データベースシステム 117−11
（1999. 1. 23）

知識獲得を用いたデータベース圧縮のための ルール選択方法について

相坂 一樹　塚本 昌彦　春本 要　西尾 章治郎

大阪大学大学院工学研究科情報システム工学専攻

あらまし: 近年のディスク価格の下落にも関わらず，データウェアハウスのような大規模なデータベースにおいては，ディスクにかかるコストはまだ主要なコストを占めている．そのコストを下げる方法として，データベースを圧縮して格納することが有効であると考えられる．それに対しこれまでに筆者らの研究グループでは，圧縮された状態のままデータベースにアクセスできるようなデータベース圧縮方法として，知識獲得を用いてデータ間に潜むルールを抽出し，そのルールをデータと順次置き換えていく圧縮方法を提案している．しかし，提案した手法では，抽出したルールを圧縮に適用する順序により圧縮率に差がでるという問題点があった．このようなルールの適用順序に関しては，単純にすべての組合せを調べることは現実的でないため，本稿では，それほど計算量は高くなく，比較的良い圧縮率が得られるルール選択方法を提案し，実験結果によってその有効性を示す．

On Selection of Rules for Database Compression using KDD

Kazuki AISAKA　Masahiko TSUKAMOTO　Kaname HARUMOTO　Shojiro NISHIO

Department of Information Systems Engineering, Graduate School of Engineering, Osaka University

Abstract: Despite the continuing price drop of memory devices, storage cost is still a major cost factor in large database applications, such as data warehouse. An effective way to reduce the storage cost is to compress a database. Based on this idea, we have proposed compression techniques so far. The proposed method finds hidden rules in a relational database by using knowledge discovery algorithms and replaces data with rules which results in a compression of the database. In this technique, a user can directly access the compressed database. However, there is a problem in this technique such that the application order of the discovered rules largely affects the amount of the resulting database. Since it is not realistic to examine all combinations of the rule application order to find the optimal result, in this paper, we propose a heuristic compression method which does not have high time complexity while can get better compression ratio. Further, we show the effectiveness of the method by experimental studies.

1 はじめに

近年，大規模なデータベースの利用が一般的なレベルまで広がっている．例えば，スーパーマーケットやコンビニエンスストアでは，売れ筋の商品を見つけ出したり，不良在庫を減らしたりするために，POS (Point Of Sales) と呼ばれる販売履歴情報を利用して，効率的な在庫管理を行っている．また，企業においても，意思決定を迅速かつ的確に行うために，データウェアハウスを導入する動きが広がっている．

このように，データベースの重要度がますます増加し，その規模も大きくなってきている．その結果，データベースにおいては，ここ数年ディスク価格が急激に下落しているとはいえ，データの保持にかかるコストがまだ主要なコストを占めている．このようなコストを下げるためには，データベースを圧縮して格納することが有効であると考えられる．

データベースを圧縮することにより，単に必要な記憶容量を減らすだけでなく，いくつかの副次的な効果が期待できる．まず，データベースにアクセスする際，扱うデータ量が減少するため，ディスクI/O

などにかかる時間が減少し，データベース処理時間を減らすことができる可能性がある．また，バックアップデータの量を減らすこともできるので，バックアップ時間を短縮することができる．さらに，ネットワークを介した分散データベース環境においても同じ記憶容量でより多くの複製を作ることができるようになる．データ転送においても，データを圧縮して転送した方が，転送時間が短くなる．これは，文献 [6] で提案されているデータベース移動機構でも有効である．

このようにデータベースを圧縮することは様々な側面において有効であるが，文献 [11] のように，従来の符号化を用いたデータ圧縮技術をそのままデータベース圧縮に適用すると，データベースにアクセスする場合には，圧縮したデータベース全体を展開しなければならない．従って，この方法には蓄積容量が小さくなる代わりに，アクセス速度が低下するという欠点がある．文献 [9] のように，ブロックごとに従来の符号化圧縮技術を適用する方法でも，問合せのために属性ごとのインデックスを作成しなければならず，また，データ更新時にかかるコストも高い．

