卒業論文

表構造のセル間の関係を用いた翻訳

指導教官　　村上 陽平 准教授

立命館大学　情報理工学部
先端社会デザインコース　4 回生
2600190332-1

増田　悠人

2022 年度（秋学期）卒業研究 3（CH）
令和 5 年 1 月 31 日

表構造のセル間の関係を用いた翻訳

増田 悠人

内容梗概

近年，国際結婚の増加や，政府による留学生の増加方針などによって外国人移住者が増加している．移住時には役所などで申請書類への記入が多く求められるが，外国語に対応している書類は少なく，機械翻訳を用いて翻訳しながら記入することが多い．紙の申請書をスマートフォンのカメラで取り込み，その画像から文字認識を行なって母国語へ翻訳を行っている．ニューラル機械翻訳による翻訳精度の大幅な向上により，書類の文書の理解は可能になりつつある．しかしながら，表構造になっている記入欄のラベルは，ニューラル機械翻訳を用いたとしても，依然として誤訳が生じる．これは表のセル内のラベルは省略や分割などされた不完全な日本語で構成されているためである．例えば，生年月日の記入欄では，年，月，日に分割されており，「月」が「month」ではなく，「moon」と翻訳される．これは，「月」単体では「moon」と翻訳される対訳データの方が多いため，ニューラル機械翻訳は「moon」と誤訳している．

そこで，本研究では表構造に基づいて構築されたラベル間の関係を用いて，表の文脈を考慮した翻訳を行う．具体的には，表構造のセル間の親子関係や兄弟関係を用いて，ラベル間の関係ネットワークを構築し，関係ネットワーク上のパターンに応じた文脈を生成し，翻訳対象ラベルに文脈を付与して翻訳を行う．本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．

セル間の関係の形式化

申請書類には多様な書式が存在するため，表のラベルの内容や特定の表構造に依存せずに表構造を解釈する必要がある．さらに，解釈した構造から文脈を生成するために，その構造を計算処理可能な形式的表現でモデル化する必要がある．

関係に基づく文脈の付与

文脈を生成するために，構造の形式的表現から文脈となる文字列に変換する必要がある．そのために，構造の形式的表現と照合可能な文脈生成ルールを構造のパターンごとに作成する必要がある．

前者の課題に対しては，表から縦方向と横方向のセルの親子関係を抽出し，セルをノード，親子関係をエッジとするグラフ構造で表構造を表現する．まずは，表構造を抽出するために，画像状の表をcsv形式に変換する．csvファイルから，

セルの高さや幅が右隣や下のセルの高さや幅よりも大きいセルのペアを同定し，セル間を親子関係のエッジで連結して有向グラフを構築する．さらに，親子関係から兄弟関係も抽出する．

後者の課題に対しては，抽出できた関係のグラフに対して，条件を満たす部分グラフを発見し，その部分グラフのノード値をテンプレートに代入することで，文脈となる文字列を生成する．例えば，親のセルが「曜日」，このセルが「日，月，火」の場合，「親セルの子セル」と「子セルの子セル」のテンプレートを適用して，「曜日の日と月と火」という文脈を生成し，日や月が Sun や Moon と誤訳されるのを防ぐ．

提案手法の有用性を示すために，30 件の申請書類を用いて既存のニューラル機械翻訳との比較を行い，提案手法による改善を確認した．本研究の貢献は以下の通りである．

セル間の関係の形式化

表構造からセル間の親子関係及び兄弟関係を抽出し，有向グラフを生成するパーサーを構築した．30 件の申請書類に適用し，計 1482 個のノードと計 357 個のエッジから構築される 30 個の連結グラフを生成した．

関係に基づく文脈の付与

2 個の文脈生成ルールを定義した．本ルールを 30 件の申請書類から抽出された表構造に適用した結果，1482 ノードのなか，既存のニューラル機械翻訳の正確さにおいて改善率 0.61，改悪率 0.023 となり，流暢さにおいて改善率 0.7，改悪率 0.037 となった．

Translation Using Relationships between Cells in a Table Structure

Yuto Masuda

Abstract

In recent years, the number of foreign immigrants has increased due to the increase in international marriages and the government's policy to increase the number of foreign students. At the time of living in Japan, many application forms are required to be filled out at government offices, but there are few documents that are available in foreign languages, and machine translation is often used while filling them out. Significant improvement in translation accuracy by neural machine translation, it is becoming possible to understand written documents. However, even if neural machine translation is used, mistranslation still occurs in the labels of the entry fields that have a tabular structure. This is because the labels in the cells of the table consist of incomplete Japanese that has been omitted or split. For example, the date of birth is divided into year, month, and day, and ``月" is translated as ``moon" instead of ``month". This is because there are more bilingual data that translates "月" alone as "moon", so neural machine translation mistranslates "moon".

Therefore, in this research, we use the relationship between labels constructed based on the table structure to perform translation considering the context of the table. Specifically, we construct a relationship network between labels using parent-child relationships and sibling relationships between cells in a table structure, generate contexts according to the patterns on the relationship network, and assign contexts to labels to be translated. The following two issues should be addressed in order to implement this method.

Formalizing relationships between cells

Since there are various forms of application documents, it is necessary to interpret the table structure without depending on the content of the table label or the specific table structure. Furthermore, in order to generate context from the interpreted structure, we need to model the structure in a computable formal representation.

