卒業論文

表構造に基づく文書の翻訳

指導教官　　村上 陽平 准教授

立命館大学情報理工学部

情報コミュニケーション学科 4 回生

2600160247-0

玉置 晴菜

2019 年度（秋学期）卒業研究 3（2Q）

令和 2 年 1 月 31 日

表構造に基づく文書の翻訳

玉置 晴菜

内容梗概

近年，機械翻訳技術は急速に発展してきた．特に，ニューラルネットワークを用いたニューラル機械翻訳は目覚ましい進歩を見せており，課題とされてきた長文の翻訳もある程度の品質が保証されるようになってきた．その結果，ニューラル機械翻訳を用いた多様な応用システムが提案されている．文書翻訳はその一つであり，翻訳対象の文書のフォーマットを維持しつつテキストを翻訳することができる．しかしながら，従来の文書翻訳はフォーマットを維持することに主眼が置かれているため，表の翻訳おいては，表の個々のセルから単語や短文を抽出し，それぞれを独立に翻訳することで，表構造の文脈が無視され，誤訳が生じる．例えば，様々な文書に存在する生年月日の記入欄では，生まれの「月」の部分が「month」ではなく，「moon」と誤訳される．これは「月」という単語が「month」を表す月よりも，「moon」を表す月としての対訳データが多いため，ニューラル機械翻訳において「moon」が適していると判断されてしまうためである．

そこで本研究では，表構造から抽出される表中のデータ間の関係を文脈として用いて表中のテキストを翻訳する手法を提案する．具体的には，表中のセル間の関係から，翻訳対象のテキストを修飾する見出しラベルを抽出し，それらを翻訳対象のテキストと連結して翻訳する．その翻訳結果から見出しラベルに該当する部分を除去することで，表の文脈に沿った翻訳を生成する．手法の実現にあたり，取り組むべき課題は以下の２点である．

見出しラベル間の親子関係の同定

見出しラベルを用いて機械翻訳への入力文を生成するため，翻訳精度は表中のテキストを修飾する見出しラベルの解釈に依存する．そのために適切な見出しラベルの抽出と見出しラベル間の親子関係の同定が必要である．

文脈の除去

翻訳対象のテキストには表構造に埋め込まれた文脈を付加しているため，翻訳結果から見出しラベルの訳を除去する必要がある．

前者の課題を解決するために，表中のセル間の接続関係に着目する．表中のセル間の関係として，幅の広いセル（a）が幅の狭いセル（b）を囲むとき，セル（a）の要素をクラス名，セル（b）の要素をプロパティとすると，セル（a）の

要素はセル（b）の要素を修飾できると仮定することが可能である．そこで，セル（a）の要素とセル（b）の要素を助詞である「の」をデフォルトとして用い，「セル（a）の要素　の　セル（b）の要素」のように繋ぎ合わせて文脈を生成することで，対訳コーパスから学習されたニューラル機械翻訳内の翻訳モデルからより適切な訳が選択される．

後者の課題を解決するために，ラベルのみで翻訳したものを翻訳結果から除去する．要素が複数ある場合は，ラベルの部分が同一の翻訳結果になっている場合が多いため，任意の文字列が含まれているか判定することでラベル部分を識別，除去の演算を実現している．

提案手法を評価するために，ニューラル機械翻訳を用いた文書翻訳システムとして Google ドキュメント翻訳との比較を行った．その結果，提案手法である見出しラベルを用いた翻訳の方が，より表の文脈に沿った翻訳結果であったという点で向上していることが確認できた．本研究の貢献は以下の通りである．

見出しラベル間の親子関係の同定

表構造からセル同士の親子関係を同定し，見出しラベルでターゲットとなるセルを適切に修飾できるよう実装を行った．

文脈の除去

ターゲットとなるセルと同一の見出しラベルで修飾されたセルをそれぞれスペースで区切りリスト化し，翻訳結果が一致している部分を除去することで文脈の除去の実装を行った．

Translation of documents based on table structure

Haruna Tamaki

Abstract

In recent years, machine translation technology has developed rapidly. In particular, neural machine translation using neural networks has made remarkable progress. And it was able to translate long sentences to some extent well. As a result, various application systems using neural machine translation have been proposed. Document translation is one of them, and it can translate text while maintaining the format of the document. However, conventional document translation translates tables into words. So the table context is ignored and mistranslations occur. For example, in the date of birthday entry fields that exist in various documents, the 「月」 part of the birthday is incorrectly translated as “moon” instead of “month”. This is because the word 「月」 has more bilingual data as the 「月」 meaning "moon" than the 「月」meaning "month", so "moon" is judged to be suitable in neural machine translation.

In this study, extract relationships between data from the table structure and use that relationship as the context of the table. In particular, extract labels that modify the target from the table, and concatenate the target with them and translate them. And generates translations in the context of the table by removing the part corresponding to the heading label. There are two issues to be addressed in realizing this method.

Identification of parent-child relationship between heading labels

Translation accuracy depends on interpretation of heading labels that modify text in tables. Because the target remains modified to the heading label. Therefore, it is necessary to extract appropriate heading labels and identify parent-child relationships between heading labels.

Remove context

Need to remove heading label translation from translation result. Because the target adds context embedded in the table structure.

In order to solve the former problem, we focus on the connection between cells in the table. When a wide cell (a) surrounds a narrow cell (b), the element of cell (a) is a class name and the element of cell (b) is a property, then we can assume

that the element of cell (a) can modify the element of cell (b). Therefore, connect the element of cell (a) and the element of cell (b) with 「の」. For example, “element of cell (a) 「の」 element of cell (b)”. And generate the context, the neural machine translation well.

In order to solve the latter problem, remove label translations from translation results. When there are multiple elements, translation result of label part are often same translation result. So the label part can be identified and removed by judging whether an arbitrary character string is included.

To evaluate the proposed method, we compared it with Google document translation that using neural machine translation. As a result, the proposed method was more translated in the context of the table. So the proposed method is effective.

The contributions of this research are as follows.

Identification of parent-child relationship between heading labels

We implemented that program identify parent-child relationships between cells from table structure and heading labels can qualify the target.

Remove context

We implemented that program target and each cell modified by the same heading label, separated by spaces, listing, and remove parts where translation results match.

表構造に基づく文書の翻訳

目次

第1章 はじめに

近年，限界を迎えつつあったルールベース翻訳，統計的機械翻訳であったが，ニューラル機械翻訳が開発され，翻訳精度が飛躍的に向上した．Google の翻訳もニューラル機械翻訳を用いたサービスのうちの 1 つであり，他にも多様なサービスが実用化されている．例えば，Amazon も Amazon Translate というニューラル機械翻訳サービスを展開しており，アプリケーションとの統合やカスタム用語集機能を用いた固有名詞などの翻訳を自分で定義することができるというサービスを提供している．他にも，NTT が展開するニューラル機械翻訳サービス KOTOHA Translator では，グローバル化の進むビジネスシーンに向けた企業向けのファイル翻訳などを提供している．しかし，ニューラル機械翻訳を含む従来の機械翻訳の手法では，表の要素を翻訳する際に表中のセルの内容を単語や短文といった個々で抜き出し，フォーマットを保持したまま翻訳してしまうため，表の文脈に適した訳にならないという問題があった．例えば生年月日を表す「月」が「moon」，曜日を表す「火」が「fire」と誤訳されてしまう．本研究の目的は，表中のテキストを見出しラベルを用いて修飾することで，翻訳機が文脈からふさわしい要素の訳を導けるようにすることである．表中で幅の広いセルが幅の狭いセルを囲むとき，前者のセルが後者のセルを修飾する見出しラベルであると仮定し，対象となるセルのテキストに見出しラベルとなるテキストを繋ぎ合わせて翻訳することで，ニューラル機械翻訳がうまく作用し，翻訳精度が向上することを検証する．

文脈を考慮した表翻訳をする場合には，以下の点が重要となる

セル間の関係の同定

表構造からセル間の関係を読み取り，ターゲットのセルを修飾する見出しラベルの抽出が必要である．具体的には 3 章で説明する．

文脈の除去

文脈を付加された翻訳結果には見出しラベルの訳も添付されている．よって表のフォーマットを保持するためには文脈の除去が必要である．具体的には 4 章で説明する．

以降では，まず 2 章で文書翻訳，文脈を考慮した翻訳についての関連研究について述べる．次に 3 章で表からの意味ネットワークの抽出について述べる．4 章では文脈を考慮した表翻訳について説明する．5 章で提案手法を評価し，6 章で

結論を述べる.