これに対して，文献 [2, 5, 7, 8] で筆者らは，知識獲得手法 [3] を用いてルールを抽出し，そのルールをデータと置き換えることによって圧縮を行う方法を提案している．この方法を用いれば，圧縮率は従来の方法に比べて低下する場合があるが，圧縮したままデータベースにアクセスできるので，アクセス速度は速くなるものと考えられる．

これらのデータベース圧縮方法においては，多数のルール候補の中からどのような順序でデータベース圧縮に適用していくかによって圧縮率が変化する．従って，圧縮率を向上させるためには，適切なルール適用順序を調べる必要がある．しかし，多数あるルール候補は互いに複雑に関連しており，単純にすべての組合せを調べて最適な圧縮率になるようなルール適用順序を求めようとすると，ルール候補数を N とした時，最悪 $O(N!)$ の計算量がかかってしまう．

そこで，本稿では，それほど計算量は高くなく，比較的良い圧縮率が得られるヒューリスティックなルール選択方法を提案し，実験によってその有効性を確認する．

以下では，まず，2章で知識獲得を用いたデータベース圧縮方法について述べる．次に，3章でルール選択の難しさについて述べ，4章でルール選択方法を提案する．5章では，そのルール選択方法を用いて実験を行った結果を示し，それに対する評価を行う．最後に，6章で結論と今後の課題を示す．

2 知識獲得を用いたデータベース圧縮

本章では，まず 2.1節で知識獲得を用いたデータベース圧縮の概要を述べ，2.2節で，その実行例を示す．

2.1 概要

データベースからの知識獲得を用いたデータベース圧縮方法は，データベースから知識獲得手法を使ってルールを抽出し，その抽出したルールを該当するデータと置き換えることによって，データ量を減らす．

抽出したルールの記憶方法には，演繹データベースに格納する方法 [2, 5] と，関係データベースに格納する方法 [7, 8] の 2 種類の方法がある．

演繹データベースに格納する方法では，抽出したルールを演繹データベースの IDB (intensional database) に格納し，ルールで記述できなかったデータを演繹データベースの EDB (extensional database) に格納する．一方，関係データベースに格納する方法では，通常の表形式でルールを格納しておくと同時に，抽出したルールからビュー定義を作成して，元の表をビューとして復元する．

ルールを演繹データベースに格納している場合は，データベースにアクセスする際，必要な部分だけを演繹すれば良い．また，関係データベースに格納している場合は，ビュー定義によって元の表が復元されているので，ビューに直接アクセスできる．

このように，これらの圧縮方法の利点は，データベースを圧縮しても，圧縮したままデータベースにアクセスできることである．

2.2 実行例

図 1の様に，ID，A，B，C，D の 5 つの属性をもつ表を圧縮することを考える．そのためにまず，知

識獲得アルゴリズムを用いてルールの候補を見つけ出す.

文献[5]では，知識獲得アルゴリズムとして，ID3アルゴリズム[10]を用いる方法と，アプリオリ・アルゴリズム[1]を用いる方法の2つの方法を提案している．ここでは，後者を圧縮に用いた例を示す.

図1の表に対して，しきい値を2としてアプリオリ・アルゴリズムを適用すると図2の様なルール候補が抽出される．例えば，属性Aの要素がaであるタップルは2つ以上あるのでルール候補になる．これを，$(_a__)$と表す.

抽出されたルール候補の中から，例えば$(_acef)$を圧縮に適用するとする．まず，$(_acef)$の要素を含むタップルの中でルールに当てはまらない属性の要素(この場合はIDの要素)を分割表として別の表に格納する(一番目のルールということで分割表1とした)．そして，分割表1のID以外の属性値がa,c,e,fであることを表すルールを作成し，元の表から$(_acef)$をもつタップルを削除する．その結果，図3の様になる.

以下同様にして，$(\_bdef)$,$(__ef)$の順で圧縮に適用したとすると，図4の様に分割表2,3とルールが作成される.