Giving Context Based on Relationships

To generate the context, we need to transform the formal representation of the structure into a contextual string. For that purpose, it is necessary to create a context generation rule that can be matched with the formal representation of the structure for each pattern of the structure.

For the former problem, we extract the parent-child relationship between vertical and horizontal cells from the table and express the table structure in a graph structure where the cell is the node, and the parent-child relationship is the edge. First, convert the table like image to csv format to extract the structure of the table. From the csv file, identify pairs of cells whose height and width are greater than the height and width of the right or bottom cell, and connect them with parent-child relationship edges to create a directed graph. Furthermore, the sibling relationship is also extracted from the parent-child relationship.

For the latter task, we find a subgraph that satisfies the conditions from the graphs of the extracted relations, and by substituting the node values of the subgraph into the template, we generate a character string that serves as a context. For example, if the parent cell is "day of the week" and the child cell is "Sunday, Monday, Tuesday", the templates "child cell of parent cell" and "child cell of child cell" are applied to Generates the context "月 and 火" to avoid mistranslation of Monday and Tuesday as Moon and Fire. In order to demonstrate the usefulness of the proposed method, we compared it with the existing neural machine translation using 30 application documents and confirmed the improvement by the proposed method. The contribution of this research is as follows.

Formalizing relationships between cells

We constructed a parser that extracts the parent-child and sibling relationships between cells from the table structure and generates a directed graph. We applied it to 30 application documents and generated 30 connected graphs constructed from 1275 nodes and 327 edges.

Giving Context Based on Relationships

We defined 2 context generation rules. As a result of applying this rule to the table structure extracted from 30 application documents, we confirmed that the proposed method improved 28 places out of 26 places where existing neural machine translation mistranslated.

表構造のセル間の関係を用いた翻訳

目次

第1章 はじめに

近年，外国人移住者が増加している．その理由として国際結婚数の増加や日本に来る留学者の増加などが挙げられる．その際に移住者が日本に来て初めに壁となるものは言語である．日本語を学んでくる移住者も多数いるが，日本語は文字種が多く漢字の読み方も多いため，日常会話を主に学んでくる者にとっては，日本の文書は読みづらいものである．特に役所の文書は日本人にとっても分かりづらいこともあり，記入が大変である．そこで，Google などが展開しているニューラル機械翻訳のサービスを用いるなどをして翻訳しながら記入にあたるのがほとんどだが，日本の書類は，本来単独では意味が変わってしまうものを単独で表に含まれていたりする．不完全な日本語で構成されていることが多いのが問題で，臨んだ翻訳結果が出てこない．例えば，業務に関する文書の中で曜日ごとに記入欄があるとき，左の枠に縦に「曜日」と記されているときに，右の枠に横に「月」から「日」が単独でそれぞれ記されているものを従来のニューラル機械翻訳を用いると，表で区切られているため「moon」や「sun」などのように翻訳されてしまう．

本研究の目的は，表によってそれぞれ区切られているセルを，関係性を用いて修飾して文脈を生成し翻訳させることで，本来記入すべき内容の翻訳結果を導出させることである．表中で幅の広いセルが幅の狭いセルを複数個囲んでいる場合，幅の広いセルと，翻訳対象のセル以外の幅の狭いセルが翻訳対象のセルを修飾すると仮定し，繋ぎ合わせて翻訳対象のセルに文脈として付与することで，翻訳精度が向上することを検証する．

その場合には，以下の二点が重要となる．

セル間の関係の形式化

表のラベルの内容や特定の表構造に依存せずに表構造を解釈する必要がある．さらに，その構造から文脈を生成するために，計算処理可能な形式的表現でモデル化する必要がある．

関係に基づく文脈の付与

文脈を生成するために，形式的表現から文脈となる文字列に変換する必要がある．そのために，文脈生成ルールを構造のパターンごとに作成する必要がある．

以下，本論文では，２章においてニューラル機械翻訳と，文書翻訳の関連研究

を説明する．次に３章では，本研究のシステムの概要を説明する．続いて，４章では本研究で使用する表構造の定義と，表からセル間の関係を抽出し，形式的表現でモデル化する方法を説明する．５章では付与する文脈の生成ルールについて説明する．６章で実験データの内容と翻訳の結果を表し，７章にて考察を述べる．最後に，８章で今後の課題と展望を述べて結論とする．

第2章 機械翻訳

この章では従来のニューラル機械翻訳を説明し，既存の文書翻訳の研究と表構造を用いた手法について説明する．

2.1 ニューラル機械翻訳

ニューラル機械翻訳とは，人間の脳神経回路の仕組みをもとに作られた人工的なニューラルネットワークを使用して機械が自ら学習し，原文に対して尤度の高い翻訳結果を選び出す機械翻訳の一種である．あらかじめルールを決めて原文を翻訳するルールベース翻訳とは翻訳結果の正確さや流暢さが大きく異なる．その仕組みは，エンコーダーにより原文を単語ベクトルの集まりとして整理され，デコードと単語ベクトルに基づいて翻訳結果への置換がされる．その結果，前後の単語などが関連付いた正確な翻訳を生成することができる．また，実行時間が短いことや，さらに学習データを学ばせることで正確性を上げる拡張性などがある．しかし，単語によってベクトルが異なるため一部を同意語などに変えるだけで翻訳結果が異なる場合や，書類や小説などの省略された単語や使われているデータが少ない単語が用いられているものは翻訳が難しいという欠点がある．