第2章 文書翻訳

この章では，まず従来の機械翻訳の主流であるニューラル機械翻訳，統計的機械翻訳，ルールベース翻訳がそれぞれどのような翻訳システムであるかを説明する．その後，文脈を考慮した翻訳を紹介する．

2.1 機械翻訳

機械翻訳とは，ターゲットとなるテキストを目的の他言語に機械的に翻訳するシステムである．現在，機械翻訳には複数のベースとなるルールが存在する．おおまかに分類すると 3 種類の機械翻訳に分けることができ，入力言語と出力言語の両方を理解する言語の専門家が翻訳ルールを記述することによって実現されるルールベース翻訳，大量の対訳データから学習データを抽出し自動的に翻訳システムを構築する統計的機械翻訳，ニューラルネットワークを用いて機械翻訳を実現したニューラル機械翻訳がある．ここでは，それぞれの翻訳システムについて説明する．

ルールベース翻訳

ルールベース翻訳とは，入力テキストの形態素解析結果をそれぞれ出力言語に翻訳し，文法に基づいて主語や述語などを並び変える手法である．しかし，ルールの抽出には人手が必要であるため，話者が少ない言語であると専門家の確保が困難であるとともに，抽出したルールを他言語のルールに使いまわすことが不可能である．また，専門家が必要なため，開発コストが高いことも課題であった．

統計的機械翻訳

統計的機械翻訳とは，テキストデータから統計モデルを学習することで確立付き辞書を作成し，形態素解析されたテキストから抜き出された句をもとに生成した仮説の中で最も確率の高いものを正しいとする手法である．ルールベース翻訳と比較すると開発コストも低く，大量の対訳データさえあれば専門家を必要とせず，多言語翻訳が容易である．しかし，フォーマットを維持したまま翻訳するため，単語や短文では分脈を掴むことが困難であるという課題がある．また，開発コストは低いが大量の仮説を生成する必要があるため計算コストが高く，ルールベース翻訳を用いた翻訳結果のほうが統計的機械翻訳を用いた翻訳結果よりも有利となることもある．

ニューラル機械翻訳

ニューラル機械翻訳とは，ニューラルネットワークが翻訳に必要な何らかの情報を自動的に学習し翻訳するシステムであり，Google の機械翻訳にも使われている技術である．エンコーダーが形態素解析された句の意味をそれぞれベクトルで表現し，アテンション機構が注目すべき単語を確率的に判断，次にデコーダーが抜き出された単語のベクトルと確率の情報をもとに次の単語を出力することで実現されている．単語をベクトルで表現しているため，類義語であっても同じような翻訳結果となり，入力テキストの置き換えを用いた翻訳システムである統計的機械翻訳では実現困難であった翻訳が達成された．また，ベクトル計算を用いているため計算コストが低いといった利点もある．しかし，統計的機械翻訳と同じく文脈から単語の訳を推測するため，入力が短文や単語だと誤った翻訳結果になることがある．また，入力文を過不足なく翻訳することができないため，訳抜けや重複訳が発生するなどの課題もある．

2.2 文脈を考慮した翻訳

統計的機械翻訳，ニューラル機械翻訳で文脈を推測するシステムが採用されているように，機械翻訳の精度を向上させるためには文脈を考慮した上での翻訳が非常に重要ある．しかし，全ての言語間での翻訳には n(n-1)個の機械翻訳が必要であり，話者が少ない言語であれば統計的機械翻訳，ニューラル機械翻訳における対訳データが十分に集まらない可能性もある．そこで，中間言語を介して複数の機械翻訳を接続するピボット翻訳と，文脈情報を管理する協調エージェントを組み合わせた翻訳を紹介する．

協調エージェントは考えられるすべての文脈をもち，ユーザから原文を受け取ると，そこから関連のある文脈を選択し，翻訳エージェントに原文と文脈を送信する．一方，翻訳エージェントは担当する言語の知識を有しており，受け取った訳文に基づいて文脈を更に織り込み，後続する翻訳エージェントに文脈を次々に伝搬させていきながら翻訳全体の制御をおこなう．各翻訳エージェントは互いにどのように翻訳しているか感知しなくても，文脈に従うことで誤訳になることを防ぐという仕組みである [1].

また，制約の最適化による単語選択によって文脈を考慮する手法も存在する．この手法では，翻訳された単語を元の文書の名詞に割り当て，翻訳された単語間の意味的関連性の合計を最大化することで最適化を実現している．単語の選択

方法は，同じ文中に存在する単語は意味的関連性があると考えから，ターゲットとなる名詞の Wikipedia から単語のベクトルを抽出することで意味の関連性を求めている．さらに，単語のベクトルを得ることで文書全体の文脈を考慮することができる[2].

2.3 文書翻訳支援ツール

ここでは，文書翻訳を支援するツールに関する既存研究を紹介する．文書の翻訳は労働，留学など他言語間でのコミュニケーションを必要とする場では非常に重要となってくる．そこで，オフショア開発におけるビジネス文書の翻訳についての研究を紹介する．

ビジネス文書の翻訳では正確性が求められるため，全てを自動化するのは困難である．そこで人手による翻訳作業を支援し，翻訳にかかる時間を軽減しつつ，翻訳の品質を高めることを目的として，翻訳支援環境を構築している．この翻訳支援環境は，リソース管理，文書の品質チェック，人手による翻訳作業の 3 つのプロセスを反復して行うことで，翻訳のためのリソースを拡充しながら翻訳作業の効率を高めていくことを目的としている．また，辞書，翻訳メモリはサーバー上で管理されており，言語リソースの共有を行うことでチームで分担して翻訳作業を行う際に品質の平準化を図っている．システムの一部に Keyword Candidate Extractor (KCE) というものが存在する．これは，オフィス文書中に含まれる名詞句をキーワードとして抽出する．これから翻訳を行う文書に対して KCE を実行し，頻度の高い順番にキーワードの訳語を TMS に登録していくことで，翻訳対象に即した効率のよい辞書リソースの作成が可能となる[3].

ビジネス文書にはもちろん表構造を含むワークシートなどの翻訳も含まれており，文脈の取得を KCE で行っていることがわかる．しかし，あくまで翻訳は手動であり，表の文脈を取得した機械翻訳は考慮されていない.

