一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
DE2018-15 (2018-09)

クエリ集約によるMarkov Logic Networkにおける構造学習の高速化

林　佑樹† 鈴木　順† 荒木　拓也†

† NEC データサイエンス研究所　〒211–8666 神奈川県川崎市中原区下沼部 1753
E-mail: †{y-hayashi@kv, j-suzuki@ax, t-araki@dc}.jp.nec.com

あらまし　Markov Logic Network は事実データとルールを入力とし，不明なデータを論理的に推論することの出来るフレームワークである．推論のためのルールをユーザが作成するのは困難であることから、教師データからルールを自動生成する構造学習が提案されている。構造学習では，ルールとして取り得る組み合わせを網羅的に列挙し，ルールが成り立つかどうかを確かめるためにデータベースの結合処理と同様の処理を行う必要があり，処理負荷が高い．本稿ではデータベースの結合処理を行う複数のクエリを集約することで結合処理回数を減らし，構造学習を高速化する手法について提案する．実験により提案手法では，既存手法に対して最大 17.4 倍学習処理が高速化することが分かった．
キーワード　Markov Logic Network, 論理推論, 構造学習, 関係データベース

Query Aggregation for Acceleration of Structure Learning in Markov Logic Networks

Yuki HAYASHI†, Jun SUZUKI†, and Takuya ARAKI†

† Data Science Research Laboratories, NEC Corporation
1753 Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa 211–8666, Japan
E-mail: †{y-hayashi@kv, j-suzuki@ax, t-araki@dc}.jp.nec.com

Abstract　Markov Logic Network is a framework that can perform logical inference using rules and input data. Since, it is difficult to create the rules for inference from scratch, structure learning which automatically generate the rules from traning data has been proposed. The structure learning algorithm generate many potential candidate rule and calculate likelihood of the rules using the training data. As preprocessing of the likelihood calculation, it is necessary to perform a join operation in the relational database by the number of rules. This is performance bottleneck of the structure learning. Our proposed method aggregates multiple join query into a query and produces a table in which aggregate multiple joined tables to reduce the cost of join operations. Evaluation of the proposed method showed that the performance is better than that conventional methods by up to 17.4 times.
Key words　Markov Logic Network, logical reasoning, structure learning, relational database

1. はじめに

確率論理推論を行うフレームワークとして Markov Logic Network(MLN) [9] が注目されている．MLN では，一階述語論理で記述されたルールを推論モデルとして用いることで，不確実な事象を論理推論することが出来る．「タバコを吸う人はガンになりやすい」などの人の知見に基づく確定的ではないルールを扱うことが出来，ナレッジベース構築システムとしての適用が提案されている [2] [10]．

MLN は，推論を行うために一階述語論理で記述されたルールを作成する必要があるが，観測事象に対して網羅的なルールをユーザが作成するのは困難である．そこで，教師データから一階述語論理のルールを自動的に作成する構造学習 [4] が提案されている．構造学習は，述語を組み合せて作成したルール候補を複数作成し，教師データに対する尤度を計算する処理を繰り返し行うことで，教師データにフィッティングするルール集合を探索する．尤度を計算するためには Grounding 処理と呼ばれる尤度計算を行うための前処理をルール毎に行う必要があり，Grounding 処理は関係データベースにおけるテーブル結合処理に相当する処理が必要となる．構造学習では，ルール候補を複数作成して尤度計算を行う繰り返し処理を行うため，このテーブル結合処理がボトルネックとなり，学習処理のコストが高いという課題がある．

MLN における Grounding 処理は学習時だけでなく推論時に

Ground Pred	Evidence
Smoke(Anna)	1
Smoke(Bob)	

(a) Smoke データ

Ground Pred	Evidence
Cancer(Anna)	
Cancer(Bob)	

(b) Cancer データ

Ground Pred	Evidence
Friend(Anna, Anna)	1
Friend(Anna, Bob)	1
Friend(Bob, Anna)	1
Friend(Bob, Bob)	1

(c) Friend データ

Rule	Weight
Smoke(x) → Cancer(x) : F_1	0.5
Smoke(x) ∧ Friend(x, y) → Smoke(y) : F_2	1

