DEIM Forum 2022 J21-1

大規模ワークロードにおける局所探索法を用いた実体化ビュー選択

乗松　奨真†　　佐々木勇和†　　鬼塚　　真†

† 大阪大学情報科学研究科　〒 565–0871 大阪府吹田市山田丘 1-5
E-mail: †{norimatsu.shoma,sasaki,onizuka}@ist.osaka-u.ac.jp

あらまし　クエリ数が膨大なワークロードにおいて高速なクエリ処理のために，実体化ビューの利用が有効である．実体化する部分クエリの選択が重要となるが，実体化するクエリを整数計画問題として解くことは解の探索空間が膨大であることと制約条件の数が膨大になるため実行が困難である．本稿では，局所探索法を利用した手法を提案する．これは総利得の高い部分クエリの近傍は同様に総利得が高い傾向があるためである．有効な局所探索法のためには，初期解の精度も重要であるため，ワークロードのおける様々な要素の top-k を初期解として複数用意する．近傍探索により解候補を改善し，新たな解候補に対し整数計画問題を解く．得られた解に対して再び近傍探索を行う．この処理を繰り返すことで最適解へと漸近する．局所探索法により整数計画問題における探索空間と制約条件の問題を解決できる．実験では Join Order Benchmark を利用し，最新技術である BIGSUBS の問題点を明確にすると共に，提案手法の有効性を示した．

キーワード　実体化ビュー，整数計画問題，データベース

1　は じ め に

近年データベースシステムに対して多くのユーザが存在し，大量のクエリが実行されている [1] [2] [3] [4] [5]．このことにより，一つの企業内で数十万単位で日々のクエリが実行されている場合がある [6]．そして，これらのクエリの処理速度を向上させることが大きな課題となっている．

データベース内で行われるクエリ処理には多くの共通した処理が含まれている [1] [2] [7]．この共通したクエリ処理の結果を実体化ビューとして保持し，実体化ビューを再利用することで，高速にクエリ処理を実現できるようになる [1] [2] [7] クエリ処理の性能は利用する実体化ビューに依存するため，実体化ビューとして保存する処理を適切に決めることが重要になる．実体化ビューを適切に評価するために利得を用いる．実体化ビューによってワークロードに含まれるクエリ処理がどの程度高速になるかがその実体化ビューの利得と定義する．最適な実体化ビューを選択する際に，考慮すべき観点には．実体化ビューを利用することによって得られる利得と，実体化ビューの更新処理の負荷が挙げられる．これら検索処理性能と更新処理性能はトレードオフの関係にある．具体的には処理に時間がかかるクエリ処理を実体化する場合，これを実体化して得られる利得は高くなるため，実体化ビューを多く作れば作るほど，高い利得が得られる．しかし，テーブルに対して更新が行われると，そのテーブルに関するクエリ処理の結果が変わってしまうため，クエリによりテーブルが更新された場合は関連する実体化ビューにも更新する必要があり，関連する実体化ビューが多い場合はその処理に時間がかかる．さらに，実体化ビューはクエリの処理結果を物理的に保存するため，容量を要する．よって，実体化ビューの保存容量をどれほど許容するかを示すストレージ容量制約が存在する．ストレージ容量制約を課すことで，実体化ビューの個数が制限されるため，更新性能の過度な劣化を防ぐことができる．これらのトレードオフや制限を考慮しつつ，全ての実体化ビュー候補に対し，整数計画問題を解くことで最適な実体化ビュー群を解として得ることができる．しかしクエリ数が多いと，その導出に時間がかかるという問題点がある．このような背景から，最適な実体化ビューを高速に選択する手法が提案されている [1] [6] [7] [8] [9] [10]．例えば BIGSUBS [6] では，巨大なワークロードを対象としているが，実体化ビュー選択問題とクエリが利用する実体化ビューを決定する問題を分けて考えている．実体化ビュー選択問題では利得と容量を考慮した確率を用いる．クエリが利用する実体化ビューを決定する問題は整数計画問題を用いる．

この手法では，利得とストレージ容量制約の両方を考慮し，容量当たりの利得の良い実体化ビューのみを選択できる．しかし，ストレージ容量制約を使い切り利得を最大化する最適解から離れてしまうという弱点が存在する．Wide-deep [7] では，上記の BIGSUBS では解の正確性が低いことを問題点とし，確率での実体化ビュー決定に強化学習を導入している．この手法により，解の正確性は向上するが，学習時間がかかってしまうという問題点がある．

