データ結合候補の削減による Programming by Example を用いたデータ整形

宇治橋　善史†　　野間　唯†

†株式会社富士通研究所 〒211-8588 川崎市上小田中 4-1-1

E-mail:　†{ujibashi, noma.yui}@jp.fujitsu.com

概要　近年、様々なデータソースのデータを組み合わせたビッグデータ分析による新たな知見創出や、ビジネス活用が注目されている。多種多様なデータを分析し知見創出や活用をするには、これらデータをプロファイリングしてデータを理解したり、データ整形を行ったりするデータプレパレーション作業が必須である。しかしながら、特にデータ整形作業は、プログラミングスキルや多大な作業工数を必要とし、データ分析の高い障壁となっていた。そこで、本論文では、自動プログラム生成技術の Programming by Example (PBE) を導入してデータ整形をプログラミングレスにするアイデアを紹介し、さらにデータ整形に必要な変換処理種別に PBE を拡充する際に必要なプログラム探索手法を提案する。既存 PBE を拡張しデータ整形に必須の意味的変換処理へ対応させるためには、テーブル結合候補の組み合わせが膨大になり、処理が現実的な時間で完了しない問題があった。そこで、本論文では、テーブル結合候補組み合わせ増大を抑止させるために、テーブル結合候補の削減手法を提案する。評価の結果、結合処理を含む変換処理を現実時間で完了することを確認した。この技術により、データ整形のプログラミングレスが実現してデータ分析の障壁が下がることで、これまで利用が難しかったデータ資産を活用した高度な分析や新たなビジネス創出が可能になる。

キーワード　データプレパレーション、データ整形、Programming by Example、グラフ探索、A*アルゴリズム

1. はじめに

膨大に増え続ける多種多様なデータを集めて高度な分析をすることによって新たな知見を発見したり、新規ビジネス立ち上げしたりして、データ利活用することが近年増々重要視されるようになってきた。実際のデータ分析プロセスでは、統計分析や機械学習などのデータ分析そのものの前に、収集したデータを理解し、分析のためのデータを用意するデータ加工の作業があり、これをデータプレパレーション作業と呼ぶ。このデータプレパレーション作業は、データ分析プロセス全体工数のなかで非常に大きい作業時間占めている。あるレポート[1] ではデータ分析プロセスの約 8 割がこのデータプレパレーション作業に費やされるという報告がある。

図 1 に示したように、データプレパレーション作業では、データ理解、データ整形、レビューの作業を繰り返し行う。データ理解作業では、収集したデータの形式がどうなっているか、どのような属性や値が入っているか等データの素性を把握するためのデータ理解のフェーズがある。次のデータ整形作業では分析用データを作成するためにどのように加工を行うかを設計し、その加工処理を行うために多くの場合は SQL 言語や Python などのスクリプト言語によるコーディングを行う。次にレビュー作業を行い、整形したデータが欲しいデータになっているかどうかを調べ、もし、想定外のデータやコーディングバグによって、意図した

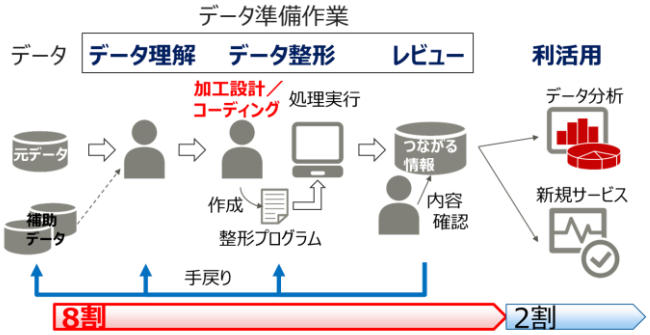

図 1 データ利活用のためのデータ分析プロセス

データになっていなかったら、またデータ理解、データ整形のフェーズに戻って作業を行う必要がある。このようにデータプレパレーション作業はコーディングのような複雑な作業を行うだけでなく、レビューのフィードバックを受けて繰り返し実施する必要がある。