2.2 文書翻訳

機械翻訳の精度の向上によりあらゆる人が気軽に翻訳機にかけ，移住や旅行などに便利なものとなっている．しかし，買い物や話すことは従来の機械翻訳でも十分な正確さを持っているが，書類などの普段使わない単語や組み合わせによって正確に翻訳することができず困ることがある．そこで，そういった従来のものを利用できないものに対して，それに特化させた翻訳機の研究が多々ある．その中でこの章では，本研究に関連する既存研究を紹介する．

まず，文書識別と情報抽出をするためのテンプレートを作るという研究である．この研究は，タイトルやレイアウトを含む文書の構造をドキュメントから自動で識別して抽出することができるとされています．確かに文書の構造が正確に取れるのであれば，表構造を読み取り本研究に取り入れることもできたが，事前に文書の種類を登録しているときという条件があるため，表構造の種類の多

さに対応できない [1].

次に，表構造を認識して目的の表構造を含む文書を検索する研究である．手法としては，対象となる文書のテキストの位置や表構造の接点情報を解析し，雛形文書との類似度で検索するという方法である．キーワードメタデータを文書から抽出するために，セル内のテキスト情報を形態素解析にかけ，tf-idf 法によってスコア付けをしてキーワードを抽出する．表構造の抽出は，セルの接点情報を取得して接点をパターン化させたものを行列化し抽出する．この方法では表構造を正しく抽出し，セルごとのテキスト内容も確認することが可能だが本研究にあたって，翻訳対象は行政文書等の表構造外に文字が含まれているものがほとんどなため，同じ仕組みを使うことはできない[2].

最後に，日常生活では使われることがない専門用語などの対訳データを効率的に作る研究である．機械翻訳は，大量のデータを対訳データに登録することや，学習させることによって翻訳結果を選出してくれるものである．そのため，合成名詞や，データがない未知語などが原文内に含まれている場合誤訳となる．したがって，それらを専門用語として対応させる対訳データを作る必要がある．まず日本語と英語で書かれた文書からそれぞれ対象となるユニットを抽出する．2 つのユニットの関係を推定し，日本語ユニットから専門用語をリスト化する．2 つのユニットの関係と同意なものを既存の対訳辞書から探して評価し，評価したものの中から最も適当なものを選定する．この研究の結果，合成名詞と未知語はそれぞれ 70%と 54%の効率の向上がみられたが，表構造の文書によく用いられる略語に対応することが難しく，さらに 2 言語の同じ書類の数も少ないためデータを集めるのが困難である．したがって，表構造の翻訳に適用することは難しい [3].

第3章 文脈を考慮した文書翻訳システム

3.1 システム概要

ここでは，本システムの流れを説明する．本研究では全ての書類に対応するものではなく，単純な表の作りになっていることと，１つのセル内に含まれたものを１つの言葉として判断したものに限られる．用いる表のデータは，表構造が含まれた PDF ファイルから表を抽出し，CSV ファイル形式に変換して用いる．

本システムの構成図と翻訳の流れを図に示す．

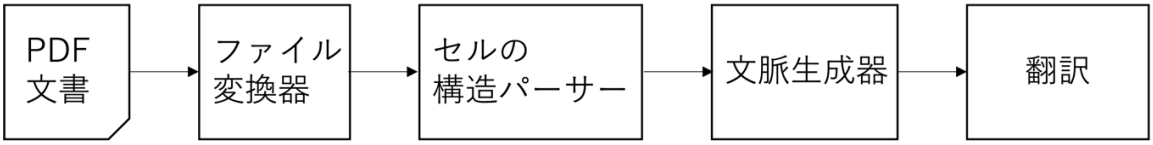

図 1　システム構成図

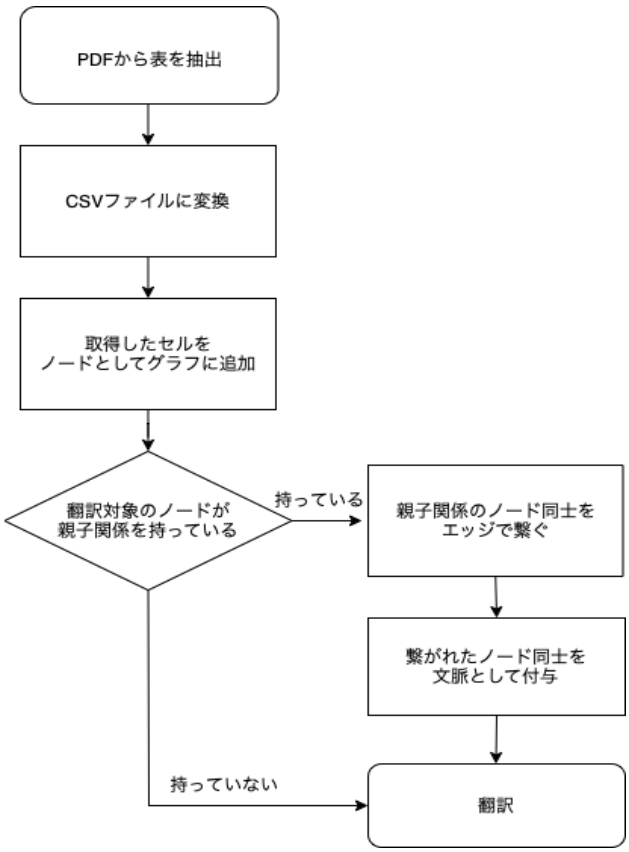

図 2　システムのフローチャート

以下に図 2 の詳しい説明を過程に分けて行う．

1． 表構造が含まれた PDF ファイルを Python のモジュールを用いて Excel ファイル形式に変換する．そして Excel ファイルのデータを CSV ファイル形式に変換させる．
2． CSV ファイルから表番号などの不要な情報を取り除き，セル内の単語や位置情報をリスト化し，グラフに追加する．
3． 表のデータからセル間の関係をもとに，親子関係を持つセル同士をノードとしてエッジで繋ぐ．
4． 接続した関係性にルールを定め，翻訳対象のセルに文脈を付与する．
5． 翻訳する．