本稿では，近似解の精度を高く保ちつつ，最適化を高速化する手法を提案する．具体的には，メタヒューリスティックである局所探索法 [11] を利用することで解の探索を高速化する．局所探索法を適用するには，初期解の選択方法および近傍解を適切に決定することが技術課題である．初期解の選択方法に関しては，利得が高い解を高速に選択できるように，サブクエリに関する情報を用いて，利得，容量当たりの利得，ワークロード内での頻度のそれぞれ $top-k$ を初期解とする 3 つの方法を提案する．次に，近傍解を適切に決定する方法においては，包含関係にあるサブクエリ同士はそれらの利得が類似するという特性を活用する．具体的には，包含関係にある親サブクエリの方

が子サブクエリよりもクエリ表現が複雑であるため，実体化することで得られる利得がより大きく，逆に子サブクエリの方がより多くのクエリで利用される可能性が高い．このため，解候補を近傍解に広げることでより優れた解を探索できる可能性がある．この特性を活用して，提案手法ではクエリの木構造において包含関係にある 2 つのサブクエリが近傍の関係にあると定義し，この定義に従って局所探索法における近傍解を決定し，解が収束するまで最適解の探索を実行する．

実験は Join Order Benchmark のデータとクエリセットを用いる．実験により，最新技術である BIGSUBS [6] の問題点を明確にすると共に，提案手法の有効性を示した．

本稿の構成は，次の通りである．2 節で事前知識，整数計画問題での定式化を説明する．3 節で提案手法の詳細について説明する．4 節で実験について述べる．5 節で関連研究について，6 節で本稿のまとめ，今後の課題を述べる．

2 問題定義

提案手法では，局所探索を用いて整数計画問題を高速化する．整数計画問題で用いられる変数の詳細について 2.1 節において前提知識として説明する．次に，2.2 節でワークロードにおける利得を最大化する最適化問題を解くために用いられる整数計画問題の定式化を説明する．本稿では，ワークロード W に含まれるクエリ数を n，サブクエリ数を m として，W に含まれるクエリ $q_i(1 \leqq i \leqq n)$ 集合を Q，サブクエリ $s_j(1 \leqq j \leqq m)$ の集合を S とする．また，クエリの実行プランは図 2 のように木構造で表すことができる．

2.1 事前知識

サブクエリの利得: 実体化ビュー選択問題はワークロード全体の実行コストを下げる実体化ビュー群を選択するものである．この実体化ビューを適切に評価するために利得を用いる．利得は実体化ビューを利用したクエリと利用しないクエリの実行コストの差で計算する．サブクエリの実行コストは，クエリ最適化で用いられる推定実行コストを用いる．

BIGSUBS [6] の定義に従って利得を以下のように定義する．

$$u_{ij} = cost(i) - cost(i|s_j) \tag{1}$$

但し，実体化ビューを利用しないで q_i を処理する際のコストを $cost(i)$，s_j を実体化したビューを利用して q_i を処理する際のコストを $cost(i|s_j)$ とする．図 1 は上記を図であらわしたものである．

サブクエリの重複: ワークロード内の異なるクエリ間で重複したサブクエリが存在する場合がある．図 2 にその例を示す．サブクエリ s_1 がクエリ q_1, q_2, q_3, q_4 に包含され，s_2 がクエリ 2, 4 に共通し，s_3 がクエリ 2, 3 に共通している．このような重複したサブクエリを実体化ビューとして保存し，再利用することで，複数のクエリの高速化され，ワークロード全体の実行が高速化される．また，このように複数クエリにまたがって同じサブクエリが存在するために，本論文における実体化ビュー選択問題は複雑になる．