他にも，表構造はビジネス文書だけでなく公文書にも多数存在しており，公文書もまた他言語間のコミュニケーションの場に深く関係している．そこで，欧州共同体（ EC ）の公文書をすべての加盟国の使用言語に翻訳する機械翻訳システムについての研究について紹介する．

EC 憲法と呼ばれているローマ条約では，EC の公式文書は加盟国のそれぞれの公用語で作成することになっている上に，現在 EC では 9 つの公用語が使用されているため，各々の言語を他の全ての言語に翻訳すると 72 通りの組み合

わせとなってしまう．そこで，機械翻訳開発プロジェクト EUROTRA が推進されてきた．EUROTRA の翻訳方式は変換方式であり，翻訳プロセスは解析，変換，生成の 3 つの段階に分かれている．原文は言語学的に形態素レベル，構文レベル，意味レベルの 3 つを経て元の言語の中間表現となる．この中間表現はそのまま変換ルーチンへの入力となり，変換ルーチンで 2 言語変換辞書を参照して目標言語の中間表現への構造変換が行われる．変換ルーチンの出力が目標言語の中間表現となり，目標言語の文生成ルーチンの入力になる．生成ルーチンでは解析ルーチンと逆の方向へ進み，目標言語の文が生成されるという仕組みである[4]．しかしここでも文書内の表については触れられておらず，表の文脈が考慮されているとは考えられない．

第3章 表から意味ネットワークの抽出

3.1 表の定義

本研究では CSV ファイルを用いて表構造を解釈する．しかし CSV ファイルでは構造の異なる表でもすべて同じ記述になってしまう場合がある. 例を図 1 に示すと，図 1 中の 3 つの異なる表をそれぞれ CSV ファイルに変換した場合，すべて同一のものとなってしまう．したがって，一意に表を解釈するには表の定義が必要である.

今回の表の定義を以下に示す.

1. 空白セルは左，上のセルに優先して連結する
2. あるセルの幅や高さは，上辺，左辺で接する上側，左側のセルの幅や高さを越えない
3. あるセルの幅や高さより上辺，左辺で接する上側，左側のセルの幅や高さの方が大きければ，当該セルは上側，左側のセルの子ノードとする
4. あるセルの幅や高さと上辺，左辺で接する上側，左側のセルの幅や高さが同じ場合，当該セルは上側，左側のセルの兄弟ノードとし，同じ親ノードを継承する

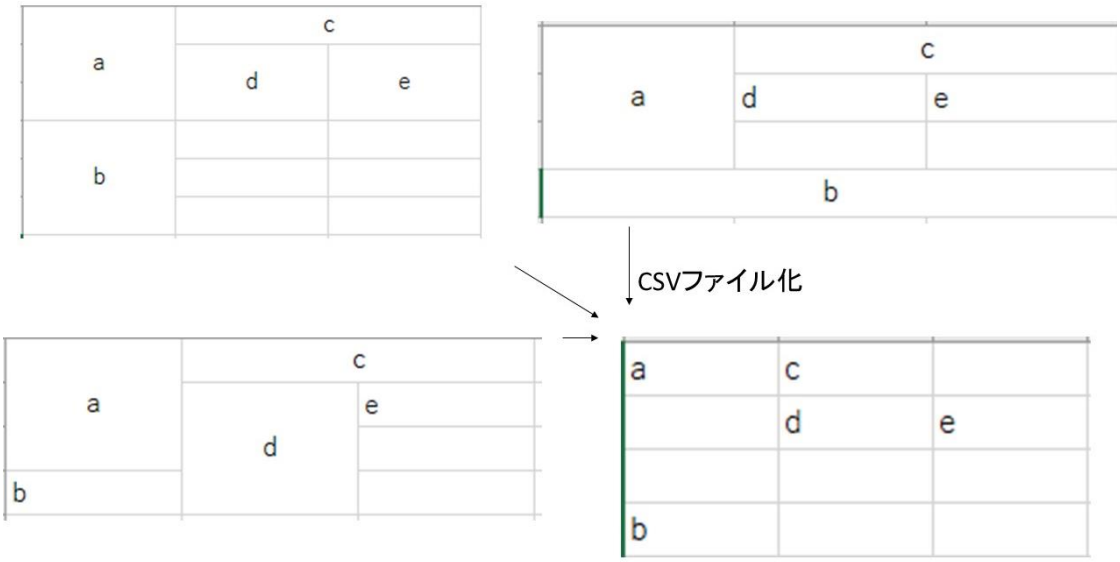

図 1.SVC ファイル化

また，今回は必ず上接するセルに幅が包含する，あるいは左接するセルに高さが包含するように解釈しているので右端，下端の決定は以下の通りである．

セルの右端の決め方：上接の右端か，右接の左端によって決まる

セルの下端の決め方：左接の下端か，下接の上端によって決まる

3.2 セル間の関係の同定

見出しラベルの選択は翻訳対象のテキストの解釈を決定づける．そのため，翻訳精度の向上には提案手法におけるセル間の関係の同定は非常に重要である．

表はある程度決まった構造を持っており，その構造により表中のデータ間の関係が表される．しかし広く Web から表を集められた表を対象とする場合には，ある表構造がどのような関係を表現するのに用いられるかは基本的に表によって異なるため，それぞれの表に応じた構造の解釈を用いて，表中のデータ間の関係を獲得する必要がある．表の観察の結果，ある表構造が表中のデータ間の特定の関係を表すとき，その表構造が表す関係は同じ表の中では多くの場合一定である．セルの隣接を図 2 のように 1 方向または双方向のセル間の接続として表す[5]．

図 2 で示したセルの関係を図 3 に当てはめると，「月」「火」「水」「木」「金」「土」「日」や「一週目」「二週目」「三週目」「四週目」，「名前」「生年月日」は重なる辺の長さが同じなので，似た種類のデータが記入されているといえる．実装では幅や高さが同じものは兄弟関係と判定しており，兄セルの親は弟セルの親であるとしている．また，「勤務表」は「名前」と「生年月日」を，「生年月日」

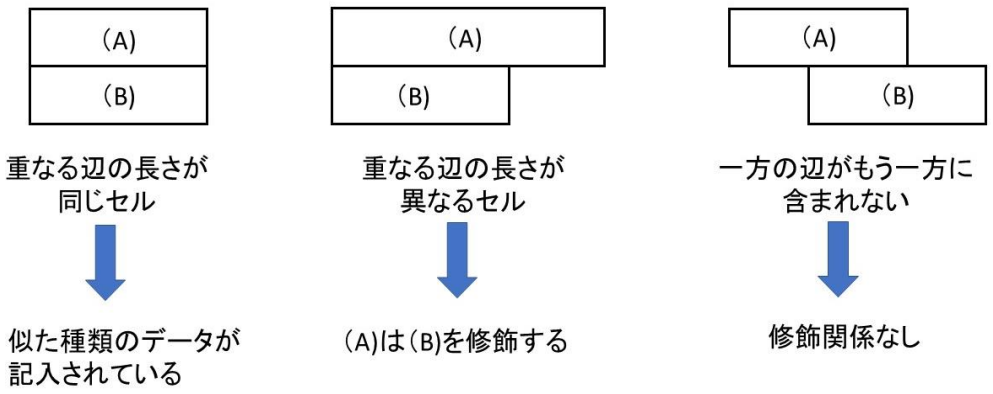

図 2.隣接するセルの関係

は「年」「月」「日」を，「曜日」は「月」「火」「水」「木」「金」「土」「日」を，見出しラベルとしてそれぞれを修飾する．このとき，「生年月日」は見出しラベルであるが，同時に「勤務表」とも親子関係が成り立つので，接続は「勤務表」→「生年月日」であると共に「勤務表」→「生年月日」→「年」でもある．これらの親子関係を意味ネットワークで表すと図 4 のようになる．

勤務表							
名前		生年月日					
			年		月		日
	曜日						
	月	火	水	木	金	土	日
一週目							
二週目							
三週目							
四週目							