過程 2，3 の用いる表構造の定義及び，セル間の関係をもとにしたグラフは 4 章，過程 4 の付与する文脈の生成ルールは 5 章にて詳しく説明する．

3.2 画像から表の変換器

PDF ファイル形式の書類データから Excel ファイル形式に変換する際に Python の Camelot モジュールを用いる．Camelot とは Python の OSS（オープンソースソフトウェア）であり，PDF から表を抽出し DataFrame 形式（二次元の大きさを格納できる表形式データ）に変換できるものである．抽出した DataFrame 形式のデータを Excel ファイル形式に変換することで CSV ファイル形式への変換を行う．CSV ファイルから，翻訳に不必要である行列番号は手動で削除しておく．

3.3 セルの構造パーサー

まず，CSV ファイルを解析して読み取った情報を有向グラフに変換し，セル間の関係を定める．次に CSV ファイルから，1 つずつテキストデータと，「,」で区切られた場所をもとに位置情報を取り出しリスト化する．リスト化されたものから順にグラフに追加していき，親子関係に当たるものを有向辺で繋いでいく．全て追加し終えると，セル間の関係を表した有向グラフが出来上がる．

3.4 文脈生成器

翻訳対象ラベルから最長パスを探索し，それを親として翻訳対象まで子へと繋げていき，1 文の文章として生成する．その際にラベル同士を定めたルールに

従って，最終的に翻訳する文脈を生成する．

第4章 セル間の関係の抽出

4.1 表構造の定義

本研究では全ての表に対応しているわけではなく，構造が単純なものを対象とする．

単純なものとは，以下の定義とする．

1. あるセルの幅と高さは，上辺や左辺で接するセルの幅や高さを超えることはない．
2. あるセルの幅と高さより，上辺や左辺で接するセルの幅や高さがあるセルを超えるとき，あるセルは上辺や左辺で接するセルの子ノードとする．
3. あるセルが，接しているセルと同じ幅と高さの場合，二つのセルは兄弟ノードとする．

図 3 は表構造の中の説明であり，本研究で使用する親子関係と兄弟関係を表した図である．

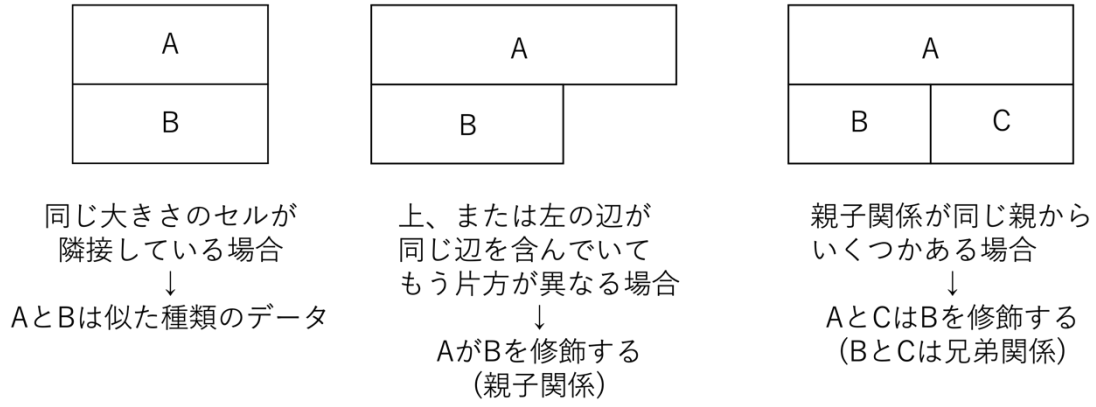

図 3　セル間の関係

4.2 セル間の関係の形式化

対象セルの翻訳の精度を上げるため，セル間の関係を同定することが本研究では重要な点である．まず，セル間の関係には前ページで定義した親子関係を用いて形式化する．親子関係になっているセル同士は，子を親のラベルが修飾する．兄弟関係は同じ親の子同士であるため，対象の子ラベルの翻訳を補う役割にあ

たる．この親子関係の形式化をするにあたって，グラフを用いる．グラフの作り方は，セルの構造パーサーを使って順に作っていく．

以下の図 4 ではグラフの生成方法を示す．

まず CSV ファイルから単語ノードと位置情報を取得しリスト化する．次に

リスト化されたノードを順にグラフ化していく．その際に親子関係を持つノード同士をエッジで繋ぎ，有向グラフを生成する．有向グラフでモデル化することによってセル間の関係を明らかにし，用いる関係を表す．表の定義で挙げた通り，全てのノードをエッジで繋ぐわけではない．以下の図 6 は，親子関係になっているもののみを，子が親に向けた有向辺で視覚化したグラフ G である．