(d) ルール集合

図 1: Markov Logic Network の事実データとルール集合の例

も必要であり，推論時の Grounding 処理を高速化するために Tuffy [8] や ProbKB [2] が提案されている．Tuffy では，ルール毎の Grounding 処理を SQL 命令に置き換えて実行し，巨大なテーブル結合処理を RDBMS の Query Optimizer に最適化させることによって高速化を実現している．しかしながら構造学習では，ルール候補作成処理においてルール候補を多数 (100〜10000) 生成するため，ルール毎に SQL 命令を発行する方式では非効率である．また，ProbKB では，複数のルールに対して SQL 命令を 1 つにまとめて発行する方法が提案されている．しかしながら ProbKB ではルール形状を限定しており，構造学習で用いる場合は任意の形式を持つルールを探索できないという課題がある．

そこで本稿では構造学習に適した Grounding 処理の高速化手法として，クエリ集約手法を提案する．提案方式では，構造学習で生成されるルール候補毎の Grounding 処理の出力として得られるテーブル構造に，同一あるいは包含関係を持つテーブルが多く存在することに着目する．提案方式は同一あるいは包含関係のあるテーブルを結果として得るルールに対する SQL クエリを集約することにより，Grounding 処理の回数を削減することで高速化する．さらに結合前のテーブルのスキーマが同一であるテーブルを集約する方法についても述べる．これにより提案方式であるクエリ集約をさらに効率的に行うことが出来る．本稿ではこのクエリ集約手法を構造学習に適用することにより，構造学習におけるボトルネックである Grounding 処理が高速化されることを示す．

以下本稿では，背景技術となる MLN の概要と構造学習と RDBMS を用いた Grounding 処理を 2 章，提案方式であるルール集約を用いた構造学習を 3 章，実データを用いた実験を 4 章で述べ，最後に第 5 章でまとめを記載する．

2. 背 景 技 術

2.1 Markov Logic Network の概要

Markov Logic Network(MLN) [9] は，一階述語論理で記述されたルールと事実データを用いて確率的論理推論をするためのフレームワークである．本節では，例を用いて Markov Logic Network の概要について説明する．

図 1(a), (b), (c), (d) にタバコと交友関係とガンの因果関係を論理推論するための MLN の例を示す．図 1(a), (b), (c) はユーザが入力する事実データであり，"タバコを吸っている人"や"ガンの人"や"友達"を示すために，Smoke(person)，Cancer(person)，Friend(person, person) を Predicate(述語) として定義する．また，Predicate に実際の定数が入っているもの (Smoke(Anna) 等) を Ground Predicate と呼び，Ground Predicate 毎の真偽値を表にしたものを事実データとする．図 1(d) はユーザが入力するルールであり，"タバコの人はガンである"は"Smoke(x) → Cancer(x)"のように記述される．MLN では，場合によって成立するかどうかが不明なあいまいなルールを用いるために，ルールに Weight を付与することが出来る．Weight 値はそのルールの確からしさを示す相対的な値である．

MLN では，事実データとルールを用いて，真偽値が不明な Ground Predicate に対する真偽値を推論することが出来る．まず，推論を行うための前処理である，ルールの Grounding 処理について説明する．MLN における Grounding とは，ルール集合 $\mathbf{F}$ に含まれるすべてのルール F_n に対して，Ground Predicate の取り得る組み合わせをすべて代入する操作のことを指す．例えば，図 1(d) のルール F_2 に対する Grounding の結果は下記のようになる．

$Smoke(Anna) \wedge Friend(Anna, Anna) \rightarrow Smoke(Anna).$
$Smoke(Anna) \wedge Friend(Anna, Bob) \rightarrow Smoke(Bob).$
…

また MLN では確率推論するために，下記の確率分布関数 [9] が定義されている．

$$P(X = \mathbf{x}) = \frac{1}{Z} exp(\sum_i w_i g_i(\mathbf{x})) \qquad (1)$$

式 (1) における Z は正規化項，$\mathbf{x}$ は Ground Predicate の真偽値を示す確率変数の集合，w_i はルール F_i の Weight 値，$g_i(\mathbf{x})$ は Grounding 処理後のルール F_i が成り立つ場合は"1"で成り立たない場合は"0"を表す素性関数である．この確率分布関数は Grounding 処理後ルールのうち，成り立つルールが多いほど（素性関数が 1 の個数が多いほど）高い値を取る．そのため真偽値が不明な Ground Predicate を推論するためには，この式が最大になるような真偽値を探索すればよい．また，Ground Predicate をすべてが既知である教師データとし，この確率分布に対する尤度が高くなるようなルールを探索することでルールを学習することが出来る．