また、近年のデータ分析の傾向として、高度なビジネス知識をもってデータ分析することで新規ビジネス創出や新しい視点に立ったマーケット分析をおこなったり、医療データなど専門的なデータを使って分析したりすることが行われている。そのため、ビジネスパーソンや科学者といった各領域の専門家（以降、ドメイン専門家と記述する）がデータ分析に積極的にかかわる必要があるが、前述のようにコーディングのような IT 知識と工数が必要な作業が分析作業を進めるう

えで非常に大きな障壁となり、分析作業が遅延したり、分析自体を諦めたりする大きな原因となっている。

これらを鑑みて、データ利活用を円滑化するためには、データプレパレーション作業を、IT 専門家が高速に実施できるように効率化したり、IT 知識が豊富でないドメイン専門家でも容易に実施できるように簡単化したりすることが重要である。現在、データプレパレーションと呼ばれる分野に於いてこれらの課題の解決が技術的な目標となっている。

2. Programming by Example によるデータ整形

2.1. Programming by Example

データプレパレーション作業を簡単化、効率化する技術としては、Data Wrangler (現在の Trifacta)[2]、OpenRefine[3]等のツールが提案されており、商用製品あるいはフリーのソフトウェアとなっている。これらのツールは Excel のような GUI ツールで、スプレッド形式でデータを閲覧可能になっている。また、属性毎の値のヒストグラム表示機能や変換処理のサジェスチョン機能等データプレパレーション作業に特化した機能を GUI として備えることで、データプレパレーション作業効率化を実現している。

我々は、前述の[2],[3]のような GUI 機能を豊富にする方向ではなく、ユーザがデータ変換の一部の例を与えることでデータ整形を自動化する技術の適用を紹介する。データ整形はデータプレパレーション作業のなかでもコーディング等の IT 知識が必要で最も工数がかかっていると考えられるため、データ整形の自動化によってデータプレパレーション作業が大幅に効率化・簡単化されると考えている。その技術の実現イメージは　図 2 のようなものである。

図 2 データ整形の自動化

まず、ユーザは整形前のデータのなかの数レコードのサンプルに対して、整形の結果欲しいレコードの内容を入力する。システムは、この整形前の例と整形後の例を頼りに、自動的に変換の組み合わせを推定し、変換プログラムを生成する。図の例では、日時の時刻を削除して日付のみに変換したり、システムで null が許可されていないために入力された乗車駅の’0000’といった値を null に変換したり、整形前の例にはない性別、年齢の属性を ID をキーに他のテーブルから結合したりする変換プログラムを、2 行の例から推定する。最後に、生成されたプログラムを整形前のサンプル以外のレコードにも適用することで、全整形後のデータを取得する。整形後のデータをレビューし、例が不足しているために整形が失敗しているようであれば、整形前の例を増やし、対応する整形後の例を入力して再度実行することができる。これらを繰り返すことで、最終的に正しい整形後のデータを取得する。

この技術を用いると、ユーザが与えた例を頼りに自動的にデータ整形することで、IT 専門家は加工設計・コーディングが不要になり大幅に工数が削減される。また、プログラミングが得意ではないドメイン専門家にとっても、データ整形が容易になりデータ分析の障壁を取り除くことができる。

このようにユーザが与えた入出力を満たすプログラムを自動生成する技術は、プログラム合成の一分野として Programming by Example (PBE)と呼ばれて以前から研究されてきた分野であるが、適用分野は限られていた[4]。しかし、近年になって、データ整形の分野で適用範囲を限って実現している研究がある。FlashFill[5]は文字列抽出に特化した PBE を実現している。Foofah[6]はテーブル縦横変換を PBE で可能にするなどレイアウト変換を得意とする。他にも[7]のように複数のテーブル情報を結合する PBE 技術がある。このようにデータ整形のなかでも個別の分野については PBE をデータ整形に活用した技術が存在する。

2.2. データ整形の変換の分類

データ整形作業には、様々な変換処理が必要である。これらのデータ変換を大きく 3 種類に分類すると、構造的変換 (Layout Transformation)、構文的変換 (Syntactic Transformation)、意味的変換 (Semantic Transformation)に分けることができる。各変換について以下に説明する。