対象セルをv_1とした時のグラフ G は，以下の式で表せられる．

$G = \{V, E\}$

$V = \{v_1, v_2, v_3, v_4\}$

$E = \{(v_2, v_1), (v_3, v_1), (v_4, v_3)\}$

$V_j = \{v_i | (v_i, v_j) \in E\}$　親ノードv_jの子集合（兄弟）

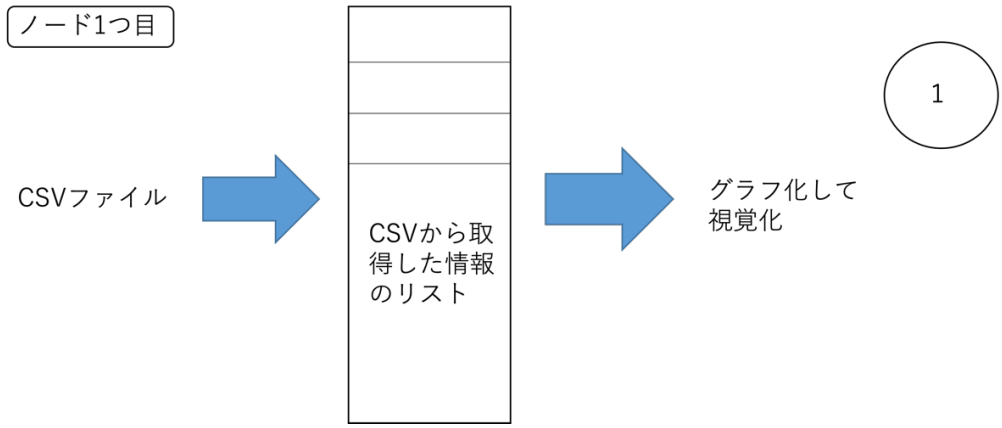

図 4　生成方法　手順 1

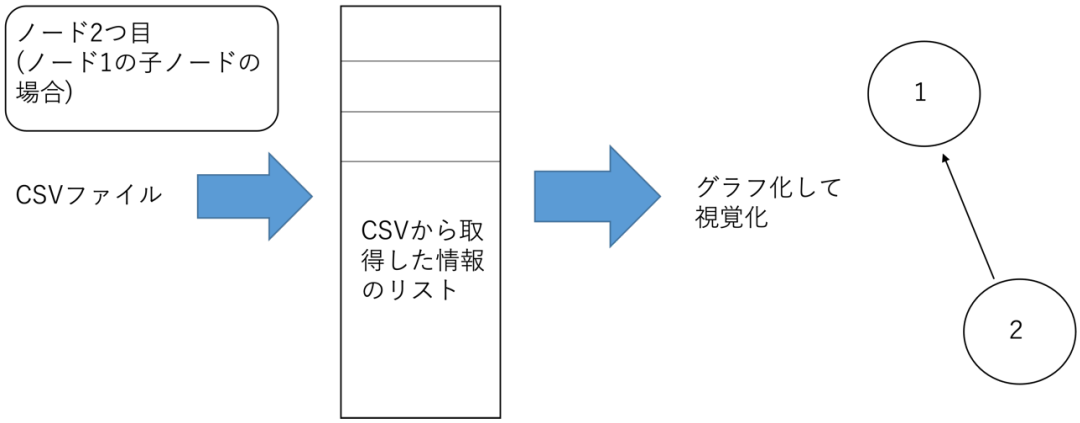

図 5　生成方法　手順 2

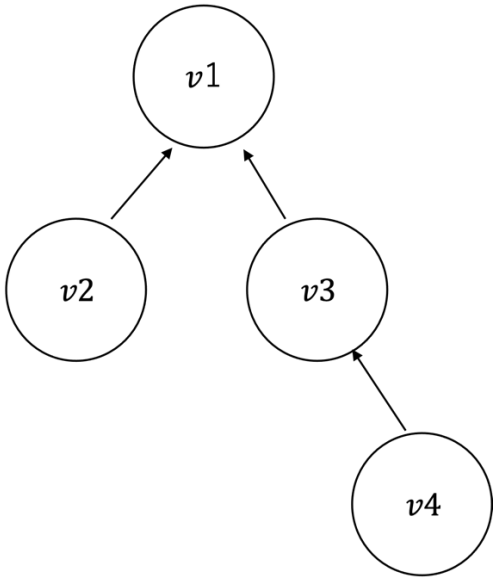

図 6　セル間の関係モデル

第5章 文脈の生成

5.1 文脈生成ルール

本研究では二つのルールを設けて文脈を生成する．まず一つ目は，親子関係のラベルを「の」で接続し，文脈を付与する．次にもう一つが，兄弟関係にあたるラベルを「と」で接続し，文脈を付与する．この二つのルールによって，定義した単純な表構造であれば翻訳精度の向上が期待できると考えられる．親子関係と兄弟関係のどちらも持たないものは，付与する文脈はなく単独で翻訳するものとする．図 7 は，あるセルを対象セルとしたときの親子関係を説明するものである．「v_1」は「v_2」と「v_3」を子として持っており，親子関係である．さらに，「v_3」は「v_4」を子として持っている．付与された文脈を含めて翻訳する文章は「v_1」の「v_2」や，「v_2」と「v_3」，「v_3」の「v_4」という文になる．