図 3.勤務表の事例

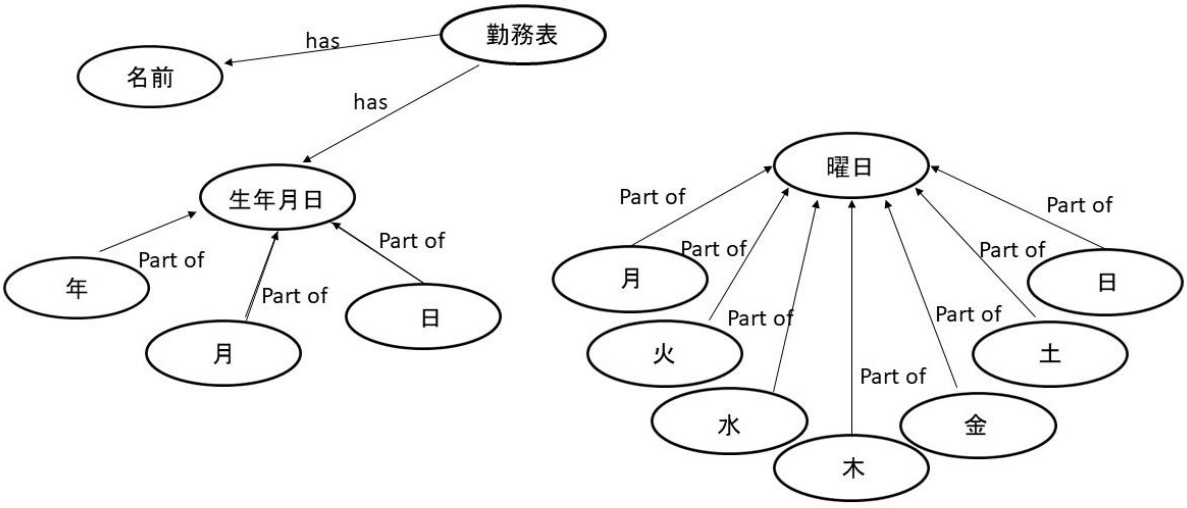

図 4.図 3 の意味ネットワーク

第4章 文脈を考慮した表翻訳

4.1 概要

ここでは，表翻訳の全体の流れを説明する．本研究では，Web 上の表を表中のセルの構成が単純で 1 つの表であるといえる, かつ 1 つのセルに 1 つのテキストのみが存在する，かつ特殊文字や記号などを除去して整理したものを CSV ファイル形式に変換したものを用いている．

表翻訳の流れを以下に示す．

1. 表の入力では CSV ファイルを「,」で区切り，セルのテキスト，位置をリスト化し，位置が決定したセルから順にグラフに追加する
2. 表から意味ネットワークを抽出し，セル同士の親子関係からノードを繋ぐ
3. ターゲットとなるセルと見出しラベルの接続では「の」を用いて，意味ネットワークの上位から下位まで繋ぐことで文脈を生成する
4. 生成した文脈をそれぞれ翻訳する
5. 翻訳結果には見出しラベルの訳も含まれているため，見出しラベルの訳を除去する

今回は例として Web 上から引用した図 5 を用いる．図 6 は図 5 から抽出した親子関係のグラフである．図 5 中の左側にある「大特・小特」をターゲットと仮定し，表翻訳の過程 2 から実行すると

2．意味ネットワークを抽出し，セル同士の親子関係からノードを繋ぐと図 5 のようになる

使用の本拠（事業所）における自動車台数及び運転者数	自動車台数	乗用				貨物				大特・小特	自動二輪	計
		大型・中型	準中型	普通	軽	大型・中型	準中型	普通	軽			

図 5. 安全運転管理者等に関する届出書

3. 接続すると「自動車台数」の「大特・小特」

4. 翻訳すると“‘‘Large specialty / small specialty ’’ the number of vehicles”

5. 見出しラベル「自動車台数」の翻訳結果，“of the number of vehicles”を除去

「大特・小特」をそのまま翻訳すると“Big and small”になり，特殊の「特」の部分が消えてしまう．「大特・小特」は大型特殊自動車，小型特殊自動車を省略したものなので，“Large special/Small special”と翻訳されるほうがふさわしいと言える．

また，今後の発展として意味ネットワークのノードのラベルを用いることができる．図 7 は図 5 から抽出した意味ネットワークモデルであり，意味ネットワーク中の「自動車台数」と「乗用」を結ぶノードのラベルは「has」，「乗用」と「大型・中型」のノードのラベルは「part of」となる．実装での見出しラベルとの接続は「の」で実現しているが，図 7 にあるようなノードのラベルを人手で与えることによりセル同士の関係性をより詳しく同定することができ，翻訳精度の向上に繋がると考えられる．例えば英語の場合であれば，ラベルが「has」の場合「自動車台数」と「乗用」を結ぶと“number of passenger cars”となり，“of”で接続されているのがわかる．「part of」の場合「乗用」と「普通」を接続すると“ordinary car for passenger”となり“for”で接続されているのがわかる．