構造的変換

構造的変換はテーブルのレイアウト変更を伴う変換操作である。例えば、カラムの削除、移動や、Excel のピボットテーブルのようにテーブルの縦横を変更したりする変換を指す。

構文的変換

構文的変換は文字列操作を伴う変換操作である。例えば、正規表現により部分文字列を抽出したり、文字列を複数の文字列に分割したり、文字列を結合して文字列生成したりする変換を指す。構文的変換は変換前の文字列及び部分文字列のみを使って変換後文字列を生成可能な変換である。

意味的変換

意味的変換は、変換後の文字列を生成するために

変換前の文字列以外の情報が必要となる変換である。この情報は、辞書から引っ張ってきたり、外部テーブルからある結合条件を満たすデータを引っ張ってきたりすることで取得する。

現状の既存技術では表 1 のように、[5]では構文的変換、[6]では構造的変換と構文的変換、[7]では意味的変換に対応しているが、全ての変換分類に対応している技術はない。そこで、本論文では、現実のデータ整形に必須の変換処理を包括的にサポートするために、構造的、構文的、意味的変換　全てに対応可能な PBE を実現する手法を提案する。

表 1　各技術の変換分類対応表

	構造的	構文的	意味的
文献[5]		✓	
文献[6]	✓	✓	
文献[7]			✓
本論文目標	✓	✓	✓

2.3. PBE による変換分類の包括的対応

PBE の実装は、入出力例条件を満たす可能な変換を網羅したプログラム空間を探索する問題に帰着する。プログラム空間を探索する手法として例えば[5],[7]では、Version space algebra を採用し、[6] では A*アルゴリズムによる有向グラフ探索手法を採用している。

我々は、構造的変換と構文的変換を実現している[6]と同様に A*アルゴリズムを採用することにした。[6]が対応していない意味的な変換処理を追加することを目標とする。この手法を採用する理由は、汎用性が高いグラフ探索アルゴリズムを利用しているので、新しい変換処理を追加することが比較的容易であり、様々な変換処理が必要な現実の問題を解くためにマッチするからである。

この手法を具体的に説明する。与えられた入出力に従って変換プログラムを自動生成する処理は、有向グラフの探索問題に帰着することが可能である。有向グラフは PBE が表現可能な全プログラム空間を表しており、その一部を図示したのが図 3 である。

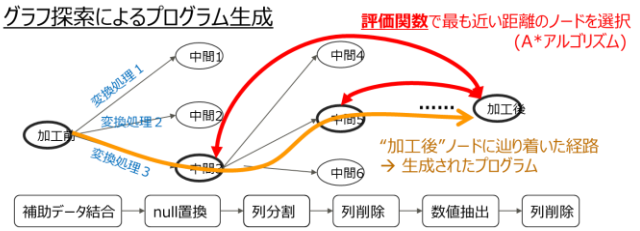

図 3 グラフ探索による変換ルール自動化

グラフの各ノードはテーブルの変換前のテーブル状態、変換の中間テーブル状態、変換後のテーブル状態を表す。また、ノードとノードを結ぶエッジはノードからノードへ状態を遷移させるための変換単位（オペレータ)を表す。変換プログラムを求めるためには、変換前ノードと変換後ノードを結ぶ最短経路を探索する問題を解けばよい。

A*アルゴリズムはこの最短経路探索のための効率化手法である。A*アルゴリズムでは、グラフ探索途中の中間ノードを n、開始ノードの推定最短距離 g(n)、終了ノードの推定距離 h(n)とすると、f(n)=g(n)+h(n)が最小となるノード n を探索経路として選択する(図 4)。これにより、ヒューリスティックに最短経路を探索する。