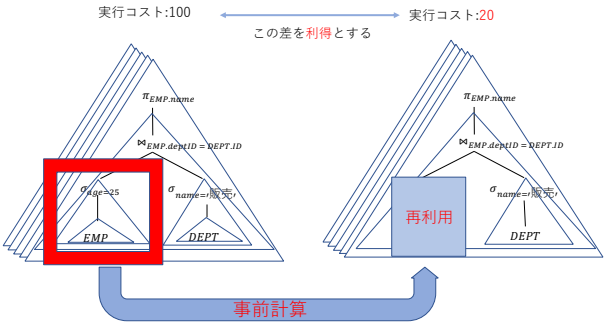

図 1　サブクエリを実体化することによって得られる利得

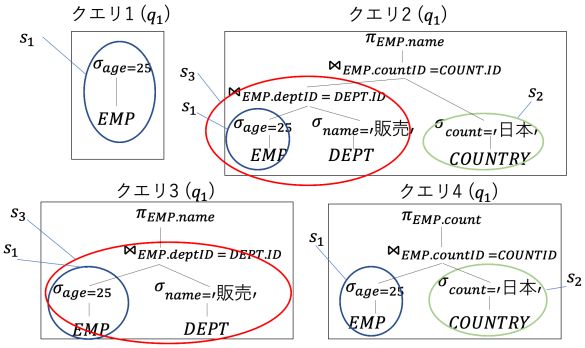

図 2　異なるクエリ間でのサブクエリの重複

2.2 整数計画問題を用いた実体化ビュー選択

この節では整数計画問題の定式化について説明する．整数計画問題に対する入出力は以下である．

- 入力:W，u_{ij}，b_j，x_{jk}.
- 出力: 実体化するサブクエリ群，得られる合計利得.

また，整数計画問題扱う変数は以下である．b_j は Q 内に存在する全てのサブクエリ s_j の容量，z_j は s_j が実体化される場合に 1 をとるバイナリ変数，y_{ij} は q_i が実体化された s_j をクエリ処理の際に利用する場合に 1 をとるバイナリ変数，x_{jk} は s_j が s_k を包含する場合に 1 をとるバイナリ変数とする．W に含まれる各クエリを $q_i \in Q$ として，定式化は以下である．

$$maximize \sum_{i=1}^{n}\sum_{j=1}^{m} u_{ij} \cdot y_{ij} \tag{2}$$

$$s.t. \sum_{j=1}^{m} b_j \cdot z_j \leqq B_{max} \tag{3}$$

$$y_{ik} + \frac{1}{m}\sum_{\substack{j=1 \\ j \neq k}}^{m} y_{ij} \cdot x_{jk} \leqq 1 \qquad \forall i \in [1,n], k \in [1,m] \tag{4}$$

$$y_{ij} \leqq z_j \qquad \forall i \in [1,n], j \in [1,m] \tag{5}$$

(2) 式がこの最適化問題の目的関数である．この目的関数は W 全体における実体化ビューの利得を最大化するものである．(3) ～ (5) 式は最適化を行う際に，満たすべき制約を示している．(3) 式は実体化ビューの容量を制限するストレージ制約である．この制約はストレージ制約を課している．また，ストレー

ジ制約により実体化ビューの個数は制限され，データベース更新があった場合の実体化ビュー再作成コストの過度な劣化を防いでいる．(4) 式はサブクエリのコストを重複して評価しないための制約である．この制約に関しては後述の 2.3 節で説明する．また，(2) 式で，$\sum_{i=1}^{n} y_{ij}$ が 1 以上となる j の集合を M_j とする．この場合，$s_k(k \in M_j)$ の実体化ビューは 1 つ以上利用されている．つまり，$b_k = 1(\forall k \in M_j)$ とする必要がある．この制約式が (5) 式である．

2.3 重複評価制約

ある実体化ビューをクエリ処理において利用する場合には，その実体化されたサブクエリが包含するサブクエリの実体化ビューが同じクエリ処理において利用されないようにする必要がある．式 (2) の整数計画問題を解くと，(4) 式の制限を設けない場合サブクエリの利得を重複して評価する場合があるが，これは適切な評価ではない．この問題を解決するのが，(4) 式であらわされる重複評価制約である．例えば，図 2 において，s_1 と s_3 を実体化する場合を考える．q_2 と q_3 においては，s_3 が s_1 を包含し，q_1 と q_4 では包含していないため，q_2 と q_3 において削減できるコストは s_3 のコストのみである．ここでクエリ q_2，q_3 での利得は s_1, s_3 の利得を合計したものとすると，重複して利得を評価することになり，過剰な利得が得られてしまう．s_3 のみの利得として評価するために，具体的には y_{ik} が 1，つまり q_i が実体化された s_k を利用する場合，q_i が s_k が包含する他のサブクエリを利用できない制限を設ける．s_k が包含する全ての他のサブクエリが q_i によって利用されているか否かは $y_{ij} \cdot x_{jk}$ によって表される．y_ik が 1 の場合は，$y_{ij} \cdot x_{jk}$ は全て 0 である必要がある．y_ik が 0 の場合は，$y_{ij} \cdot x_{jk}$ は自由に選択できる．

(2) ～ (5) 式の整数計画問題は，巨大なワークロードに対しては解の候補が膨大となり，解の導出に時間がかかる．そのため，提案手法では局所探索法を用いて，解の導出の高速化を図る．