圧縮の結果，圧縮前に50個あった要素数が20個になり，要素数は元の40%になったことになる.

ID	A	B	C	D
1	a	c	e	f
2	a	c	e	f
3	a	c	e	f
4	a	c	e	f
5	b	d	e	f
6	b	d	e	f
7	h	j	e	f
8	i	k	e	f
9	g	l	e	f
10	g	m	n	o

図 1: 圧縮対象となる表の例

$(\_a\_\_)$	$(\_ac\_)$	$(\_\_c\_f)$	$(\_bde\_)$
$(\_b\_\_)$	$(\_a\_e\_)$	$(\_\_de\_)$	$(\_bd\_f)$
$(\_\_c\_)$	$(\_a\_\_f)$	$(\_\_d\_f)$	$(\_b\_ef)$
$(\_\_d\_)$	$(\_bd\_\_)$	$(\_\_\_ef)$	$(\_\_cef)$
$(\_\_e\_)$	$(\_b\_e\_)$	$(\_ace\_)$	$(\_\_def)$
$(\_\_\_f)$	$(\_b\_\_f)$	$(\_ac\_f)$	$(\_acef)$
$(\_g\_\_)$	$(\_\_ce\_)$	$(\_a\_ef)$	$(\_bdef)$

図 2: 図1の表に対するルール候補

ルール: 元の表(X,a,c,e,f) ← 分割表1(X)

元の表

ID	A	B	C	D
5	b	d	e	f
6	b	d	e	f
7	h	j	e	f
8	i	k	e	f
9	g	l	e	f
10	g	m	n	o

分割表1

ID
1
2
3
4

図 3: $(_acef)$をルールにした結果

ルール:
元の表(X,a,c,e,f) ← 分割表1(X)
元の表(X,b,d,e,f) ← 分割表2(X)
元の表(X,Y,Z,e,f) ← 分割表3(X,Y,Z)

元の表

ID	A	B	C	D
10	g	m	n	o

分割表2

ID
5
6

分割表1

ID
1
2
3
4

分割表3

ID	A	B
7	h	j
8	i	k
9	g	l

図 4: 例1の圧縮結果

3 ルール選択の難しさ

2.2節で示した例では，単純に減った要素数を数えていたが，実際はルールを記憶しておくためのデータ量や，分割表のヘッダサイズなどのデータ量が増加する．そのため，ルールになる要素数の多いルール候補から圧縮に適用する方法が，常に一番良いルール選択方法であるとは限らない．以下では，例を用いてルール選択の難しさについて述べる.

まず，ルール候補を圧縮に適用することによって，実際に減るデータ量を減少量と呼ぶ．すなわち，ルールのデータサイズをRD，ルールになるタップル数をRT，ルールを記憶するためのデータサイズをRR，分割表を作成することで増えるデータサイズをDDとした時，次の式で与えられる値を減少量と呼ぶ.

$$RD \times RT - (RR + DD)$$

次に，いくつかのルール選択方法を用いて図1の表を圧縮し，減少量が変化することを示す.

簡単のため，要素一つのデータサイズを1，$RR = RD$，$DD = 3$とする．例えば，$(_acef)$の減少量は$4 \times 4 - (4 + 3) = 9$と計算する．また，$(_g__)$の減少量は$1 \times 2 - (1 + 3) = -2$と負になり，データ量