2.2 構 造 学 習

本節では，本稿において高速化を実現する MLN における構造学習について説明する．MLN では，推論のためにルール集合を用意する必要があり，ユーザが事実データを説明するためのルールを網羅的に作成するのは困難である．そこで MLN におけるルールを自動生成するための学習手法として構造学習がある．構造学習とは，事実データの真偽値がすべて既知である教師データに対して，式 (1) の確率分布関数に対する尤度関数が高くなるようにルールを探索し，ルールセットを作成する．

構造学習の基本アルゴリズムを Algorithm1 に示す．構造学習では，真偽値が確定した Ground Predicate 集合 $\mathbf{x}$ と Predicate

Algorithm 1 構造学習

Input: **x**: a set of ground predicates and evidences
P: a set of predicates
Output: **F**: a set of Rule
1: $\mathbf{F} = NULL$
2: $\mathbf{g} = NULL$ //a set of grounding table
3: $score = 0$
4: **repeat**
5: **candlist** $= CreateCandidate(\mathbf{P})$
6: **for** $cand$ in **candlist do**
7: **new_g** $= \mathbf{g} + Grounding(cand, \mathbf{x})$
8: $newscore = Learning(\mathbf{new\_g})$
9: **if** $newscore - score > 0$ **then**
10: $score = newscore$
11: $BestCand = cand$
12: **g** = **new_g**
13: $\mathbf{F} = \mathbf{F} + BestCand$
14: **until** no update of the score.
15: **return F**

集合 **P** を入力として，ルール集合 **F** を求める．アルゴリズムは，ルール候補の作成処理，Grounding 処理，Learning 処理を繰り返し実行し，Learning 処理の結果として得られるスコア（尤度）を向上させるルール候補が無くなると終了する．まず，ルール候補の作成処理（5 行目）では，Predicate 集合 **P** を組み合わせてルール候補の集合（**candlist**）を作成する．ルール候補となる Predicate の組み合わせは Predicate の持つ変数の組み合わせを考慮して全通り列挙した場合，組み合わせが膨大な数となる．そこで組み合わせの数を枝狩りするいくつかのヒューリスティック手法が提案されている [4] [7] [3]．本稿では最も基本的なルール候補生成アルゴリズムである BeamSearch [4] を用いる．次に，生成したルール候補の集合（**candlist**）に対して，Grounding 処理と Learning 処理を繰り返し実行する（6〜8 行目）．Grounding 処理では，繰り返し内のルール候補（$cand$）に対する Grounding 処理を実行する（7 行目）．この Grounding 処理結果と，前回以前の繰り返し内で確定したルールに対する Grounding 処理結果を合わせた **new_g** を用いて，Learning 処理を実行する（8 行目）．Learning 処理では文献 [4] に記載の対数尤度関数を用いてスコアを計算する．すべてのルール候補の計算が終了した後，最もスコアが高いルール候補 $BestCand$ をルールとして採用する（13 行目）．これらの処理を繰り返し実行し，繰り返し処理の中で尤度を更新するルール候補が発見できなくなると処理を終了する．

構造学習では，ルール候補の数だけ Grounding 処理と尤度計算処理を行う必要があり，Predicate の組み合わせを全探索しない場合でも数 100〜数 10000 程度のルール候補を探索する [4] [7]．特に Grounding 処理は，ルール候補に対して Ground Predicate の取り得る組み合わせをすべて代入する操作を行う必要があり，計算コストが高いことが知られている [8]．