3 提 案 手 法

3.1 提案手法概要

提案手法は，メタヒューリスティックな手法である局所探索法 [11] と整数計画問題を用いてワークロード高速化に最も寄与する実体化ビュー群である最適解を探索する．また，実体化ビュー選択問題における解はサブクエリである．局所探索法のアルゴリズムは以下の通りである．まず初期解群を生成する．次に現在の解の近傍を探索して近傍解とする．近傍解が事前に定義した条件を満たすのであれば近傍解を入れ替える．終了条件を満たすまで近傍探索以降を繰り返す．

以降，局所探索法を用いて最適な実体化ビューを選択するアルゴリズムの概要を説明する．まず図 3 のように初期解として，s_j の総利得 U_j，容量 b_j 当たりの U_j，W 内に出現する s_j の頻度 f_j のそれぞれ $top-k$ であるサブクエリ群を選択する．その後図 4 の近傍探索に示すように初期解の近傍に解候補を広げ，得られた解候補に対して整数計画問題を解くことにより解を決

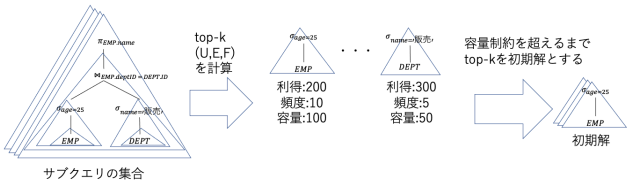

図 3　初期解の決定

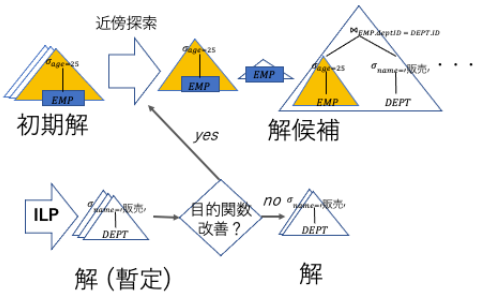

図 4　近 傍 探 索

定する．この解の決定が局所探索法における近傍解の入れ替えに相当する．ここでサブクエリ s_i の近傍クエリ $neighbors(s_i)$ を，s_i を含む親サブクエリおよび s_i に含まれる子サブクエリの和集合として，以下のように定義する．

定義 1 (近傍クエリ)**.**

$$neighbors(s_i) = parents(s_i) \cup children(s_i) \tag{6}$$

但し，$parents$ は引数の親サブクエリを全クエリ群から探索して返す関数であり，$children$ は引数の子サブクエリ群を返す関数である．

以降，近傍を解候補として整数計画問題により解を求める処理を繰り返すことで，最適解を探索する．終了条件は実体化ビューにより得られる利得が改善しなくなる場合とする．局所探索法を利用することで，提案手法の整数計画問題は (2) ～ (5) 式と比較して，解候補が削減されるため処理する変数が削減される．つまり，全ての式において解候補であるサブクエリ s_j の探索範囲が狭まり，解候補に含まれるサブクエリのみを扱えば良くなる．t 回目の反復において整数計画問題で扱う解候補群を $B_t(t = 0 \cdots$ 反復回数) とすると，以下のように定式化される．

$$maximize \sum_{i=1}^{n} \sum_{j \in B_t} u_{ij} \cdot y_{ij} \tag{7}$$

$$s.t. \sum_{j \in B_t} b_j \cdot z_j \leqq B_{max} \tag{8}$$

$$y_{ik} + \frac{1}{m} \sum_{\substack{j \in B_t \\ j \neq k}}^{m} y_{ij} \cdot x_{jk} \leqq 1 \qquad \forall i \in [1,n], k \in [1,m] \tag{9}$$

$$y_{ij} \leqq z_j \qquad \forall i \in [1,n], j \in B_t \tag{10}$$

3.2 提案手法詳細

3.2.1 初期解の列挙

局所探索法を有効に用いるために優れた初期解が必要となる．図 3 のようにワークロードに含まれるクエリのサブクエリに関する統計情報を用いて初期解を決定する．用いる統計情報はサブクエリの利得，容量，ワークロード内での頻度である．この初期解選択方法として 3 つ列挙する．