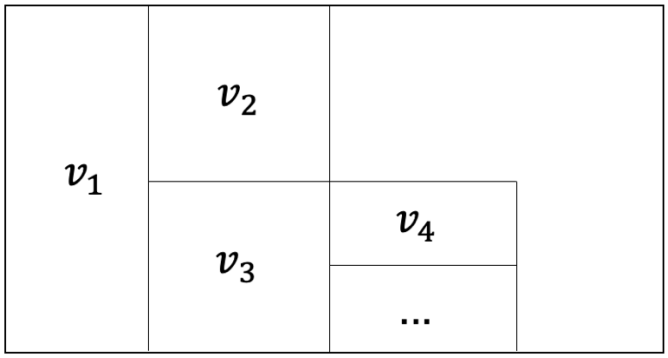

図 7　表構造の例

5.2 ルールの合成

さらに翻訳精度を上げるためルールの合成も行う．今回のルール二つを合成し，親子＋兄弟で翻訳することによって翻訳精度の向上が考えられる．ルールを合成するにあたって注意すべき点は，親の翻訳を複数回しないことである．例えば，図 7 と同じ表構造のセル間の関係であった場合，「v_1」の「v_2」と「v_1」の「v_3」などのように親子関係を兄弟関係のルールで接続してしまうと，親を複数回翻訳することになる．これでは子の単語によって親「v_1」の翻訳結果にズレが生じてしまう可能性がある．したがって，親に該当するものは文脈中に一度だけ

翻訳して翻訳のブレが起きないようにする. 図 7 で例を挙げると,「v_1」の(「v_2」と(「v_3」の「v_4」))になる.

第6章 実験

6.1 実験データ

本研究の評価方法は，Google ドキュメント翻訳を用いて翻訳したものと，本研究の提案手法を用いて翻訳したものを比較する方法である．まず，Web 上から Excel ファイルを取得した．このとき，Excel ファイルから提案手法の妨げになるような表外の文字は削除し，表構造だけを読みよるようにした．取得したデータは 30 件で，これを分類に分けて表構造によって提案手法の影響の違いも確認することとした．分類は，親子関係などの関係が無いもの，親子関係はあるが兄弟関係が無いもの，兄弟関係を含むもの，親子関係の子がさらに子を持っているもの，の 4 つの分類とする [表 1].

表 1　実験データ件数

	親子関係を持たない	親子関係のみを持つ	兄弟関係を持つ	親子関係の子が親子関係を持つ
件数	3	2	20	5

分類したものをそれぞれ翻訳し，翻訳結果の正確さと流暢さの 2 点で評価する．実験の流れは，まず取得した Excel ファイルを Google ドキュメント翻訳で翻訳し，誤訳の有無を記録する．次にその Excel ファイルを CSV ファイル形式に変換して提案手法のプログラムを実行させ翻訳結果を出力する．翻訳結果から誤訳の有無と Google ドキュメント翻訳で誤訳していた部分の比較，生成した文脈の流暢さを記録する．一例として 1 つ書類を挙げ，それについての実験結果を述べつつ, 最後に全体での割合などの評価をする. 先行研究で用いられていた，自動車の車種に対しての誤訳を改善しようという書類があり，その考えを用いてさらに誤訳の生じる可能性が高そうな表を用意した[4].

以下の図 8 は親子関係の子が親子関係を持ち，兄弟関係を多く含む文書の一部である．Google ドキュメント翻訳の翻訳結果が図 9 であり，提案手法の翻訳

結果が図 10 である.