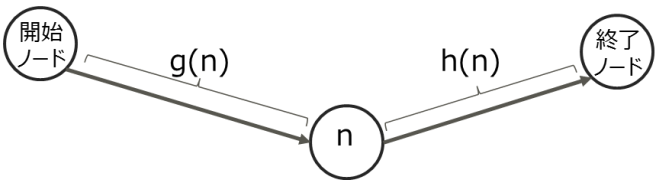

図 4 A*アルゴリズム

3. 課題

2.2 章で述べたように、我々の目標は構造的、構文的、意味的変換の全てに対応可能な PBE を実現することである。そのため、[6]が対応していない意味的変換処理を A*アルゴリズムグラフ探索に適用する。しかしながら、意味的変換処理を A*アルゴリズム探索に適用すると探索空間が爆発的に広がり、探索処理に時間がかかり現実的な時間で探索を完了することが不可能になってしまう。それは以下の 2 つの理由に起因する。

3.1. 結合候補の組み合わせ爆発

意味的変換では、外部テーブルを参照して必要なデータを引っ張ってくるが、このためには、変換されるカラム、変換されるテーブルの結合キーカラム、外部テーブルの結合キーカラムを決定する必要がある。A*アルゴリズムでナイーブにこの決定処理を実現すると、これらのカラム候補の組み合わせをしらみつぶしに試して中間ノードを作成し、これらの中間ノードのうち、最も f(n)の値が小さくなる経路を選択して探索処理を進める。しかしながら、このカラム候補の組み合わせは、変換されるテーブルのカラム数を t, 外部テーブルのカラム数を s とすると t×s のようにカラム数に比例して多くなるので、両テーブルのカラム数の増加により A*アルゴリズム探索の探索空間が膨大に広がってしまう。また、候補となる外部テーブルは一つではなく、結合し得る外部テーブル全てが候補になるので、外部テーブルの結合カラム候補数 s はこの外部テーブル群のなかの外部テーブルに存在するカラム数を全部

足した数となり、外部テーブル数が増えるとさらに膨大な探索空間となる。テーブルカラムの結合処理は、対象カラムの全データに対して参照と突合処理をする必要があるため、テーブル操作の中でも重いコストがかかる処理である。その処理を膨大な探索空間上で実行するので、グラフ探索処理が現実的な時間で終わらなくなるほど探索処理に時間がかかってしまう。

3.2. 自己結合の発生

我々は、上記の問題を、4.1 に詳記するようにデータ類似性によって結合カラム、結合キーカラム候補を絞り込む手法で解決する。この手法では変換されるテーブルのカラムと外部テーブルのカラムのうちで、カラムが持っている値の類似度が高いカラム同士を結合キーとして結合処理を実施する。これによりテーブル全カラムの組み合わせ数の結合を試行する必要がなくなり、組み合わせ爆発は回避できる。しかしながら、4.1 の手法をつかっても、同じ自己結合の問題が残ってしまう。一度結合処理を行うと以降は外部テーブルに存在するカラムが、変換されるテーブルにも存在するので、同じ外部テーブルのカラムを結合最有力候補として算出されてしまう。その為、積極的に再度結合を試行してしまう。同じ外部テーブルを使った結合を連続して実行することは、変換されるテーブルに既に存在する情報を単に追加するだけなので、実用上無意味な処理である。このような処理をここでは自己結合と呼ぶ。また、結合処理は他の処理と比較して多くの処理量を必要とするので、自己結合処理を行うことは処理コストの面で非常に不利である。グラフ探索処理を現実的な時間で終わらせるためには、この問題も解決する必要がある。

4. 提案手法

4.1. データ類似性による結合候補絞り込み

変換カラムと結合キーカラムの結合候補の組み合わせが爆発し探索空間が膨大になる問題を解決するために、データの類似性に基づいて結合候補の選択を行う手法を提案する（図 5）。