が逆に増えることになる．そこで，減少量が正になるもののみをルール候補にすることにした．

i ルールになる要素数の多いルール候補から圧縮に適用した場合．

$(\_acef)$,$(\_bdef)$,$(\_\_ef)$ の順に圧縮に適用され，減少量の合計は次のように計算される．

$$\{4\times4-(4+3)\}+\{4\times2-(4+3)\}+\{2\times3-(2+3)\}=11$$

ii 減少量最大のルール候補から圧縮に適用した場合．

$(\_\_ef)$ が圧縮に適用され，減少量の合計は次のように計算される．

$$2\times9-(2+3)=13$$

iii 最適なルール選択をした場合．

$(\_acef)$,$(\_\_ef)$ の順に圧縮に適用され，減少量の合計は次のように計算される．

$$\{4\times4-(4+3)\}+\{2\times5-(2+3)\}=14$$

このように，ルールになる要素数の多いルール候補からルールに適用しても，減少量が一番大きいルール候補からルールに適用しても，最適なルール選択にならない場合がある．

ルール選択を最適化する上で，一番問題になっている点は，多数のルール候補が互いに複雑に関連していることである．このため，あるルール候補を圧縮に適用すると，他の関連するルール候補のルールになるデータ領域が変化する．例えば，$(\_acef)$ を圧縮に適用すると，$(\_\_e\_)$,$(\_\_\_f)$,$(\_\_ef)$ のルールになるタップル数が4減り，$(\_a\_\_)$,$(\_\_c\_)$,$(\_ac\_)$,$(\_a\_e\_)$,$(\_a\_\_f)$,$(\_\_ce\_)$,$(\_\_c\_f)$,$(\_ace\_)$,$(\_ac\_f)$,$(\_a\_ef)$,$(\_\_cef)$ がルール候補ではなくなる．このように，ルール候補を一つ圧縮に適用すると，今までの状態が変化してしまうので，最適なルール選択をするには，変化した後の状態まで考慮しなければならない．しかし，単純にすべての組合せを調べて，最適なルール選択を見つけようとすると，ルール候補数を N とした時，最悪 $O(N!)$ の計算量となり，実用的な時間内で解くのは困難である．

図 5: ルール適用順序の優先度を表すグラフ

4 PO (pair ordering) ルール選択方法

3章で述べたように，現在の状態で一番減少量の多いルール候補を圧縮に適用しても，最適なルール選択にならない場合があった．そこで，本稿では現在の状態だけでなく，その次の状態まで考慮してルール選択を行う PO ルール選択方法を提案する．

まず，ルール候補から2つを取り出すすべての組合せについて，どちらから圧縮に適用した方がよりデータ量が減少するか計算し，圧縮に適用する順序を決定する．

次に，その順序を考慮して，圧縮に適用するルール候補を決定する．例えば，図5左のようにルール候補 A,B,C の中で順序が決定しているとする．A → B は，A が B を制限している状態であり，A を圧縮に適用してから，B を圧縮に適用することを表す．まず，制限されていないルール候補を圧縮に適用する．この場合，A が制限されていないので，圧縮に適用される．A を圧縮に適用すると，B と C はルールになるデータ領域が変化するので，順序付けの計算をもう一度行う．その結果によって，B と C のどちらが先に圧縮に適用されるか決まる．

順序付けの結果によっては，図5右のようにループが発生する場合がある．ループが発生すると，すべてのルール候補が制限されている状態になるので，ルールを選択することができない．このような場合，ループになっているルール候補の中から一つを選んで圧縮に適用すれば，ループの輪が切れて，ルールを選択できるようになる．本稿では，ループの中から要素数最大かつ減少量最大のルール候補を選ぶようにした．

以上より，PO ルール選択方法は次のようになる．

PO ルール選択方法

入力：圧縮する表とルール候補の集合

出力：圧縮した表

1. ルール候補から2つを取り出すすべての組合せの順序を計算する.

2. ルール候補が残っている間, 以下を繰り返す.

   (a) 制限されていないルール候補を圧縮に適用する. 制限されていないルール候補がない場合は, ループの中から要素数最大かつ減少量最大のルール候補を圧縮に適用する.

   (b) 順序を更新する.

このルール選択方法では, 現在とその次の状態までの最適解しか求めていないため, 全体として最適な解が得られる保証はない.

最後に, このルール選択方法の計算量を示す. ルール候補数を N とすると, 順序付けの計算に $O(N^2)$ かかる. 次に, ルール選択では, まず制限されていないルール候補を見つけるのに $O(N)$, 順序の更新に最悪 $O(N^2)$ かかる. さらに, ループがあってルールの選択ができない場合に, ルール候補から一つ選ぶのに最悪 $O(N^2)$ かかる. ルール選択は, 最悪 N 回繰り返されるので, ルール選択全体で最悪 $O(N^3)$ かかることになる. 以上より, PO ルール選択方法の計算量は $O(N^3)$ になる.

