比較し実験を行ったが，PostgreSQL 内部で利用されているカーディナリティ推定手法はヒストグラムを用いた手法であり，高速に動作するが推定精度は低い．今回の実験ではカーディナリティ推定モジュールを統一していないため，単に機械学習によるカーディナリティ推定が優れていて探索手法は優れていない可能性を否定できない．提案手法の有用性を示すためにはカーディナリティ推定モジュールを統一した上での実験が必要となる．

提案手法ではカーディナリティ推定モジュール内部で扱われる部分クエリに対するカーディナリティ推定結果だけを探索に利用した．この部分クエリはカーディナリティ推定の精度が高くなるように選ばれている．推定精度が高くなる順序とコストが小さくなる結合順は関係がないと考えて提案手法では部分クエリのカーディナリティだけを利用することにしたが，推定精度が高くなる推定順とコストが小さくなる結合順が統計的に無関係とは言い切れない．よってこれらの関係性を調べることによってより効率的な探索手法を発見できる可能性がある．

今後の研究ではこれらの課題点を解決するために実験・検証を行い，より高速なクエリ処理が実現できるよう貢献していきたいと考えている．

文　献

[1] R.Ramakrishnan, B.Sridharan, J. Douceur, P.Kasturi, B.Krishnamachari-Sampath, K.Krishnamoorthy, P.Li, M.Manu, S.Michaylov, R.Ramon, N.Sharman, Z.Xu, Y.Barakat, C.Douglas, R.Draves, S.S.Naidu, S.Shastry, A.Sikaria, S.Sun, and R.Venkatesan, “Azure data lake store: A hyperscale distributed fileservice for big data analytics,” SIGMOD, pp.51–63, 2017.

[2] A.Y.Halevy, “Answering queries using views: A survey,” VLDB, pp.270–294, 2001.

[3] Leis, V., et al. How Good Are Query Optimizers, Really? VLDB 2015.

[4] S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das. Deep learning models for selectivity estimation of multi-attribute queries. In Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020, pages

[5] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig. Deepdb: Learn from data, not from queries! Proc. VLDB Endow., 13(7):992–1005, 2020.

[6] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings. www.cidrdb.org, 2019.

[7] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, P. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica. Deep unsupervised cardinality estimation. Proc.

[8] R.Marcus, O.Papaemmanouil, ”Deep Reinforcement Learning for Join Order Enumeration” aiDM 2018 Article No.: 3Pages 1–4

[9] P.Negi, R.Marcus, A.Kipf, H.Mao, N.Tatbul, T.Kraska, M.Alizadeh ”Flow-Loss: Learning Cardinality Estimates That Matter” VLDB 2021 pp. 2019 - 2032

[10] Krishnan, Sanjay et al. “Learning to Optimize Join Queries With Deep Reinforcement Learning.” ArXiv abs/1808.03196 (2018): n. pag.

[11] V.Leis, A.Gubichev, A.Mirchev, P.Boncz, A.Kemper, T.Neumann ”How Good Are Query Optimizers, Really?” Proceedings of the VLDB EndowmentVolume 9Issue 3November 2015 pp 204–215

[12] Selinger optimizer https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-830-database-systems-fall-2010/lecture-notes/MIT6_830F10_lec09_selinger.pdf

[13] M. Steinbrunn, G. Moerkotte, A. Kemper, Heuristic and randomized optimization for the join ordering problem, VLDB 1997 pp. 191-208

[14] PostgreSQL:GEQO https://www.postgresql.jp/document/9.4/html/geqo

[15] X.Wang, C.Qu, W.Wu, J.Wang, Q.Zhou ”Are we ready for learned cardinality estimation?” Proceedings of the VLDB EndowmentVolume 14Issue 9May 2021 pp 1640–1654