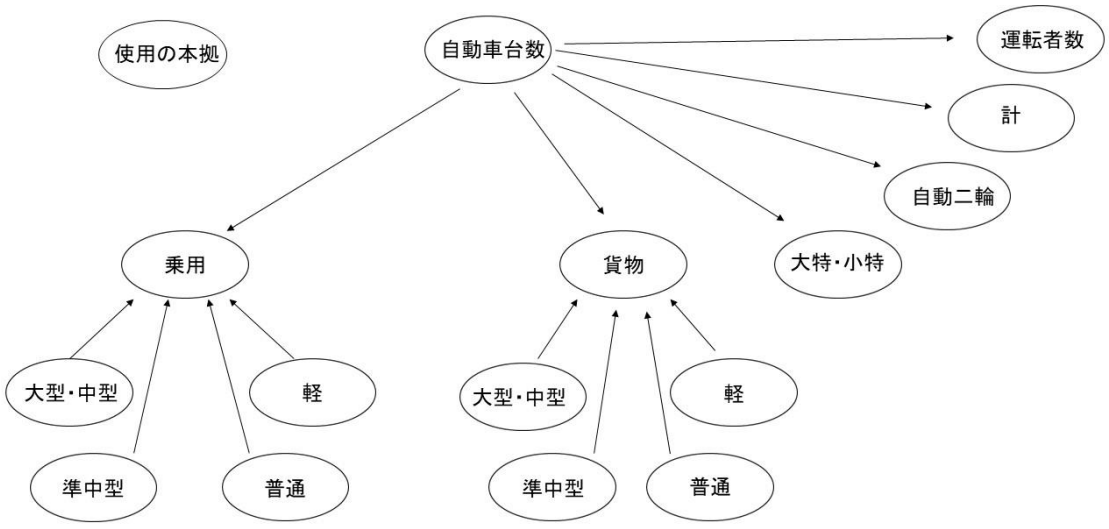

図 6. 安全運転管理者等に関する届出書の親子関係のグラフ

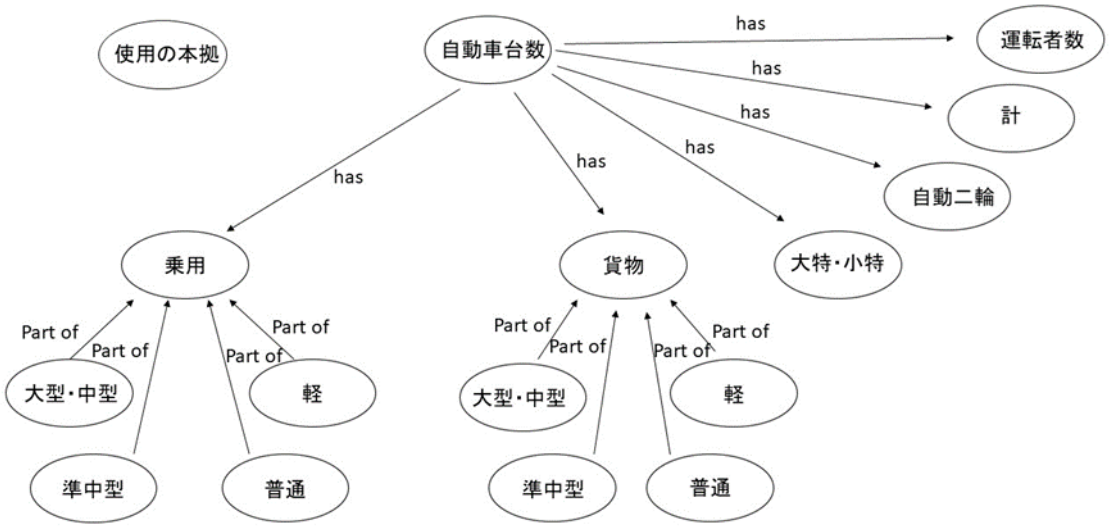

図 7. 安全運転管理者等に関する届出書の意味ネットワーク

言語ごとに接続に用いられる語は異なるが，ある程度決まっており応用可能である．

また，今回は日英翻訳のみを対象としており，接続に用いるデフォルトとして「の」を用いているが，言語ごとに変換規則を用意することで日英翻訳以外への応用も可能である．

4.2 意味ネットワークから文脈の生成

実装では，各セルから最上部セル（表枠外の上部に表の幅のセルを仮定），もしくは最左部セル（表枠外の左部に表の高さのセルを仮定）への最長パスから文脈を作成している．図 8 は検出したセルの一覧を表示したもの，図 9 は構築したグラフを視覚化したのである．図 8 の ID:1 は最上部セル，ID:2 は最左部セルを表す．ID:8「自動二輪」に注目し，図 9 のグラフからノードを読み取ると，ID:8→ID:1，ID:8→ID:4 であると確認できる．ID:1 は実在しないセルであるので，実際の接続は ID:8→ID:4 のみである．実装ではこれらを用いて文脈を生成しており，実際に生成された文脈は自動車台数の自動二輪であり，翻訳結果は“Motorcycles with the number of vehicles”となる．自動二輪だけで翻訳した結果は図 10 にあるように“Self-moving two-wheeled”となってしまうが，見出し

ラベルを用いることで自動二輪車を意味する“Motorcycles”に変換される.

```
ID:1, v:None, left:0, top:0, right:13, bottom:0
ID:2, v:None, left:0, top:0, right:0, bottom:2
ID:3, v:使用の本拠（事業所）における自動車台数及び運転者数, left:1, top:1, right:1, bottom:2
ID:4, v:自動車台数, left:2, top:1, right:2, bottom:2
ID:5, v:乗　　用, left:3, top:1, right:6, bottom:1
ID:6, v:貨　　物, left:7, top:1, right:10, bottom:1
ID:7, v:大特・小特, left:11, top:1, right:11, bottom:1
ID:8, v:自動二輪, left:12, top:1, right:12, bottom:1
ID:9, v:計, left:13, top:1, right:13, bottom:1
ID:10, v:大型・中型, left:3, top:2, right:3, bottom:2
ID:11, v:準中型, left:4, top:2, right:4, bottom:2
ID:12, v:普通, left:5, top:2, right:5, bottom:2
ID:13, v:軽, left:6, top:2, right:6, bottom:2
ID:14, v:大型・中型, left:7, top:2, right:7, bottom:2
ID:15, v:準中型, left:8, top:2, right:8, bottom:2
ID:16, v:普通, left:9, top:2, right:9, bottom:2
ID:17, v:軽, left:10, top:2, right:10, bottom:2
```

図 8. 検出したセルの一覧

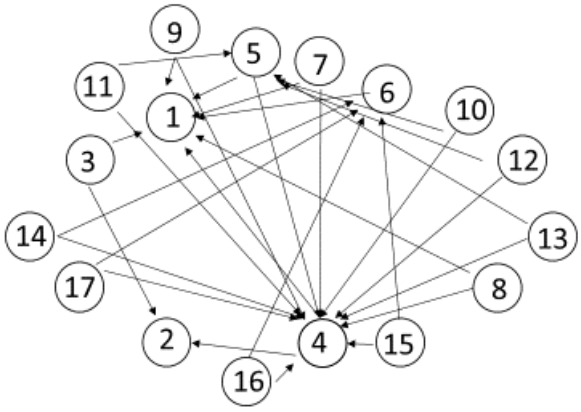

図 9. 構築されたグラフ

Number of cars and drivers at the base of use (office)	Number of cars	Riding				Cargo				Large special - small Features	Self-moving two - wheeled	Total
		Large / medium	Quasi-medium type	usually	Light	Large / medium	Quasi-medium type	usually	Light			

図 10.図 5 の翻訳結果

4.3 文脈の除去

4.2 では文脈の生成について述べたが，単語の翻訳結果は見出しラベルが接続されたままになっているため，表のフォーマットを維持するためには翻訳結果からラベルの訳を取り除かなければならない．同一の見出しラベルで修飾されたセル同士の翻訳結果は，見出しラベル部分の訳が類似していることがほとんどであるため，実装では，リスト差分を用いてすべてのデータで重複データを残して差分データを求める方法で見出しラベル部分を除去している．具体的には，「大特・小特」をターゲット，同一の見出しラベルで修飾されるセルを「自動二輪」，見出しラベルを「自動車台数」とすると，まず見出しラベルを接続した翻訳結果は，「自動車台数」の「大特・小特」では ““Large specialty / small specialty" of the number of vehicles，「自動車台数」の「自動二輪」では “Motorcycles with the number of vehicles”となる．次に重複データを求めると，この場合では “the number of vehicles”が重複しているため，繰り返しの部分を ““Large specialty / small specialty” of the number of vehicles”から除去すると ““Large specialty / small specialty” of”となる．実際の実行結果は図 11 である．「大特・小特」の翻訳結果が “Big and small”であったのに対し，提案手法では “Large specialty / small specialty”となっていることから提案手法は有効であるといえる．また，だが，図 11 にもあるように，不要な “of” などの単語が残ってしまう，“special” ではなく “specialty”となってしまうなど，品詞が変化してしまう場合もあるため，表のフォーマットを保持するための課題が残る面もある．

```
['"Large', 'specialty', '/', 'small', 'specialty"', 'of']
```

図 11.ラベルの除去実行結果

第5章 評価

ここでは，実際に google ドキュメント翻訳を用いて翻訳したものと，提案手法を用いて翻訳したものを比較した結果を示す．まず，評価に用いる表を入手するために，Web 上からサンプルとなる Excel ファイルを選び出した．このとき，なるべく表の文脈をとらえやすくするために，見出しラベルとなりうるセルが多い Excel ファイルを選出した．選出した Excel ファイルをあらかじめ google ドキュメント翻訳にかけ，表の誤訳箇所を記録した．見出しラベルを用いて誤訳箇所を修飾したテキストを翻訳し，表の文脈に沿ったテキストに変化したものが多かった表をサンプルとして選択した．次に，サンプルの表は複数の表が結合しているものや，表中のセル結合などで複数のセルにわたってテキストが挿入されているなど，表の構成が原因で翻訳の結果が変わる可能性があるものも多かったため，整形の必要があった．そこで，見出しラベルと誤訳箇所のテキストを残したターゲットとなる部分のみを抜き出し，表中のセルの構成が単純で 1 つの表であるといえる，かつ 1 つのセルに 1 つのテキストのみが存在する，かつ特殊文字や記号などを除去し，整形することでドキュメント翻訳に不要な要因を取り除いた．また，実装では整形した表を CSV ファイル化したものを用いて翻訳した．