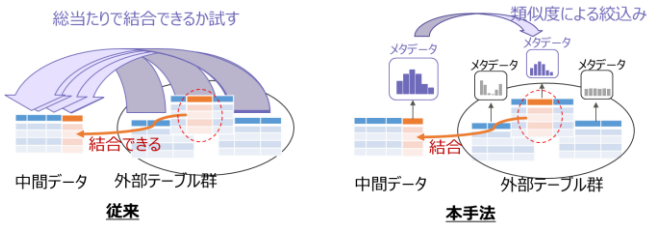

図 5 データ類似性による結合候補絞り込み

この手法では、変換プログラム導出処理を行う前にあらかじめ、外部テーブル群の各外部テーブルのカラムについて、データ型、データの値分布などのメタデータを計算し、保存しておく。このメタデータと中間データのメタデータの類似度を計算して、類似度が高いカラム候補から順番に結合を試行し、類似度が低く結合キーになりえない候補については類似度閾値により結合処理を回避することで、結合候補を絞りこむ。中間データのメタデータは、グラフ探索処理において中間データを生成するたびに、変更されたカラムに対してメタデータ算出を行う。グラフ探索処理中にも中間データに対してメタデータ算出を行うためグラフ探索の処理量に影響を与えるが、中間データのなかの変更されたカラムについてのみ実行すればよいし、中間データはユーザが例示として入力するデータが基本となるためデータ量は比較的少なく、中間データのカラムに対してメタデータ算出する処理量は軽微なものである。このようにあらかじめ算出しておいたメタデータの類似性に基づいて結合候補を絞り込むことにより、グラフ探索空間が大幅に減少する。

4.2. 処理履歴を利用した自己結合回避

自己結合を回避するため、A*アルゴリズム探索にマルコフ性を導入し、結合処理が過度に連続実行されることを回避する。そのために、各中間状態に、外部テーブル毎に結合処理のコストをあらわすカウンタを設ける。結合処理は該当外部テーブルのカウンタがゼロのときにだけ実行可能である。ある中間データに対して結合処理を実行すると、中間データの該当外部テーブルカウンタが閾値 N インクリメントされ、他の外部テーブルカウンタは 1 デクリメントされて、変換後の中間データの該当外部テーブルにコピーされる。ただし、カウンタがゼロになったらそれ以上デクリメントしない。この処理により、各外部テーブルに対する結

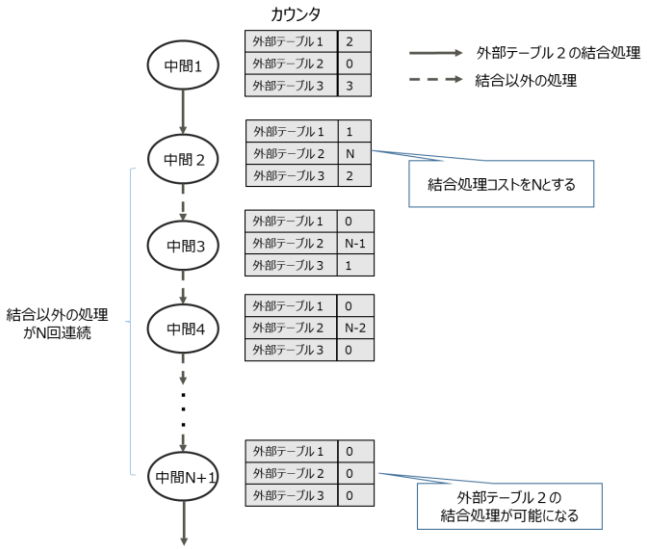

図 6 処理履歴による自己結合回避

合処理を最低 N 処理分保留し、結合処理の連続実行を回避する（図 6）。

5. 評価

5.1. 評価手法

今回の提案手法を評価するために、構造的変換、構文的変換、意味的変換、及びそれらを組み合わせた変換を必要とするデータ整形入出力例テストセットを用意してベンチマーク試験を行った。ベンチマークでは、今回追加した意味的変換を含む変換プログラム作成に成功するか、意味的変換の探索を加えても高速に変換プログラム作成が完了するかを確認する。また、既存技術と比較するために、Foofah の実装を Github の公開サイト[8]からダウンロードし実行環境を作成し、提案手法と Foofah とで同じテストセットで性能の比較も行った。評価用データは、我々が作成した 15 件のテストセット(fujitsu benchmark)及び Foofah[6]の評価に使われている 50 件のデータセット[9](foofah benchmark)の、合計 65 件のデータセットを使った。これらのテストセットを変換種別毎に分類すると、構文的変換のみで構成されるテストセット(変換種別 a)、構文的変換、構造的変換、意味的変換を組み合わせたテストセット(変換種別 b, c, d)に分類される。各変換種別のテスト数は表 2 のようになっている。