1 つ目は実体化ビューの利得が高い順に k 個を列挙する手法である．ワークロードに含まれる各クエリ q_i とし，そのサブクエリ s_j を実体化することで得られる利得 u_{ij} の全クエリにおける和，つまり s_j のワークロード全体における総利得 U_j として，この $top-k$ から b_j の総和が B_{max} 満たすまで初期解として選択する ($topk-U$ と呼ぶ)．ワークロード全体における総利得 U_j は以下のように定義される．

$$U_j = \sum_{i=1}^{n} u_{ij} \tag{11}$$

2 つ目は実体化ビューの容量当たりの利得が高い順に k 個を列挙する手法である．ワークロード全体に対して s_j の容量当たりの総利得 U_j を E_j として，この $top-k$ から b_j の総和が B_{max} 満たすまで初期解として選択する ($topk-E$ と呼ぶ)．ワークロードに全体における容量当たりの総利得 E_j は以下のように定義される．

$$E_j = U_j / b_j \tag{12}$$

3 つ目はワークロードにおける s_j の出現頻度が高い順に k 個列挙する手法である．W における s_j の出現頻度 f_j の $top-k$ から b_j の総和が B_{max} 満たすまで初期解として選択する ($topk-F$ と呼ぶ)．$topk-F$ は W での頻度の高い s_j が初期解となり，多くのクエリに共通するサブクエリが初期解となることにより，後の近傍探索により広い範囲での探索が行え，最適解に近似しやすくなる．

3.2.2 近傍探索による解候補の列挙

クエリの木構造において，子サブクエリの方がクエリ表現が簡略であるため，親サブクエリよりワークロード内での出現頻度が高い．また，同様に親サブクエリの方がクエリ処理が複雑であるため，子サブクエリよりクエリ内での利得が高い．

このような理由から，解候補となっているワークロード全体における総利得の高いサブクエリの近傍クエリもまた総利得が高いと推測して解候補に追加する．

3.2.3 整数計画問題の繰り返しによる最適解の探索

提案手法では，整数計画問題と解候補の探索を反復する．以下の式のように解候補 $B_t(t = 0 \cdots$ 反復回数) は一つ前の反復における解 C_t に対して近傍探索をおこなうことで得られる．

$$B_t = C_t \cup \bigcup_{c_{ti} \in C_t} neighbors(c_{ti}) \tag{13}$$

但し，C_0 は 3.2.1 節に示した方法で得られる初期解群とする．C_t $(t > 0)$ の場合は，以下の式によって，解候補 B_t に対し整数計画問題を解くことで得られる解であると定義する．

Algorithm 1 提案手法のアルゴリズム

Input: クエリワークロード (workload)，クエリの木構造，閾値 k，サブクエリのコスト，サブクエリのストレージ，クエリとサブクエリの対応関係

Output: 実体化するサブクエリ，クエリワークロードの総実行コスト

```
1: iteration_end = 0
2: preUtility = 0
3: ans_cand =select_cand(workload, k)
4: while (iteration_end == 0) do
5:     ans_cand =neighbors(ans_cand)
6:     interaction =interaction_check(ans_cand)
7:     result =ILP(ans_Utility, interaction)
8:     Utility = result[0]
9:     ans_cand = result[1]
10:    if (Utility <= preUtility) then
11:        break
12:    end if
13:    preUtility = Utility
14: end while
15: return (ans_Utility, ans_view)
```

$$C_{t+1} = ILP(B_t) \tag{14}$$

ここで ILP は解候補に対して整数計画問題を解く関数である．この反復は得られた解の実体化ビューにより得られる総利得が改善しなくなった場合に終了とする．

3.3 アルゴリズム

提案手法のアルゴリズムを Algorithm1 に示す．$select\_cand$ はそれぞれ E_j, U_j, F_j が上位のサブクエリからサブクエリの容量である b_j の総和が B_{max} を超えない範囲まで初期解として選択する関数，5 行目の $neighbor\_serach$ は近傍探索，$interaction\_check$ は解候補同士の包含関係の確認をする関数であるが，$interaction\_check$ はサブクエリ同士が近傍クエリの関係にあるかどうかを確認することで，クエリが利用できる実体化ビューを制限できるようにする．つまり，(7) ～ (10) 式における整数計画問題における (9) 式の制約条件を満たすためのものである．ILP は整数計画問題を解く関数である．ILP は整数計画問題によって計算された目的関数の値を配列の第 1 要素として返し，実体化に選ばれた解の集合を配列の第 2 要素として返す．

まず 3 行目で 3.2.1 節に示したように初期解を決定する．5 ～ 7 行目で，3.2.2 節で示したように解候補に対して近傍探索と整数計画問題を実行する．10 ～ 12 行目は初期解に対して，または一つ前の反復における整数計画問題でクエリロードの総実行コストが改善されない場合に実行し，反復を終了する．4 行目によりクエリの総実行コストが改善されなくなった時点でアルゴリズムの反復は終了し，15 行目へと進み決定解と総実行コストを出力する．

4 実　　験

実験では，解候補を削減しないで行う整数計画問題 (N_ILP)