2.3 RDBMS を用いた Grounding 処理

全節で説明したとおり，Grounding 処理は MLN の処理ボトルネックの一つであることが知られている．この処理は，Ground Predicate をテーブル表現した時，テーブルの結合処理として SQL 命令で記述することが出来る．この性質を用いて，Grounding 処理を SQL 命令で記述し，RDBMS を用いて高速化する手法が提案されている [8]．Grounding 処理を既存の SQL 関数に置き換えて実行することで，RDBMS 内の Query Optimizer によりクエリが最適化されるため，ループ処理によって Grounding 処理を行う手法に対して容易に高速化することが可能である．

Grounding 処理を SQL 命令に置き換える場合のクエリについて説明する．まず，図 1(a), (b), (c) に例で示した Ground Predicate を RDB におけるテーブルとして用意する．テーブルはスキーマとして，{変数の種類, Evidence の真偽値}を持つ．変数の種類とは，Predicate の変数の種類であり，Evidence の真偽値とは"0"または"1"の bool 値となる．たとえば Friend(person, person) であれば，{person, person, evidence}のようなスキーマを持つ．ここで，下記のようなルールに対する Grounding を考える．

$$Friend(x, y) \wedge \ Friend(y, z) \rightarrow Friend(x, z).$$

このルールに対する代入処理を先ほど作成した Predicate テーブルを用いて SQL 命令で記述すると下記のようなクエリで実現できる．

```
SELECT  f1.person1, f1.person2, f1.evidence, ...
FROM    Friend as f1, Friend as f2, Friend as f3,
WHERE   (f1.person1 = f3.person1)
        AND (f1.person2 = f2.person1)
        AND (f2.person2 = f3.person2);
```

本クエリはルール内の変数 x, y, z を，Friend テーブルの結合条件 (WHERE 句) とした SQL 命令である．このクエリから得られる各行がそれぞれ Grounding 処理後のルールとして得られる．このようなクエリをルール毎に生成することで，Grounding 処理を SQL 命令として記述でき，テーブル結合処理の最適化を RDBMS の Query Optimizer により自動化することが出来る．

3. クエリ集約による構造学習の高速化

3.1 提案方式概要

2.3 節において，RDBMS を用いた Grouding 処理について説明した．文献 [8] においては，本手法を MLN における推論処理に適用したが，構造学習においても Grounding 処理は同様に行うため，本方式による高速化効果が見込める．しかしながら構造学習においては，生成されるルール候補毎に毎回 Grounding 処理のためのテーブル結合クエリを発行するため，非効率である．

そこで，Grounding 処理結果として得られるテーブルが同様あるいは包含関係を持つルールを選択し，クエリを集約実行する方法について考える．構造学習では，Predicate 内の変数の組み合わせを考慮してルール候補を生成するため，下記のような構造を持つルール候補が多数生成される．

$$\begin{aligned} F_1 &: Friend(x, y) \rightarrow Friend(y, x). \\ F_2 &: Friend(x, y) \rightarrow !Friend(y, x). \\ F_3 &: Friend(x, y) \rightarrow Friend(y, z). \end{aligned} \tag{2}$$

これらのルールは否定を表す"!"の有無と，Predicate の変数の組み合わせが異なるルールである．これらのルールに対する

Algorithm 2 クエリ集約を用いた構造学習

Input: **x**: a set of ground predicates and evidences
P: a set of predicates
Output: **F**: a set of Rule

1: $\mathbf{F} = NULL$
2: $\mathbf{g} = NULL$ //a set of grounding table
3: $score = 0$
4: **repeat**
5: **candlist** $= CreateCandidate(\mathbf{P})$
6: **parentlist** $= Aggregation($**candlist**$)$
7: **parent_g** $= NULL$
8: **for** *parent* in **parentlist do**
9: **parent_g** $=$ **parent_g** $+ Grounding(parent, \mathbf{x})$
10: **for** *cand* in **candlist do**
11: **new_g** $=$ **g** $+ SelectGrounding($**parent_g**$, cand)$
12: $newscore = Learning($**new_g**$)$
13: **if** $newscore - score > 0$ **then**
14: $score = newscore$
15: $BestCand = cand$
16: **g** $=$ **new_g**
17: **F** $=$ **F** $+ BestCand$
18: **until** no update of the score.
19: **return F**