図 12 は web 上の表を整形，図 13 は図 12 を google ドキュメント翻訳で翻訳した結果である．図 12 の「乗用」「準中型」「普通」「自動二輪」「大特・小特」は車の種類を説明する内容であるのに対し，翻訳結果は“Riding”“Quasi-medium type”“usually”“Self- moving two -wheeled”“Large special / small Features”となっている．それぞれの正しい訳は，「乗用自動車」は“Passenger

使用の本拠（事業所）における自動車台数及び運転者数	自動車台数	乗用				貨物				大特・小特	自動二輪	計
		大型・中型	準中型	普通	軽	大型・中型	準中型	普通	軽			

図 12. 安全運転管理者等に関する届出書（再掲）

car”「準中型自動車」は“Semi-medium-sized car”,「普通自動車」は「Ordinary car」,「自動二輪車」は“Motorcycle”,「大型特殊自動車・小型特殊自動車」は“Large special/Small special”であるので，google ドキュメント翻訳では表の文脈が無視されてしまっている．そこで，図 12 に提案手法を用いて翻訳した結果を図 14，図 15 に分割して示す．今回は見出しラベルを除去する際に，同じ見出しラベルのみで修飾された左接するセル，左接するセルがない場合は右接するセルを用いて見出しラベルの訳を判定し．図 13 と図 14，図 15 を見比べた結果，「乗用」「自動二輪」「準中型」の翻訳結果はそれぞれ“Passenger cars”“Motorcycles with the”“Semi-medium for passenger car”となり，表の文脈に沿った訳に変化したといえる．しかし,「大特・小特」の翻訳結果は“Large/ small”となり「特殊」の部分の訳が消えてしまった．また,「普通」は“Ordinary”には変化せず,“Normal”に変化してしまった．そこで，ラベルとターゲットの連結の順番を入れ替え,「普通」の「乗用」の「自動車台数」として翻訳した結果,“Number of ordinary passenger cars”となり表の文脈に適した翻訳結果に変化

Number of cars and drivers at the base of use (office)	Number of cars	Riding				Cargo				Large special - small Features	Self-moving two - wheeled	Total
		Large / medium	Quasi-medium type	usually	Light	Large / medium	Quasi-medium type	usually	Light			

図 13.図 12 の翻訳結果

Number of cars and drivers at the base of use (office)	Number of cars	Passenger cars			
		Large and medium-sized vehicles	Semi-medium for passenger	Normal the number of cars used	Passenger light vehicles

図 14. 提案手法を使用した結果 1

Car number of freight				Large/ small	Motorcycles with the	Total
Large / medium-sized	Semi-medium-sized cargo	Freight Normal	quantity light			

図 15. . 提案手法を使用した結果 2

した．そのため，ラベルとターゲットの連結には順番も重要であるといえる．ラベルの除去が不十分であるものも多く，表のフォーマット維持のためにはラベルの除去の精度を向上させることが今後の課題である．

同様に，住宅の点検に関する表に提案手法を用いた結果を示す．図 16 は Web 上から取得した住宅の点検に関する表を整形したもの，図 17 は図 16 の表を google ドキュメント翻訳を用いて翻訳した結果である．図 16 の「軸組」「小屋組」「雨どい」「軒裏（軒裏天井）」は住宅の設備を説明する内容であるのに対し，翻訳結果は“Framing”，“Hut”，“Keep in rain”，“Behind the eaves (ceiling behind the eaves)”となり，少し意味のずれた翻訳結果となっている．それぞれの正し

点検部位						
スケルトン	骨組	基礎	割れ	蟻道	不同沈下	換気不良
		土台・床組	腐朽	蟻害	さび	床の沈み
		軸組	腐朽	蟻害	破損	割れ
		小屋組	腐朽	蟻害	さび	はがれ
	屋根	瓦葺き	ずれ	割れ		
		厚形スレート瓦葺き	色あせ・色落ち	ずれ	割れ	さび
		金属板葺き	色あせ・色落ち	さび	浮き	
		雨どい	詰まり	はずれ	ひび	
		軒裏（軒裏天井）	腐朽	雨漏り	はがれ	たわみ

図 16. 長期優良住宅

い訳は,「軸組」は“Framework”「小屋組」, は“Roof truss”,「雨どい」は“Gutter”,「軒裏（軒裏天井)」は“Eaves back (Eaves ceiling)”である．ここで提案手法を用いると,「軸組」「小屋組」の見出しラベルは「点検部位」の「スケルトン」の「骨組」雨どい」「軒裏（軒裏天井)」の見出しラベルは「点検部位」の「スケルトン」の「屋根」となる．実際に提案手法を用いて翻訳した結果が図 18 である．「雨どい」は“Rain gutter on the the”となっており，表の文脈に沿った訳に変化したことがわかる．しかし「軸組」「小屋組」「軒裏（軒裏天井)」の翻訳結果はそれぞれ“Inspection skeleton”，“Skeleton framing shed for”，“Behind eaves of the (eave behind ceiling)”となっており，google ドキュメント翻訳の結果と比べてあまり変化がない，あるいはさらに意味が変化したものもある．これはニューラル機械翻訳が文章を翻訳するとき，文脈のつじつまを合わせようとすることで訳抜けや重複訳が発生したためであると考察される．実際に，ラベルを全て接続するのではなく一部を用いて修飾すると,「軸組」では「骨組み」をラベルとして連結させると“Framed framework”となり，表の文脈に適した翻訳結果に変化する．「小屋組」「軒裏（軒裏天井)」でもラベルを選択し連結してみたが“Roof truss”，“Eaves back (Eaves ceiling)”には変化しなかった．ラベルの除去においても，ニューラル機械翻訳の訳抜けや重複訳が起こったことによってラベルの除去が不十分となった，また連結したラベルの数が多くニューラル機械翻訳の特徴である単語同士の意味ベクトルが分散してしまったことが原因で表の文脈に沿った訳に変化しなかった,「の」の部分の翻訳結果が多く残ってしま

		Inspection site	Main inspection items			
skeleton	Frame	Foundation	Crack	Ant way	Differential settlement	Poor ventilation
		Base / floor	Decay	Ant harm	rust	Sinking floor
		Framing	Decay	Ant harm	Corruption	Crack
		Hut	Decay	Ant harm	rust	Peeling
	roof	Tiled	Gap	Crack		
		Thick slate tiled roof	Fading / fading	Gap	Crack	rust
		Metal sheet	Fading / fading	rust	float	
		Keep in rain	Jam	Off	crack	
		Behind the eaves (ceiling behind the eaves)	Decay	Leak	Peeling	Deflection

図 17. 図 16 の翻訳結果

Inspection site					
Skeleton of	skeleton	Inspection framework foundation	Skeleton crack at	Ant way frame of	Differential subsidence of skeleton
		Skeleton base / floor frame	Decay of	Ant damage to at	Rust of the frame of the inspection
		Inspection skeleton	Decay of	Ant damage	Damage to the skeleton frame at the
		Skeleton framing shed for	Decay of	Ant damage to at	Rust of of
	roof	Inspection tiled	Misalignment of the skeleton the	Skeleton crack	
		Thick slate roofing	Fading and fading	Misalignment of the skeleton the	Skeleton crack at
		Metal sheeting of	Fading and fading of the the	Rust on skeleton	Skeleton lift
		Rain gutter on the the	Clogged of	Missing at inspection	Inspection crack
		Behind eaves of the (eave behind ceiling)	Decay	Leakage of	Peeling off