により得られる最適解および BIGSUBS [6] と比較して，提案手法の有効性を評価する (実験 1)．また，近傍探索の有効性を評価するために，3 つの初期解に対して整数計画問題を適用した近傍探索を行わない場合との比較を行う (実験 2)．但し，BIGSUBS では反復的アルゴリズムで最後に得られる結果が最適値とは限らないため，解の探索途中で得られた，最も性能が高い解の結果を掲載した．サブクエリの実行コストには，PostgreSQL の explain コマンドを用いてサブクエリの推定実行コストを用いる．

4.1 ベンチマーク

本稿では，インターネットムービーデータベース (IMDB) を想定したワークロードである Join-Order-Benchmark (JOB) [12] を使用する．JOB で使用されるデータモデルは，構成するテーブル数が 21 と多い．また，ワークロードにはテーブルの結合処理を含むクエリが多数存在する．よって，整数計画問題によって処理される決定変数が多くなる．決定変数が多いと整数計画問題を解くコストが大きくなるため，本実験を行うワークロードに選択した [1]．但し，本ワークロード内では，実体ビューの容量と実体化ビューの更新コストとの間に相関を持たせるためにクエリ内の最後の集約演算を削除した．[2]

4.2 実験環境

具体的な実験環境を表 1 に示す．ワークロード予測実行コストはワークロードに含まれる全てのクエリの予測実行コストの和のことである．

マシン	MacBook Pro (Retina, Early 2015)
cpu	2.7 GHz dual Intel Core i5
メモリ	8 GB 1867 MHz DDR3
ILP solver	Gurobi9.1[3]
データベース	IMDB
データベースサイズ	7GB
ワークロード	JOB
ワークロード予測実行コスト	200M

表 1　実験環境

4.3 実験結果

ストレージ容量制約の大小によって実体化ビューに選ばれるサブクエリは変化する．提案手法の有効性のストレージ容量制約の変化に応じてどのように変化するか確かめるために，ストレージ容量制約である B_{max} を $0.1GB$，$1GB$，$10GB$ として実験を行った．

1：postgreSQL による cost 予測値あるいは容量予測値が極端に大きいクエリを外し，109 クエリで実験を行った．

2：JOB 内のクエリでは全て最終的にる集約演算により，1 つのクエリによって返される結果は少なく，実体化した時の容量が小さくなる．この場合全てのクエリ全体を実体化すれば大きな利得が得られ、使用容量も少なくなるため，問題が複雑とならない．また，今回は実体化ビューの容量を更新コストとして評価したいため，より複雑なクエリは実体化した際の容量が大きくなるようにしたい．

2：https://www.postgresql.jp/document/11/html/using-explain.html

3：http://www.gurobi.com

4.3.1 実　験　1

この実験は N_ILP により得られる総利得の最適値，BIGSUBS と提案手法により得られる総利得とアルゴリズム実行時間，使用容量を比較するために行った．それぞれの結果を表 2, 3, 4 に示す．

容量制約:0.1GB	実行時間 (s)	合計利得 (M)	使用容量 (%)
N_ILP	485	39.8 (最適値)	99.9
BIGSUBS	1.28	31.5	24.3
最適値との比較 (%)		79.1	
提案手法 ($topk-E$)	3.12	37.1	99.9
最適値との比較 (%)		93.2	
提案手法 ($topk-U$)	2.88	28.0	99.9
最適値との比較 (%)		70.4	
提案手法 ($topk-F$)	5.39	31.9	99.9
最適値との比較 (%)		80.2	

表 2　実験 1 結果 (容量制約:0.1GB)

容量制約:1.0GB	実行時間 (s)	合計利得 (M)	使用容量 (%)
N_ILP	477	43.8 (最適値)	99.9
BIGSUBS	30.49	37.2	20.5
最適値との比較 (%)		84.9	
提案手法 ($topk-E$)	7.36	43.3	99.9
最適値との比較 (%)		98.9	
提案手法 ($topk-U$)	8.02	39.5	99.9
最適値との比較 (%)		90.2	
提案手法 ($topk-F$)	9.45	39.1	99.9
最適値との比較 (%)		89.3	

表 3　実験 1 結果 (容量制約:1.0GB)

容量制約:10GB	実行時間 (s)	合計利得 (M)	使用容量 (%)
N_ILP	523	60.1 (最適値)	99.9
BIGSUBS	10.87	54.8	9.2
最適値との比較 (%)		91.2	
提案手法 ($topk-E$)	8.12	46.4	99.9
最適値との比較 (%)		77.2	
提案手法 ($topk-U$)	6.05	56.1	99.9
最適値との比較 (%)		93.3	
提案手法 ($topk-F$)	7.61	45.0	99.9
最適値との比較 (%)		74.9	