Grounding 結果を図 2(a), (b), (c) に示す．図 2 のテーブルはそれぞれ，{person, person, evidence}のスキーマを持つ Friend に対応するテーブルを，2.3 節で説明した SQL 命令によってテーブル結合した結果である．図 2 を参照すると，F_1 及び F_2 のテーブルは，F_3 のテーブルのレコードの部分集合となっていることから，F_3 に対する Grounding 処理のみを行うことで，F_1 及び F_2 の Grounding 処理結果を得ることが出来る．この性質を用いることで構造学習におけるルール探索において候補となるルールの Grounding 処理に必要なテーブル結合の回数を削減できる可能性がある．

そこで本稿では，ルール集約を用いた構造学習の高速化を提案する．構造学習では，式 (2) のような Grounding 処理結果のテーブルに包含関係のあるルールが，ルール候補として多数生成される．そこで提案方式では構造学習において生成されたルール候補に対して，包含関係を持つ複数のルールを集約して Grounding 処理を実行することで，Grounding 処理に必要なテーブル結合回数を削減する．

提案方式のアルゴリズムを Algorithm2 に示す．Algorithm1 では，ルール候補の集合が生成された後，ルール候補毎に Grounding 処理と Learning 処理を行っていた．それに対して Algorithm2 では，ルール候補の集合を生成した後，ルール候補集合を参照して Grounding 処理結果に包含関係のあるものをグループ化する（6 行目）．このとき，グループ化したルールの内，他のルールの Grounding 処理結果を包含するルールの集合を **parentlist** とする．次に，他のルールの Grunding 処理結果を包含するルール *parent* 毎に Grounding 処理を実施する（8-9 行目）．この時，Grounding 処理を行わなかったルール，すなわち Grounding 処理結果が他のルールの Grounding 処理結果の部分集合となるルールについては，*parent* 毎の Grounding 処理結果から抽出できるため，Grounding 処理結果の集合 **parent_g** からレコードを抽出することで Grounding 結果を得る（11 行目）．このように，すべてのルール候補に対して Grounding 処理を行うのではなく，Grounding 処理結果に包含関係のあるクエリを集約して Grounding 処理を実行することでテーブル結合回数を削減し，構造学習のボトルネックである Grounding 処理を高速化する．

Friend(x, y)→Friend(y, x)

x	y	evi	y	x	evi
A	A	1	A	A	1
A	B	0	B	A	0
B	A	0	A	B	0
B	B	1	B	B	1

(a)

Friend(x, y)→!Friend(y, x)

x	y	evi	y	x	evi
A	A	1	A	A	0
A	B	0	B	A	1
B	A	0	A	B	1
B	B	1	B	B	0

(b)

Friend(x, y) → Friend(y, z)

x	y	evi	y	z	evi
A	A	1	A	A	1
A	A	1	A	B	0
A	B	0	B	A	0
A	B	0	B	B	1
B	A	0	A	A	1
B	A	0	A	B	0
B	B	1	B	A	0
B	B	1	B	B	1

(c)

図 2: クエリ集約可能な結合後テーブル

Friend(p1, p2)

p1	p2	evi
A	A	1
A	B	0
B	A	0
B	B	1

(a)

Parent(p1, p2)

p1	p2	evi
A	A	0
A	B	0
B	A	1
B	B	0

(b)

Agg(p1, p2)

p1	p2	evi
A	A	10
A	B	00
B	A	01
B	B	10

(c)

図 3: テーブル集約可能な事実データ

3.2 クエリ集約条件

本節では，3.1 節で説明したクエリ集約の条件について説明する．ルール i に対する Grounding 処理のためのテーブル結合クエリに関して，結合する対象となるテーブルの集合を $\mathbf{P_i}$ とし，結合条件の集合を $\mathbf{C_i}$ とした時，クエリ集約の条件は以下で定義される．

クエリ集約条件 1： ルール i と j に関して，$\mathbf{C_i} = \mathbf{C_j}$ かつ $\mathbf{P_i} = \mathbf{P_j}$ の時，ルール i は j の Grounding 結果を包含する．
クエリ集約条件 2： ルール i と j に関して，$\mathbf{C_i} \subset \mathbf{C_j}$ かつ $\mathbf{P_i} \supset \mathbf{P_j}$ の時，ルール i は j の Grounding 結果を包含する．