図 18. 図 16 に提案手法を用いた結果

ったことが読み取れる．これらの改善策として，

表の翻訳結果からどれが誤訳であるかを判断することで，ドキュメント翻訳

で正しく翻訳されていたものは残し，誤訳部分だけに提案手法を用いる

　表の文脈に適した翻訳結果に変化するような連結ラベルの選択，

　意味ネットワークから導き出されたラベルの連結ルールを用いることで，「の」の部分の除去精度を向上させる

などが考えられる．

第6章 おわりに

多言語間のコミュニケーションを支援するために表の翻訳を自動化し，コストを削減することは, グローバル化に向け必要不可欠である. 今後も留学や労働, 生活のために他言語を使用する地域で生活する人々は増え続けるが，表を含む文書は現状の機械翻訳技術だけでは誤解が発生する可能性が高いといえるためシステムの改善が望まれる.

本研究では，表から見出しラベルを用いて文脈を生成することで表翻訳の精度を向上させる手法を提案した．そして生成した見出しラベルをターゲットに接続し翻訳することで，表中の単語が表の文脈に沿った訳に変化することを示した.

表中のセル同士の位置を判定することで親子関係を同定し，そこから意味ネットワークを構築することでセル同士を接続し文脈を生成した．生成した文脈を機械翻訳し，付加された余分な見出しラベルを除去することで表の意図に沿ったテキストに変化させた．その後，google ドキュメントを用いて翻訳したものと提案手法を用いた翻訳結果を見比べ，精度が向上しているかどうかの判定を行った.

本研究の貢献は以下の 2 点である.

見出しラベル間の親子関係の同定

CSV ファイル化した表を入力することで，自動的に親子関係を判定するプログラムの実装を行った. また, 同定した親子関係を用いて表から文脈を抽出することができた.

文脈の除去

表の文脈を付加されたテキストから文脈を除去する手法を提案, 実装した. これは日英翻訳だけでなく，他言語間の翻訳にも応用可能であると推測される.

今後, 表の翻訳を日英翻訳以外にも応用させることを考えた場合, 見出しラベルを用いて接続するためのルールの変更が言語ごとに必要となるが，意味ネットワークのノードのラベルを使用することで接続に用いる語を判定可能である.

また，見出しラベルの除去に関しては精度がまだ甘く，付加された文脈が十分に取り除かれていないものもあった. 解決のためには, 見出しラベルの訳も同時に取り除く，名詞の複数形も同一単語と判定するなどが有効であると見込まれる. 他にも，ラベル選択の精度を上げるためにベクトルを利用する，ラベルが

複数ある場合ラベルと要素を一つ一つ繋げ，要素の翻訳結果を多数決するなど，拡張の余地は十分あるといえる．

謝辞

本研究を行うにあたり，熱心なご指導，ご助言を賜りました村上陽平教授に深謝申し上げます．最後に，日ごろからお世話になっている村上研究室の皆様に心より感謝いたします．

参考文献

[1] 村上陽平，田仲理恵，石田亨：サービス連携時の文脈を用いたピボット翻訳の品質改善，電子情報通信学会論文誌2014/1， Vol.J97-D ，No.1，pp.165-167 (2013).

[2] Jun Matsuno，Toru Ishida: Constraint Optimization Approach to Context Based Word Selection，Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence，pp.1846-1848.

[3] 伊川洋平，金山博，竹内広宜，渡辺日出雄，三品拓也，秋本仁志，清水淳也：オフィス文書のための翻訳支援ツールの実装と評価，The 23rd Annual Conference of the Japanese Society for Artificial Intelligence，pp.1-2 (2009)

[4] 板橋慶造，米澤稔：欧州共同体（EC）における機械翻訳システム，情報の科学と技術，43巻，5号，pp.446-449(1993)

[5] 田仲正弘，石田亨：表構造の一般化に基づくオントロジの獲得，情報処理学会論文誌，Vol.47，No.5，pp.1530-1532(2006).