表 2　テストセットの変換種別

変換種別	fujitsu benchmark	foofah benchmark	計
a.　構文的変換	0	35	35
b.　構文的変換＋構造的変換	7	15	22
c.　構造的変換＋意味的変換	1	0	1
d.　構文的変換＋構造的変換＋意味的変換	7	0	7
合計	15	50	65

また、このベンチマーク試験では、変換プログラム生成に完了したら成功とみなし、ある時間内に成功するテストの割合を成功率としてこれを性能の指標値とした。

5.2. 意味的変換を含む変換プログラム生成

本節では、提案手法によって、意味的変換を含む変換プログラム生成が可能になったかを確認する。テストセットは、意味的変換を含む c, d のテストセット(全 8 件）を利用した。また、評価に使った手法は、意味的変換の追加のみを行って提案手法である結合候補の絞り込み採用しない手法（Semanti 有）と、提案手法の絞り込みを行う手法（Semantic 有、提案手法）の 2 つを用いて性能の比較を行った。テストの打ち切りのタイムアウトは 30 秒とした。ユーザがインタラクティブに画面操作するうえで処理が完了するまでに待てる時間としては、およそ数十秒程度と考えられるためである。評価結果を図 7 に示す。絞りこみ処理を行わない手法では、テスト時間の 30 秒を過ぎても成功するテストはなかった。それに対して、提案手法である絞り込みを行う手法では、プログラム生成を開始してから 2 秒後に全 8 テストのプログラム生成に成功した。この結果により、提案手法である結合候補の絞り込みを行うことで、探索空間の膨大な広がりを抑えて、意味的変換を含む変換プログラム生成を可能にしたことがわかる。

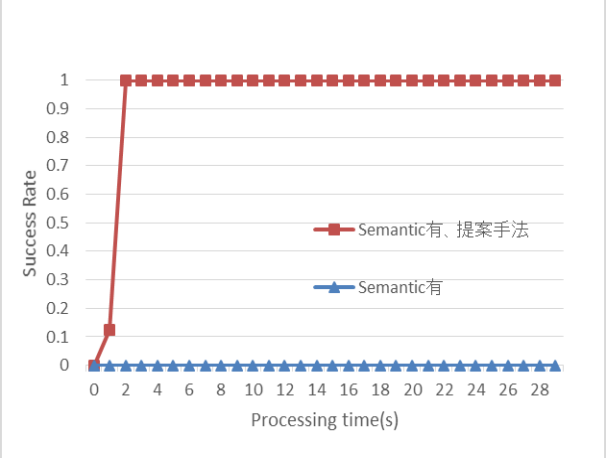

図 7 意味的変換を含むプログラム生成成功率

5.3. 構文的・構造的変換プログラム生成への影響

次に、意味的変換の追加が、性能へ負の影響を、構文的変換や構造的変換プログラムの生成にどれだけ与えるか確認する。意味的変換の追加により探索空間が広がるため、他の構文的変換や構造的変換も探索が遅延してしまうことが予想される。一方で提案手法の結合候補絞り込みによって探索空間の広がりを抑えることができると考えられるので、その探索空間増加の抑制効果を確認する。

テストセットとしては、意味的変換を含まない、変換種別 a（全 35 テスト）、b（全 22 テスト）の変換種別のテストを利用し、各種別について評価を行った。評価に使った手法は、意味的変換の追加のみを行った手法（Semantic 有）、提案手法（Semantic 有、提案手法）、意味的変換の追加を行っていないもの（Semantic 無）を使って比較した。その結果を図 8（変換種別 a）、図 9（変換種別 b）に示す。

意味的変換の追加のみを行った手法（Semantic 有）はどちらの変換種別でもタイムアウト（30 秒）までに成功しなかった。対して提案手法では、どちらの変換種別もタイムアウトまでにテストセットの 50%程度が成功した。つまり、提案手法の絞り込みで探索空間が