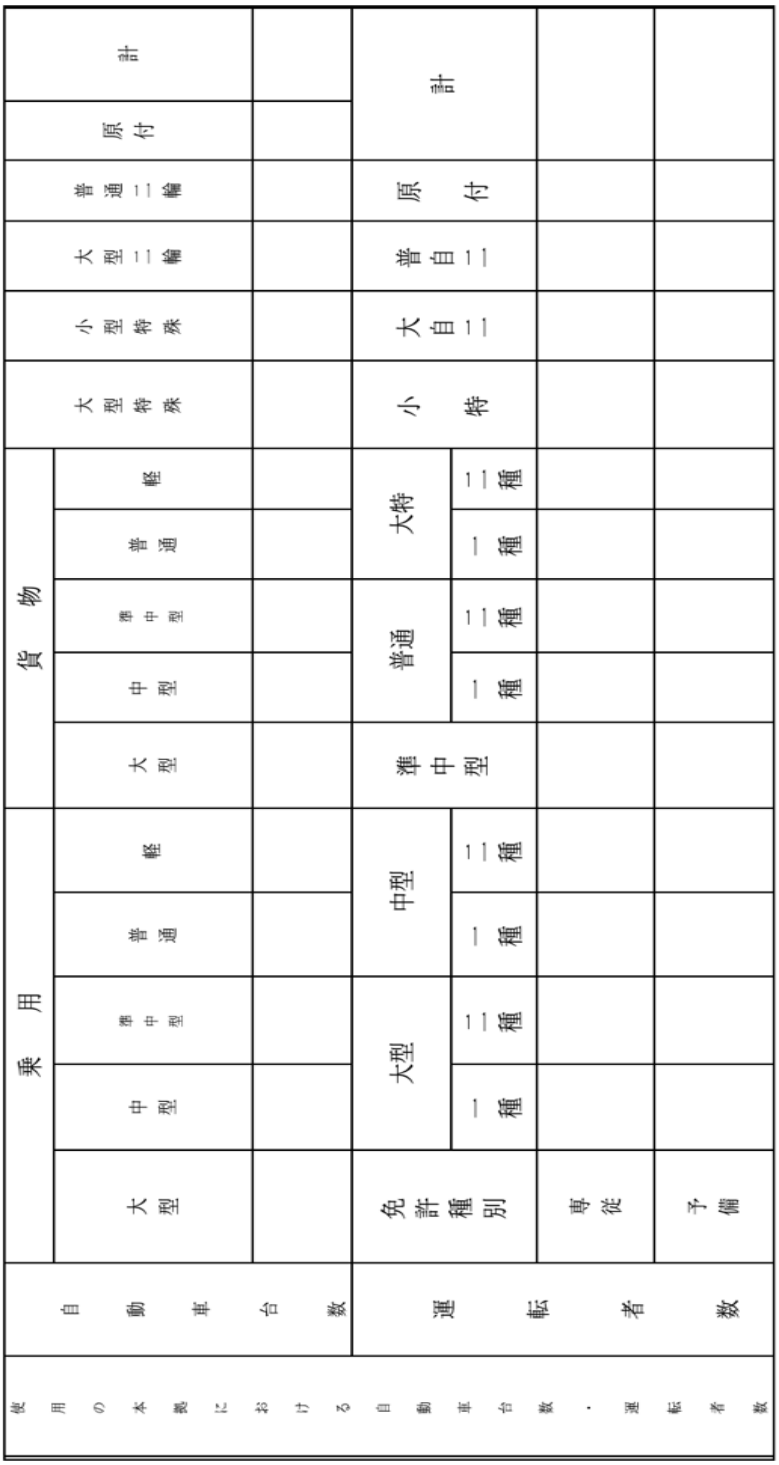

図 8. 実験データ例

Number of motor vehicle drivers at the base of use	number of cars	riding					cargo					large special	small special	large two wheels	Ordinary two wheels	scooter s	Total
		large	medium size	Semi-medium size	usually	Light	large	medium size	Semi-medium size	usually	light						
	number of drivers	License type	large		medium size		Semi-medium size	usually		big special		small special	self =	Universal =	scooters	Total	
			kind of	Two types	kind of	Two types		kind of	Two types	kind of	Two types						
		full-time															
		spare															

図 9．Google ドキュメント翻訳による図 8 の翻訳結果

Total Number of Vehicles /Drivers																													
The number of motor vehicles			passenger						freight									large special	small special	large motorcycles	Ordinary motorcycle	mopeds	Total							
		Large	Medium-size	Semi-medium size	Average	Minimum		Large	Medium-size	Semi-medium size	Average	Minimum																	
The number of drivers	License classification	Large								Semi-medium size	Average			big special		small special	large car 2s	general car 2s	mopeds	Total								
		A type	Two types		A type	Two types					A type	Two types		A type	Two types														
	Dedicated to																												
	Reserve																												

図 10. 提案手法による図 8 の翻訳結果

実験データの例として用いたのは警察庁の配布している「安全運転管理者に関する届出書」に含まれる表の一部である．自動車の車種を示す言葉がほとんどの割合を占める書類だが，それぞれに車という情報は含まれておらず，Google ドキュメント翻訳では「乗用」の翻訳結果は「riding」となっている．しかし本来省略されていない場合，「乗用」は「乗用車」であることから正しい翻訳は「passenger car」とならなければならない．提案手法による結果の方では，左の「仕様の本拠における自動車台数・運転者数」が親である「自動車台数」の子であるため，乗用車であると判断することができ「passenger」と出力することができた．他にも，顕著な部分である「大自二」と「小自二」の翻訳結果を見ると，Google ドキュメント翻訳では省略された言葉がわからず，「二種」では「Two」と判断できた「二」を「＝（イコール）」と出力している．この点に関して提案手法では大型自動二輪，普通自動二輪とまで判断できなかったものの，「Large car 2s」，「general car 2s」といったように自動車関係の言葉であると判断することができた．これらはセルの関係を取得して不足している情報を補った結果と言える．このような翻訳結果の評価を正確さと流暢さによって行う．正確さと流暢さは，一般的な機械翻訳の 5 段階で評価する．正確さの判断は，評価 1 と 2 を誤訳とする．流暢さは一般的にスピーキングで用いられる評価方法であるが，文脈の流れは翻訳結果に現れるので，単語の言い換えなどに着目して評価する．[5] 流暢さがないという判断は評価 1 と 2 の値とする． 評価した文書に対し，翻訳が有用でないものを 1 とし，有用であるものを 5 とする．本研究では，書類ごとの有用さと，セルだけに着目する 2 種類の判定で行う

6.2 翻訳の正確さ

本研究の正確さ（Accuracy）は，原文から正しい意味を反映できているかを判定基準とする．翻訳に誤訳が含まれていると，他の翻訳部分への信頼性も下がると考えるため 2 つの誤訳に対して 1 段階の評価を下げる．したがって，8 つ誤訳が生じた時点で対象の文書の正確さは 1 となり，その文書に対する翻訳は有用で無いと判断する．

図 8 のデータを例にすると，Google ドキュメント翻訳の正確さは 2 とし，提案手法による正確さは 4 とする．前者は「軽」，「二種」，「大型二輪」，「普通二輪」，「大自二」，「小自二」の 6 つ，後者は「二種」，「大自二」，「小自二」の 3 つが評価対象である．