表 4　実験 1 結果 (容量制約:10GB)

BIGSUBS は容量制約を使い切っていないことから，最適値には近似できない．上記に対して提案手法では容量を使い切ることで，どのストレージ制約の場合でも N_ILP により得られる最適値と比べて 75% ～ 97% 程度の精度を達成した．

4.3.2 実　験　2

この実験は提案手法における近傍探索の有効性を確かめるために行った．初期解のみに整数計画問題を適用した総利得と，近傍探索と整数計画問題を繰り返すことにより得られる総利得を比較する．それぞれの結果を表 5，6，7 に示す．

各近傍探索なし $topk$ と比較して大幅に利得を改善していることから提案手法における局所探索法の有用性が示された．初期解群に注目すると，必ずしも同じ種類の $topk$ が一番良い結果を示すとは限らないことも分かった．このため，適切な $top-k$

容量制約:0.1GB	実行時間 (s)	合計利得 (M)	使用容量 (%)
近傍探索なし ($topk-E$)	0.01	25.7	99.9
近傍探索なし ($topk-U$)	0.01	19.2	99.9
近傍探索なし ($topk-F$)	0.01	11.2	99.9
提案手法 ($topk-E$)	3.12	37.1	99.9
提案手法 ($topk-U$)	2.88	28.0	99.9
提案手法 ($topk-F$)	5.39	31.9	99.9

表 5　実験 2 結果 (容量制約:0.1GB)

容量制約:1.0GB	実行時間 (s)	合計利得 (M)	使用容量 (%)
近傍探索なし ($topk-E$)	0.03	25.9	99.9
近傍探索なし ($topk-U$)	0.03	31.2	99.9
近傍探索なし ($topk-F$)	0.03	27.0	99.9
提案手法 ($topk-E$)	7.36	43.3	99.9
提案手法 ($topk-U$)	8.02	39.5	99.9
提案手法 ($topk-F$)	9.45	39.1	99.9

表 6　実験 2 結果 (容量制約:1.0GB)

容量制約:10GB	実行時間 (s)	合計利得 (M)	使用容量 (%)
近傍探索なし ($topk-E$)	0.03	26.0	99.9
近傍探索なし ($topk-U$)	0.03	46.5	99.9
近傍探索なし ($topk-F$)	0.03	37.7	99.9
提案手法 ($topk-E$)	8).12	46.4	99.9
提案手法 ($topk-U$)	6.05	56.1	99.9
提案手法 ($topk-F$)	7.61	45.0	99.9

表 7　実験 2 結果 (容量制約:10GB)

を選択する手法，もしくは初期解群選択手法を選択する必要がある．

5 関 連 研 究

データベース上のクエリ処理を高速化するために適切な実体化ビューを選択する研究が行われており，ヒューリスヒックな手法 [13], [14] [15] と，提案手法と同様に整数計画問題を用いた手法 [6], [7], [8] が提案されている．

Harinarayan ら [13] の研究ではストレージ制約の条件下で貪欲アルゴリズムを用いて実体化ビューを選び．クエリの処理時間を低減している．

T．Dokeroglu ら [14] は分枝限定法，HillClimbinf，Genetic，Hybrid Genetic-Hill Climbing アルゴリズムを含むヒューリスティックな手法を採用している．サブクエリが共通のタスクを共有する機会を増やすために各クエリの代替クエリプランを生成する．このプラン生成器はコストモデルと相互作用し，サブクエリを再利用する価値を導出する．そしてこれらの代替案の中からより低いコストのプランを提供する．

D.C.Zilo ら [15] は類似のビューをマージする，選択条件を変更する，group by 句を追加することにより抽出されたビューを一般化する．これによりストレージの消費を抑えてより多くの実体化ビューを作ることで，多くのクエリがビューを利用する．

BIGSUBS [6] では二部グラフのラベリング問題を用いることで，整数計画問題を細分化し，確率的なアルゴリズムを導入することで整数計画問題で扱う変数を削減する．

Wide-deep [7] は上記の BIGSUBS 改良版とされており，実体化ビュー選択問題の他に，実体化するサブクエリのコストも課題とし，コスト推定モデルを提案している．ビュー選択問題で強化学習を用いることで BIGSUBS の解の正確性の問題を解決している．