5 実験

4章で提案した PO ルール選択方法と, 次の2つのルール選択方法を用いて, 表を圧縮する実験を行った.

- 最大要素数ルール選択方法 (ELEM)
  - 要素数が最大のルール候補から圧縮に適用する. 要素数が同じ場合は, 減少量が最大のルール候補を選択する.
- 最大減少量ルール選択方法 (DEC)
  - 減少量が最大のルール候補から圧縮に適用する. 減少量が同じ場合は, 要素数が最大のルール候補を選択する.

実験には, 一様分布を用いて合成した人事表1と人事表2を使用した. 人事表1は, 従業員ID, 部門, 給料の3つの属性をもつ. 人事表2は, 従業員ID, 部門, 保険のグレード, 給料の4つの属性をもつ.

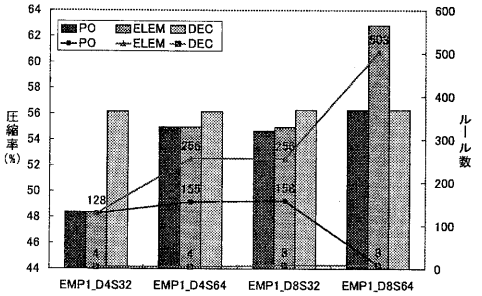

図 6: 人事表1の圧縮結果 (1万行)

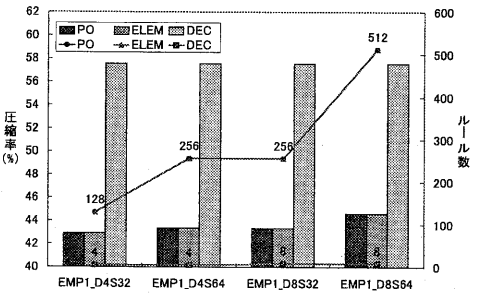

図 7: 人事表1の圧縮結果 (10万行)

まず, パラメータを変えた4種類の人事表1を圧縮した. 例えば, EMP1_D4S32 は部門数を4, 給料のランク数を32にした表である.

1万行の場合の圧縮結果が図6である. 棒グラフが圧縮率, 線グラフがルール数を表す. このグラフから, 異なる属性値の数が増えてもある程度圧縮されていることがわかる. また, PO ルール選択方法が, 常に一番良い圧縮率を記録している.

10万行の場合の圧縮結果を図7に示す. この時, PO ルール選択方法と最大要素数ルール選択方法が, 同じ結果になり, 常に一番良い圧縮率を記録した. また, PO ルール選択方法と最大要素数ルール選択方法は, 1万行の時より良い圧縮率を記録した.

次に, 人事表2を圧縮した. 人事表1の時と同様に, パラメータを変えた6種類の表を用いた. 例えば, EMP2_D4I4S32 は部門数4, 保険のグレード数4, 給料のランク数を32とした表である.

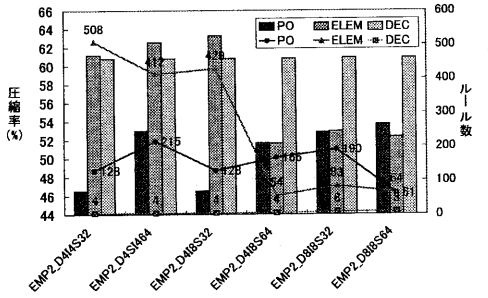

図 8: 人事表 2 の圧縮結果 (1 万行)

表 1: 圧縮時間

	ELEM	DEC	PO
人事表 1 (1 万行)	343 秒	113 秒	313 秒
人事表 1 (10 万行)	3532 秒	1134 秒	4877 秒
人事表 2 (1 万行)	442 秒	159 秒	992 秒