減少し、意味的変換以外の変換もプログラム生成が可能になったといえる。

提案手法と意味的変換を追加していない手法を比較すると、意味的変換を追加していない手法の成功率が高くなっている。しかしながら、多くのテストは差が 1 秒内に成功している。　また、1 秒内に成功していないテストも数秒の差で収まっている。この結果により、提案手法の絞り込みの効果により、提案手法の性能が意味的変換を追加していない手法の性能にかなり近いことが確認できた。

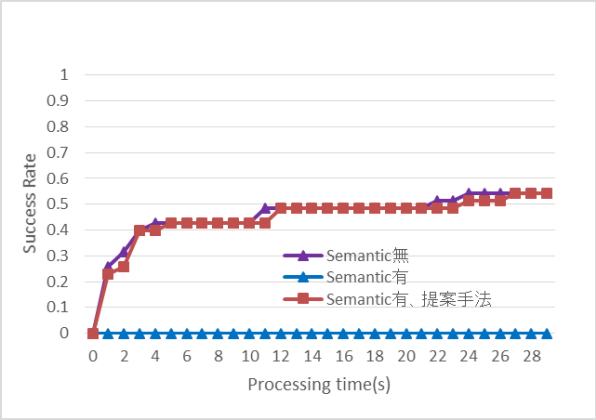

図 8 構文的変換のプログラム生成成功率

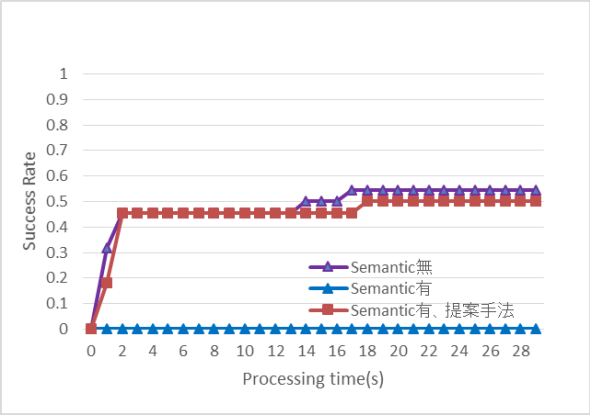

図 9 構文的・構造的変換のプログラム生成成功率

5.4. Foofah との性能比較

本節では、提案手法と Foofah との比較を行う。Foofah は我々と同じ A*アルゴリズムをグラフ探索アルゴリズムとして採用し、構文的変換と構造的変換に対応している。Foofah は意味的変換に対応していないので、意味的変換を含まない、変換種別 a 及び変換種別 b の計 57 件のテストセットを用いて、提案手法と比較を行う。計測結果は図 10, 図 11 に記載する。

Foofah では変換種別 a、b の成功率が開始後 30 秒で約 80%以上となり、高い成功率であった。一方で、提案手法では 50%を少し超える成功率であり、Foofah のほうが高い成功率となった。提案手法のほうが Foofah よりも変換プログラム生成の時間がかかったテストセットを精査したところ、構造的変換が多く、そのなかでも特にテーブルの行と列を入れ替える処理や、ピボット変換に相当する処理といった、テーブルの行数・列数が大きく変わるテーブル構造変換処理が多いテストであることがわかった。そして我々の提案手法では、特に大幅にテーブル構造が変換されるデータセットで処理時間がかかっている傾向があった。

Foofah のほうが構造的変換に向いている原因は、A*アルゴリズムで探索効率化に利用している距離評価関数の違いであると考えている。我々の提案技術では距離評価関数として文字列の編集距離をベースとした距離を利用しているのに対して、Foofah ではテーブル編集距離(Table Edit Distance Batch)というテーブル構造の類似度の精度が良い評価関数を採用していて、採用している距離評価関数の得手不得手が評価性能に影響している。今回のテストセットは構造的変換を含むテストセットが多いためこのような結果となったが、我々の提案技術がベースとしている文字列編集距離が