BIGSUBS と類似した手法として CloudViews [8] がある．ワークロードに応じてコンパイル時間と実行時間の統計情報を用いて，各サブクエリの利得とコストの正確な値を収集するフィードバックループを実装することで実体化ビューを選択する．周期的なワークロードに焦点を当てているため，異なるパターンや分布を持つ新しいワークロードには迅速に対応できないことが欠点となる．

6 ま と め

提案手法はワークロード内の E_j を考慮して解候補を削減し，整数計画問題を近傍探索を行いながら繰り返し解く．この手法により，最新技術である BIGSUBS と比較して，高速に，高い利得をもたらすことができた．今後の課題として，以下の点が考えられる．

- ワークロードの拡張
- 更新処理の性能を厳密に評価
- 最適実行プラン以外の実行プラン内のサブクエリ

提案手法の主な貢献点は，実体化ビューの品質を保ちつつ，最適化時間を低減することである．最適化時間が長く問題となるような場合に，提案手法は有効である．よって，提案手法は対象とするワークロードが巨大な場合により有効となる．そのために，JOB をさらに拡張し，実験を行う必要があると考えている．

提案手法では，更新性能がストレージ制約によるものでしか考慮されていない．更新処理が頻繁に行われる場合を想定すると，更新処理の性能も厳密に考慮されるべき点となる．更新処理の性能の劣化の評価方法も重要である．利得が高いサブクエリは更新性能を劣化させる程度が大きいことが予想される．そのため，検索性能と更新処理の性能のトレードオフを厳密に評価し，トレードオフの中で最適な解を導き出すために，更新処理の性能も厳密に考慮した整数計画問題の定式化が必要であると考える．

提案手法ではクエリの実行プランは QueryOptimizer から出力される最適な実行プランしか考慮していない．別の実行プランも考慮することでより高い利得が得られると考えられる．しかし，入力するサブクエリの個数が増加すると，整数計画問題に要する時間も増加する．このトレードオフも考慮した実行プランの列挙を行いたい．

7 謝　　辞

この成果は，国立研究開発法人新エネルギー・産業技術総合開発機構（ＮＥＤＯ）の助成事業の結果得られたものです．

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] S.Papadomanolakis, and A.Ailamaki., “An integer linear programming approach to database design,” ICDE, pp.442-449, 2007.

[4] G. G, and M. WJ, “The volcano optimizer generator: extensibility and efficient search. in: Proceedings of the ninth international conference on data engineering,” IEEE, pp.209-218, 1993.

[5] H.Lan, Z.Bao, and Y.Peng, “A survey on advancing the dbms query optimizer: Cardinality estimation, cost model, and plan enumeration,” Data Science and Engineering, pp.86-101, 2021.

[6] A. Jindal, K. Karanasos, S. Rao, and H. Patel, “Selecting subexpressions to materialize at datacenter scale,” PVLDB, pp.800–812, 2018.

[7] H. Yuan, G. Li, L. Feng, J. Sun, and Y. Han, “Automatic view generation with deep learning and reinforcement learning,” ICDE, pp.1501–1512, 2020.

[8] A. Jindal, S. Qiao, H. Patel, Z. Yin, J. Di, M. Bag, M. Friedman, Y. Lin, K. Karanasos, and S. Rao, “Computation reuse in analytics job service at microsoft,” SIGMOD, pp.192–203, 2018.

[9] Y.N.Silva, P.Larson, and J.Zhou, “Exploiting common-subexpressions for cloud query processing,” ICDE, pp.1337–1348, 2012.

[10] J.Zhou, P.Larson, J.C.Freytag, and W. Lehner, “Efficient exploitation of similar subexpressions for query processing,” SIGMOD, pp.533–544, 2007.

[11] 野々部宏司, 柳浦睦憲, “局所探索法とその拡張-タブー探索法を中心として,” 特集 メタヒューリスティクスの新潮流, pp.493–499, 2009.

[12] V. Leis, rey Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann, “How good are query optimizers, really?,” VLDB, pp.204–215, 2015.

[13] V.Harinarayan, A.Rajaraman, and Ullman.J.D., “Implementing data cubes efficiently,” SIGMOD, pp.205–216, 1996.

[14] T. Dokeroglu, M.A. Bayir, , and A. Cosar, “Robust heuristic algorithms for exploiting the common tasks of relational cloud database queries,” Applied Soft Computing, pp.72–82, 2015.

[15] D.C. Zilio, C. Zuzarte, S. Lightstone, W. Ma, G.M. Lohman, R. Cochrane, H. Pirahesh, L.S. Colby, J. Gryz, E. Alton, D. Liang, and G. Valentin, “Recommending materialized views and indexes with ibm db2 design advisor,” ICAC, pp.180–188, 2004.