P_i と C_i は，2.3 節で説明した Grounding 処理の SQL クエリのうち，P_i は FROM 句で使用される結合前のテーブルを示し，C_i は WHERE 句内の AND で指定されたテーブルの結合条件に相当する．クエリ集約条件 1 はクエリに使用するテーブル種別が同じで，テーブル結合条件が同じであることを示しており，図 2(a)，(b) のように否定を表す"!"の有無のみが異なるルールのクエリを集約する条件となる．クエリ集約条件 2 はクエリに使用するテーブル種別とテーブル結合条件が部分集合の関係となっていることを示しており，図 2(a)，(c) のように Groundin 処理結果のテーブルが他のテーブルの部分集合となるルールのクリを集約する条件となる．

3.3 テーブル集約によるクエリ集約の拡張

本節では，3.1，3.2 節で説明したクエリ集約による Grounding 処理の高速化の効果を拡張するためのテーブル集約手法について説明する．クエリ集約では，Grounding 処理を行うルールのうち，Grounding 処理のためのクエリを参照して，クエリ内で使用するテーブルとテーブル結合条件が部分集合になっているものを集約する．ここで，下記のような 2 つのルールを考える．

$$F_1 : Friend(x, y) \rightarrow Friend(y, x).$$
$$F_4 : Parent(x, y) \rightarrow Parent(y, x).$$

表 1: 実験に用いたデータセット

	Predicate 種類	Ground Predicate 数
smoke	5	80600
webkb	6	54990
imdb	10	70686
cora	10	94632
uwcse	15	69261

これらはルール内の Predicate の種類が異なるのみのルールである．ここで Friend 述語及び Parent 述語は共に同一の変数を持つため，両者のテーブルスキーマは同様で{person, person, evidence}となる．このように同一のスキーマを持つ述語を用いる場合，F_1 や F_4 のルールの Grounding 処理のためのテーブル結合は異なるテーブルに対して同様の処理を行うこととなる．そのため，同一スキーマを持つテーブルを 1 つのテーブルとして扱うことにより，F_1 と F_4 も 3.2 節に説明したクエリ集約条件によってクエリを集約することが出来る．

図 3(a), (b), (c) にテーブル集約の例を示す．図 3(a), (b) のように同一スキーマを持つテーブル同士を集約し，Aggregation テーブルとして (c) を生成する．ここで，evidence 列については，集約前のテーブルは bool 値が入っているため，これを集約し bit 列として集約前の bool 値を保持する．このように同一スキーマを持つテーブルを集約し，1 つのテーブルとして扱うことによって，3.2 節で説明したクエリ集約条件に当てはまるルールを拡張し，クエリ集約の効果を増加させることが出来る．

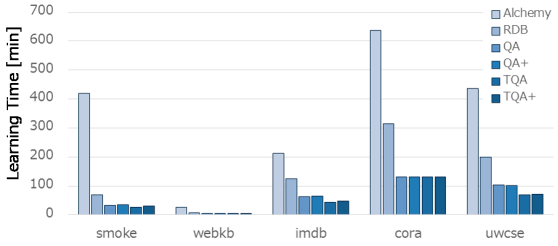
4. 評 価 実 験

本章では提案方式の有効性を示すために，構造学習の Grounding 処理を For 構文によるネステッドループで行う従来方式 (Alchemy [5]) と，Grounding 処理を RDBMS によって高速化した方式（RDB）と，提案方式を比較する．提案方式は，クエリ集約条件 1 のみを用いた方式（QA）と，クエリ集約条件 1 と 2 を用いた方式（QA+）と，クエリ集約条件 1 に加えテーブル集約を行った方式（TQA）と，クエリ集約条件 1 と 2 に加えテーブル集約を行った方式（TQA+）で比較実験を行う．