付録：ソースコード

A.1 表構造より要素を抽出するソースコード

```
import networkx as nx
import matplotlib.pyplot as plt
from langrid.clients import TranslationClient
from settings import lg_config

gnmt = TranslationClient('http://langrid.org/service_manager/wsdl/kyoto1.langrid:GoogleTranslateNMT',
                         lg_config['userid'], lg_config['password'])

#--- Cell クラスの定義
class Cell:
    def __init__(self, id, value, left, top, right = None, bottom = None):
        self.id = id
        self.value = value #  インスタンス変数に値を代入
        self.left = left
        self.top = top
        self.right = right
        self.bottom = bottom

    def set_right(self, right):
        # print('set_right:ID: {}, v:{}, left:{}, top:{}, right:{}, bottom:{}'.format(self.id, self.value, self.left, self.top, self.right, self.bottom))
        self.right = right

    def set_bottom(self, bottom):
        self.bottom = bottom
```

```
    # (row, col)の上部で直近のセルを探す関数
    def get_upper_cell(cells, row, col):
        for i in range(row):
            if row-1-i in bottom_index:
                for cell in bottom_index[row-1-i]:
                    if cell.left <= col and col <= cell.right:
                        return cell
        return None

    # (row, col)の左部で直近のセルを探す関数
    def get_lefter_cell(cells, row, col):
        for i in range(col):
            if col-1-i in right_index:
                for cell in right_index[col-1-i]:
                    if cell.top <= row and (cell.bottom == None or row <=
cell.bottom):
                        return cell
        return None

    def fixed(cell): #  セルのサイズが確定しているか判定
        if cell.right != None and cell.bottom != None:
            return True
        else:
            return False

    def add_node(cells, cell, graph):
        G.add_node(cell.id, cell=cell) # cell をノードとしてグラフに追加

        # upper エッジの追加
        upper = get_upper_cell(cells, cell.top, cell.left) # cell 左上端の上部の
直近のセルを取得
```

```
        if (upper.right > cell.right) or (upper.right == cell.right and upper.left < cell.left): # 上部のセルが cell を包含するとき
            G.add_edge(cell.id, upper.id, parent='upper') # 上部のセルに upper エッジを張る
        else: # 上部のセルが cell と同幅のとき
            for adj, attrs in G[upper.id].items(): # 上部のセルから upper エッジが張られた親ノードに upper エッジを張る
                if attrs['parent'] == 'upper':
                    G.add_edge(cell.id, adj, parent='upper')

        # lefter エッジの追加
        lefter = get_lefter_cell(cells, cell.top, cell.left) # cell 左上端の左部の直近のセルを取得
        if (lefter.bottom == None) or (lefter.bottom > cell.bottom) or (lefter.bottom == cell.bottom and lefter.top < cell.top): # 左部のセルが cell を包含するとき
            G.add_edge(cell.id, lefter.id, parent='lefter') # 左部のセルに lefter エッジを張る
        else: # 左部のセルが cell と同高さのとき
            for adj, attrs in G[lefter.id].items(): # 左部のセルから lefter エッジが張られた親ノードに lefter エッジを張る
                if attrs['parent'] == 'lefter':
                    G.add_edge(cell.id, adj, parent='lefter') # upper の左接親ノードをひく必要あり

    # 表構造の解釈①：左，上のセルから優先して空白を連結.
    # 表構造の解釈②：上辺，左辺で接するセルの幅や高さを越えない
    # 表構造の解釈③：上辺，左辺で接するセルの幅や高さの方が大きければ，そのセルを親ノードとする
    # 表構造の解釈④：上辺，左辺で接するセルの幅や高さの方が同じ場合，そのセルを兄弟ノードとし，親ノードを継承する
    # セルの右端の決め方：上接の右端か，右接の左端によって決まる
```

```
# セルの下端の決め方：左接の下端か，下接の上端によって決まる

#--- CSV ファイルから読み込み
# FILE_NAME = 'ctest.csv'
# FILE_NAME = '長期優良住宅　整理 2.csv'
# FILE_NAME = '車両系建設機械作業計画書　整理.csv'
# FILE_NAME = '慶弔事届 整理.csv'
FILE_NAME = '安全運転管理者等に関する届出書　整理.csv'
# FILE_NAME = 'プログラム依頼書　整理.csv'
fi = open(FILE_NAME, 'r', encoding = 'utf-8')
lines = fi.readlines()

# 検出したセルを格納するリスト．表枠外の最上部と最左部のセルをセット
cells = [Cell(1, None, 0, 0, len(lines[0].split(',')), 0), Cell(2, None, 0, 0, 0, len(lines))] # ID=1 のセルは最上部，ID=2 のセルは最左部

# セルの bottom/right のインデックス
bottom_index = {cells[0].bottom: [cells[0]], cells[1].bottom: [cells[1]]}
right_index = {cells[0].right: [cells[0]], cells[1].right: [cells[1]]}
left_index = {cells[0].left: [cells[0]], cells[1].left: [cells[1]]}

G = nx.DiGraph()
G.add_node(cells[0].id, cell=cells[0])
G.add_node(cells[1].id, cell=cells[1])

id = 2 # Node/Cell ID
row = 0 # 行カウンタ
for line in lines:
    row += 1
    line = line.rstrip()
    items = line.split(',') # 1 行を半角スペースで区切って items リストに
```

```
代入
        col = 0 # 列カウンタ
        for item in items:
            col += 1
            # print('row:{}, col:{}'.format(row, col))
            if item != '': # セルの値を発見
                # print('FIND:row:{}, col:{}'.format(row, col))
                id += 1
                cells.append(Cell(id, item, col, row)) # セルの左上端を確定
                left_index.setdefault(col, []).append(cells[-1]) # left インデックスに格納

                # 左辺が隣接するセルの右端を決定する
                if cells[-2].top == row and cells[-2].right == None:
                    cells[-2].set_right(col-1) # 一つ前（左接）のセルの右端を確定（解釈①と解釈②より）
                    right_index.setdefault(col-1, []).append(cells[-2]) # right インデックスに格納
                    if fixed(cells[-2]): # セルの四つ角（サイズ）が確定したら
                        add_node(cells, cells[-2], G) # セルをグラフに追加

                # 上辺が隣接するセルの下端を決定する
                for cell in cells: # 上接のセルの下端を確定（解釈①と解釈②より）
                    if cell.bottom == None and cell.top < row and cell.right >= col:
                        cell.set_bottom(row-1)
                        bottom_index.setdefault(row-1, []).append(cell)
                        if fixed(cell): # セルの四つ角（サイズ）が確定したら
                            add_node(cells, cell, G) # セルをグラフに追加
```

```
            # 走査中のセルの右端を決定する
            upper = get_upper_cell(cells, row, col)
            if upper != None and upper.right == col and cells[-1].right ==
None: # 上接のセルの右端と当該セルの左端が等しい
                cells[-1].set_right(col) # 当該セルの右端を確定（解釈②より）
                right_index.setdefault(col, []).append(cells[-1]) # rightインデ
ックスに格納
                if fixed(cells[-1]): # セルの四つ角（サイズ）が確定したら
                    add_node(cells, cells[-1], G) # セルをグラフに追加

            # 走査中のセルの下端を決定する
            lefter = get_lefter_cell(cells, row, col)
            if lefter.bottom == row: # 左接のセルの下端と当該セルの下端が
等しい
                if col in left_index:
                    for cell in left_index[col]:
                        if lefter.top <= cell.top and cell.top <= row and
cell.bottom == None:
                            cell.set_bottom(row)
                            bottom_index.setdefault(row, []).append(cell)
                            if fixed(cell): # セルの四つ角（サイズ）が確定
したら
                                add_node(cells, cell, G) # セルをグラフに
追加

    fi.close()

    #--- 検出したセルの一覧表示
    for cell in cells:
        print('ID:{}, v:{}, left:{}, top:{}, right:{}, bottom:{}'.format(cell.id,
cell.value, cell.left, cell.top, cell.right, cell.bottom))
```

```
    #--- 構築したグラフのエッジ一覧表示
    #print(G.edges())

    #--- 構築したグラフの視覚化
    #plt.subplot(121)
    #pos = nx.spring_layout(G, k=3.0)
    #nx.draw(G, with_labels=True, font_weight='bold', pos = pos)
    #plt.show()

    #--- 翻訳処理
    for i in range(3, id+1):
        source = G.nodes[i]['cell'].value.replace(' ', '').replace('　', '') # 翻訳対象のラベルを取得
        result = gnmt.translate('ja', 'en', source) # 翻訳対象のラベルの翻訳結果

        path = max(nx.all_simple_paths(G, i, [1, 2]), key=lambda x: len(x)) # 各ノードからの最長パスを取得

        context = '' # 付加する文脈用の変数
        for id in path[-2:0:-1]: # 最長パスの逆順
            context = context + G.nodes[id]['cell'].value + 'の'
        context = context.replace(' ', '').replace('　', '') # 半角・全角空白を除去
        if (context[0:-1] != ''):
            contextResult = gnmt.translate('ja', 'en', context[0:-1])
        else:
            contextResult = ''
        concat = context + source # 文脈とラベルを連結
        concatResult = gnmt.translate('ja', 'en', concat)

        print('ID: ' + str(i))
```

```
            print('SOURCE: '+ source)
            print('SOURCE_TRANS: '+ result)
            print('CONTEXT: '+ context[0:-1]) #  最後の「の」を除いた文字列を表
示
            print('CONTEXT_TRANS: '+ contextResult)
            print('CONCAT: '+ concat)
            print('CONCAT_TRANS: '+ concatResult)
        print('')
```

A.2 文脈を除去するソースコード

```
from langrid.clients import BindingNode, TranslationClient
from settings import lg_config

twtd_client =
TranslationClient('http://langrid.org/service_manager/wsdl/kyoto1.langrid:Go
ogleTranslateNMT',

lg_config['userid'], lg_config['password'])

rabel_source = ''
main_source = ''
sub_source = ''
#print(twtd_client.translate('ja', 'en', main_source))
#print(twtd_client.translate('ja', 'en', sub_source))

sum1_source = rabel_source + 'の' + main_source
sum2_source = rabel_source + 'の' + sub_source
#print(twtd_client.translate('ja', 'en', sum1_source))
#print(twtd_client.translate('ja', 'en', sum2_source))

#翻訳サービス呼び出し
trans1_source = twtd_client.translate('ja', 'en', sum1_source)
```

```
trans2_source = twtd_client.translate('ja', 'en', sum2_source)


def list_difference(list1, list2):
    result = list1.copy()
    for value in list2:
        if value in result:
            result.remove(value)

    return result

l1=trans1_source.split(' ')
l2=trans2_source.split(' ')
result = list_difference(l1, l2)

print(result)
```