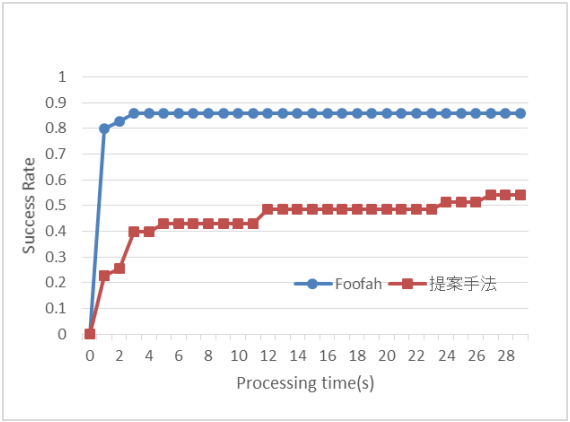

図 10 構文的変換のプログラム生成成功率 Foofah 比較

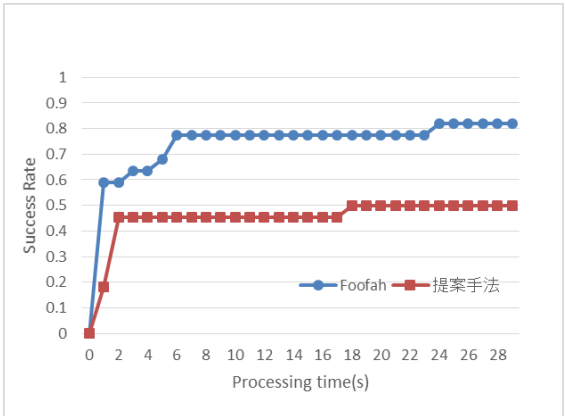

図 11 構文的・構造的変換のプログラム生成成功率 Foofah 比較

適している構文的変換が多いテストセットでは逆の結果になる可能性がある。

このような性能のデータ依存性を解消するために、構文的変換や意味的変換の成功率を維持しつつ、構造的変換も高い成功率を実現できる新しい工夫を今後技術開発し、提案手法の性能をさらに向上させる予定である。

6. まとめ

本論文では、データ分析プロセスのなかで多くの工数が費やされるといわれているデータ整形作業を自動化して効率化するために、Programming by Example (PBE)による自動化を紹介し、PBE をデータ整形に応用するために必要な意味的変換に対応させるために、結合候補の削減手法を提案した。この手法により、これまでの既存技術では無かった、構造的変換、構文的変換、意味的変換を包括的にサポートする PBE 技術の実現にめどをつけることができた。今後は節 5.4 で述べた課題の解決や機械学習の導入などによりさらに性能向上を行う予定である。この技術により、PBE による変換プログラムの自動生成が幅広いデータ整形の問題に対して現実的な時間で完了できるようになり、PBE を現実のデータ整形作業に応用することができるようになると期待している。

現在、本技術と A* アルゴリズムによるグラフ探索技術をベースにした PBE エンジンのプロトタイプを、フリーのオープンソースデータプレパレーションツールである OpenRefine に組み込み、PBE によるデータ整形自動化技術の実用化を進めている。これを実用化することによって、従来プログラミングが必要であったデータ整形が入出力例を提示するだけで行えるようになり、データ分析のためにデータ整形作業を行う際の障壁を大きく下げることができると考えている。

参 考 文 献

[1] New York Times, For Big-Data Scientists, ‘Janitor Work’ Is Key Hurdle to Insights. https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html?ref=business

[2] S. Kandel, A. Paepcke, J. Hellerstein and J. Heer. Wrangler: Interactive Visual Specification of Data Transformation Scripts. CHI 2011, May 7-12, 2011.

[3] D. Huynh and S. Mazzocchi. OpenRefine. https://github.com/OpenRefine/OpenRefine

[4] S. Gulwani. Programming by Examples. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/12/pbe16.pdf

[5] S. Gulwani. Automating String Processing in Spreadsheets Using Input-Output Examples. PoPL’11, January 26-28, 2011.

[6] Zhongjun Jin, Michael R. Anderson, Foofah: Transforming Data By Example. SIGMOD’17, May 14-19, 2017

[7] R. Singh, S. Gulwani. Learning Semantic String Transformations from Examples. VLDB, August 27-31 2012.

[8] https://github.com/umich-dbgroup/foofah

[9] https://github.com/markjin1990/foofah_benchmarks