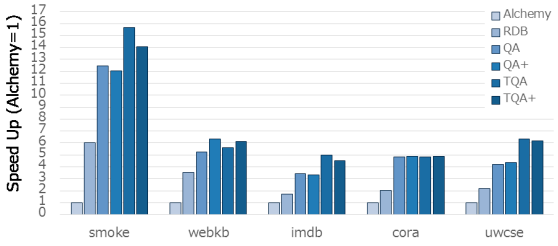
4.1 実 験 環 境

実験にはプロセッサが Intel Xeon CPU E5-1650 (6 コア，3.20GHz)，メモリが 16GB，OS が Linux 3.10.0 の計算機を用いた．また，Alchemy はオープンソースとして公開されている C++ 言語で実装されたもの [1] を用いる．RDBMS と RDBMS+Agg は Java 言語で実装を行い，データベースシステムとして PostgreSQL 9.6.5 を使用した．構造学習のパラメータとして，生成されるルール候補の最大 Predicate 数を 3 とした．その他は Alchemy に実装されている構造学習のデフォルトパラメータとした．

実験では，5 種類のデータセット（smoke，webkb，imdb，cora，uwcse）を使用して評価を行った．それぞれのデータセットの Predicate 数と Ground Predicate 数を表 2 に示す．smoke は基本評価のために我々が教師データセットを作成したものである．Cancer(person)，Smoke(person)，Drink(person)，Friend(person, person)，Parent(person, person) の 5 種類の Predicate を持ち，Cancer を全 person の半数生成し，Cancer である person は Smoke と Drink が真になりやすく，Smoke と Drink が真になっている person 同士は Friend と Parent が真になりやすいように教師データを作成した．webkb, imdb, cora, uwcse は，[1] で公開されている実データに基づくデータセットであり，文献 [4] [7] を参照して実験用に教師データを整理して使用した．

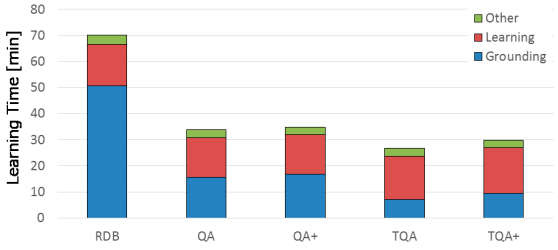


(a) 処理時間

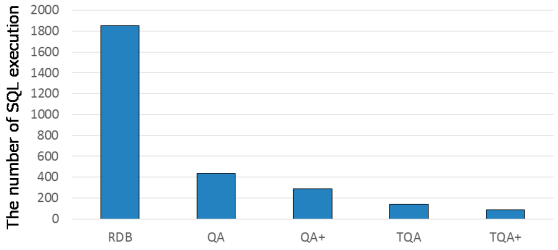


(b) 高速化比率

図 4: 構造学習の処理時間



(a) 処理時間内訳



(b) クエリ回数

図 5: Smoke データセットの処理時間内訳

4.2 ルール集約による構造学習の高速化

まず，提案方式の有効性を示すため，4.1 節で説明した 5 種類のデータセットを使用して構造学習の実行時間について評価を実施した．

図 4(a)，(b) に，構造学習の実行時間と Alchemy の実行時間を 1 とした場合の高速化比率を示す．いずれのデータセット

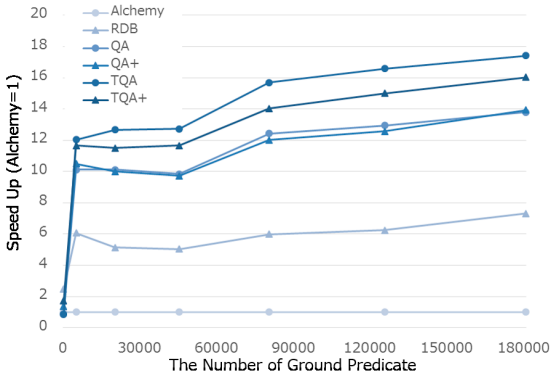


図 6: データサイズに対する計算時間

においても，提案方式のほうが Alchemy に対して 3.3-15.7 倍，RDB に対して 1.5-2.8 倍程度高速化されていることが分かった．提案方式同士の比較では，クエリ集約のみの場合（QA/QA+）よりもテーブル集約を加えた場合（TQA/TQA+）のほうが平均して 1.2 倍程度高速化されていることが分かった．一方，webkb 以外のデータセットでは，同じテーブル結合クエリを集約するクエリ集約条件 1 のみの場合（QA/TQA）に比べ，包含関係を考慮してクエリを集約するクエリ集約条件 1 ＋ 2（QA+/TQA+）のほうが性能が劣化することとなった．