1 万行の人事表 2 を圧縮した結果が図 8 である．この場合も，PO ルール選択方法と最大要素数ルール選択方法が良い圧縮率を記録した．特に，PO ルール選択方法は，一番ではない場合でも一番に近い値を記録している．また，人事表 1 を圧縮した場合とほぼ同じ圧縮率を記録していて，属性数が増えても圧縮できることを示している．

圧縮時間の平均は，表 1 のようになった．タップル数が増えるのに比例して，圧縮時間が増加している．また，属性数が増えても圧縮時間は増加した．特に PO ルール選択方法の増加が著しいが，これはルール候補が増えたためである．

6 おわりに

本稿では，知識獲得を用いたデータベース圧縮方法において，計算量はそれほど高くなく，比較的良い圧縮率が得られるヒューリスティックなルール選択方法を提案した．提案した方法は，ルール候補から 2 つを取り出すすべての組合せについて，順序付けを行うため，現在の状況だけでなく，次の状況も考慮して，ルール選択を行うことができる．この方式では，次の状況までの最適解しか求めていないため，全体として最適な解が得られる保証はないが，実験の結果から，それほど計算量が高くないにも関わらず，比較的良い圧縮率が得られていることを確認した．

今後の課題として，他の知識獲得アルゴリズムと組み合わせて，より圧縮率を高めることなどがあげられる．また，より高速なシステムを構築するために文献 [4] で提案されているアルゴリズムを用いて，知識獲得を圧縮した状態から行うことなどを検討する必要がある．

参考文献

[1] R. Agrawal and R. Srilant: "Fast Algorithms for Mining Association Rules," in *Proc. of the 20th VLDB Conference*, pp.487-499 (1994).

[2] 相坂 一樹, 呉 乾禮, 塚本 昌彦, 西尾 章治郎: "データベースからの知識獲得を用いたデータベース圧縮システム REDUCE1," 情報処理学会 第 55 回全国大会講演論文集 (3), pp.407-408 (Sept. 1997).

[3] U. M. Fayyad, G. Piatetsky-Shapiro and R. Uthurusamy: "Advances in Knowledge Discovery and Data Mining," AAAI Press / The MIT Press (1996).

[4] C. -L. Goh, M. Tsukamoto, and S. Nishio: "Knowledge Discovery in Deductive Databases with Large Deduction Results: The First Step," in *IEEE Trans. on Knowledge and Data Eng., Special Issue on Database Mining*, Vol.8, No.6, pp.952-956 (1996).

[5] C. -L. Goh, K. Aisaka, M. Tsukamoto, K. Harumoto, and S. Nishio: "Database Compression with Data Mining Methods," in *Proc. of the 5th Int. Conf. on Foundations of Data Organization (FODO'98)*, pp.97-106 (Nov. 1998).

[6] T. Hara, K. Harumoto, M. Tsukamoto, and S. Nishio: "Database Migration: A New Architecture for Transaction Processing in Broadband Networks," in *IEEE Trans. on Knowledge and Data Eng.*, Vol.10, No.5, pp.839-854 (1998).

[7] 春本 要, 塚本 昌彦, 西尾 章治郎: "ビューを用いたデータベース圧縮手法," 電子情報通信学会データ工学研究会報告, DE97-13, pp.45–50 (July 1997).

[8] 小西 孝重, 相坂 一樹, 春本 要, 西尾 章治郎: "関係データベースのための圧縮機構と問合せ処理機構の実現," 第 9 回データ工学ワークショップ (DEWS'98) 論文集, pp.46–51 (Mar. 1998).

[9] W. K. Ng and C. V. Ravishankar: "Block-Oriented Compression Techniques for Large Statistical Databases," in *IEEE Trans. on Knowledge and Data Eng.*, Vol.9, No.2, pp.314-328 (1997).

[10] J. R. Quinlan: "Induction of Decision Trees," in *Machine Learning*, Vol.1, No.1, pp. 81-106 (1986).

[11] M. A. Roth and S. J. V. Horn: "Database Compression," in *ACM SIGMOD RECORD*, Vol.22, No.3, pp.31-39 (1993).