この性能劣化要因を分析するために，図 5(a)，(b) に Smoke データセットに対する方式毎の処理時間内訳と，Grounding 処理におけるテーブル結合クエリ実行回数を示す．まず，図 5(a) を参照すると，それぞれの方式に関して提案方式の想定通り Grounding 処理時間が高速化されることで，構造学習処理時間が改善していることが分かる．また，クエリ集約条件 1 のみの場合に比べ，クエリ集約条件 2 を加えた場合のほうが Grounding 処理時間が若干延びていることが分かった．次に，図 5(b) を参照すると，クエリ集約条件 1 のみの場合に比べ，クエリ集約条件 2 を加えた場合に回数が削減されていることが分かる．ただし，クエリ集約しない場合に比べ，クエリ集約条件 1 のみの削減効果に対して，クエリ集約条件 2 を加えた場合の削減効果は小さい．クエリ集約条件 2 では結合クエリ間の包含関係を参照じてクエリ集約を行うが，テーブルの包含関係を考慮して集約する場合，包含される側のテーブルは比較的小規模なテーブルとなり，テーブル結合のコストが小さくなる．一方で，クエリ集約を行うと，得られたテーブルの部分集合を Learning 前に選択的に取り出す操作が必要となるため，集約されたテーブルの大きさに差がある場合、このコストが相対的に大きくなる．すなわち，データセットの種類や構造学習内で生成するルール候補の種類によっては，Grounding 処理結果が同一となるルールのクエリを集約することで十分な高速化効果が得られるということが分かった．

4.3 データサイズに対する評価

次に，データサイズを変更した場合の高速化効果について比較するために，smoke データセットの教師データの Ground Predicate 数を 230〜180900 で変化させて評価を実施した．図 6 にデータサイズに対する構造学習の計算時間を示す．本結果から，データサイズを変化させた場合においても，提案方式は同程度の割合で実行時間が高速化されており，最大データサイズにおいては Alchemy に対して 17.4 倍，RDB に対して 2.6 倍高速化されていることが分かった．本結果より，データサイズが大きい場合においても，提案方式によるクエリ集約の効果が得られることが確認できた．

5. ま と め

本稿では，MLN における構造学習内の Grounding 処理において，繰り返し実行するテーブル結合処理のクエリを集約することで結合処理回数を減らし，構造学習を高速化する手法について提案した．また，クエリ集約手法を MLN における構造学習へ実装する方法について述べ，評価を実施した．実験により，提案方式ではクエリ集約によるテーブル結合処理の回数を削減することで，構造学習の処理時間を最大 17.4 倍高速化できることを示した．

文　献

[1] Alchemy: Open source ai. `https://alchemy.cs.washington.edu/`.

[2] Yang Chen and Daisy Zhe Wang. Knowledge expansion over probabilistic knowledge bases. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 649–660. ACM, 2014.

[3] Jesse Davis and Pedro Domingos. Bottom-up learning of markov network structure. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 271–280. Omnipress, 2010.

[4] Stanley Kok and Pedro Domingos. Learning the structure of markov logic networks. In *Proceedings of the 22nd international conference on Machine learning*, pp. 441–448. ACM, 2005.

[5] Stanley Kok, Marc Sumner, Matthew Richardson, Parag Singla, Hoifung Poon, Daniel Lowd, Jue Wang, and Pedro Domingos. The alchemy system for statistical relational {AI}, 2009.

[6] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, Vol. 45, No. 1-3, pp. 503–528, 1989.

[7] Lilyana Mihalkova and Raymond J Mooney. Bottom-up learning of markov logic network structure. In *Proceedings of the 24th international conference on Machine learning*, pp. 625–632. ACM, 2007.

[8] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *Proceedings of the VLDB Endowment*, Vol. 4, No. 6, pp. 373–384, 2011.

[9] M Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, Vol. 62, No. 1-2, pp. 107–136, 2006.

[10] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental knowledge base construction using deepdive. *Proceedings of the VLDB Endowment*, Vol. 8, No. 11, pp. 1310–1321, 2015.