取得したデータ 30 件を正確さで評価したところ Google ドキュメント翻訳の評価の平均は約 4.06 となり，提案手法の評価の平均は約 4.33 となった．結果として提案手法の方が正確に翻訳できるという結果を得た．

6.3 翻訳の流暢さ

本研究での流暢さ（Fluency）は，出力した翻訳の文脈が理解しやすいかを判定基準とする．初期値を 3 とし，流暢さがない翻訳結果 2 つにつき 1 段階の評価を下げる．流暢さがある翻訳結果 1 つにつき 1 段階の評価を上げる．

図 8 のデータを例にすると，Google ドキュメント翻訳の流暢さは 1 とし，提案手法による流暢さは 3 とする．前者は「専従」，「一種」，「普通」，「乗用」の 4 つ，後者は「一種」，「普通」の 2 つが減点対象で，「原付」が加点対象である．Google ドキュメント翻訳の評価の平均は約 2.46 となり，提案手法の評価の平均は約 3.16 となった．結果として提案手法の方が流暢に翻訳できるという結果を得た．

6.4 評価結果

30 件のデータを上記の方法で評価したところ，Google ドキュメント翻訳の平均評価は 5.63，提案手法の翻訳の平均評価は 8.4 となった．親子関係などのセルの関係によって翻訳結果に違いが出ると考えたため 6.1 で挙げた 4 つの分類に分けて平均評価を算出した．

以下の表 2 はその結果である．

表 2　表構造ごとの平均評価

平均評価	親子関係を持たない	親子関係のみを持つ	兄弟関係を持つ	親子関係の子が親子関係を持つ
提案手法	9	8	9.2	6.6
Google ドキュメント翻訳	7	8	5.4	6.4

図から分かるように兄弟関係を持つときの表に対して最も高く，順に親子関

係を持たない，親子関係を持つ，親子関係の子が親子関係を持つという結果になった．

上記の結果を得る中で評価された全てのセルのうち，既存のニューラル機械翻訳が誤訳したのは 116 個であり，そのうち 71 個の改善が確認できた．したがって，正確さの改善率は 0.61 になる[表 3]．有用さにおいては，既存のニューラル機械翻訳が流暢さのない結果になったのは 192 個であり，そのうち 136 個の改善が確認できた．したがって，流暢さの改善率は 0.70 となる[表 4]．

表 3　セルごとの翻訳の正確さ

		提案手法		計
		誤訳	正しい訳	
Google翻訳	誤訳	45件	71件	116件
	正しい訳	32件	1334件	1366件
計		77件	1405件	1482件

表 4　セルごとの翻訳の流暢さ

		提案手法		計
		流暢さなし	流暢さあり	
Google翻訳	流暢さなし	56件	136件	192件
	流暢さあり	49件	1241件	1290件
計		105件	1377件	1482件

第7章 考察

提案手法による翻訳に有意差があった理由として，周囲のセルの情報をとることで誤訳を減らすというアプローチが成功したと考えられる．しかし，取得するデータによっては，表構造を抽出することができない場合や Excel ファイルに変換した際にセルの結合などが失敗することがあった．これは本研究で用いたプログラムでは対応できないファイルである．本研究で定義した単純な表は，日本中の文書のごく一部であり，全ての表構造の文書に対応するのは現時点では不可である．対応できなかった原因は，Excel ファイルに変換する際の問題点としてセルの形と，セル内部の文字の非統一性であると考える．セルの形は必ず正方形や長方形というわけではなく，凹や凸の形をしたセルがいくつもの書類に確認できた．CSV ファイル形式にした際，凹や凸の形を表現できないことが問題点であると考える．CSV ファイル以外に表構造の情報を表せるもので凹凸のセル情報も読み取れる形式があれば可能であると考える．セル内部の文字の非統一性とは，縦書き横書きが混在していること，文章量が多くセル内に文字が満たされ縦書きか横書きかが判断できないことだと考えられる．縦書きか横書きかはプログラムによって判断することは可能だが，長さの同じ文章塊を縦書きか横書きかを判断することは困難である．例えば 21 文字で書かれた文章が 7 行 3 列のように書かれていると，言葉が途切れる可能性が非常に高い．したがって，この問題を解決するには縦書きか横書きのどちらかのみの書類作成ルール，またはセル内の文字の密度の基準などの指定が日本の文書に必要になる．

本研究の内容として文脈生成ルールを用いて翻訳を行ったが，これを改善することによってさらに翻訳の精度が向上すると考える．例えば，本研究では親子関係と子同士の兄弟関係についてルールを作り評価したが，兄弟関係が特に多い場合，全ての意味がつながっているわけではなく部分的に一致していることがある．その問題を解決するために，一定以上の親子関係や兄弟関係があった場合，元の親との距離や兄弟関係の距離によって文脈の付与する長さを変えるルールなどを作ると，不要な情報が混ざらないように翻訳文を生成することが可能と考える．今回の結果の中で誤訳が出た原因として，最初の親の誤訳が最も多かった．そこで，翻訳精度の向上が得られる方法として，文書名を全ての親とすることで文書名と翻訳内容に不一致が出る可能性を消せると考える．本研究の親子関係のルールに則った文脈の付与を行うと一部に不要な情報となる可能性

があるため，表構造の始祖のような位置に結びつける．例えば，実験データ図 7 の文書名を情報として付与することができれば，車という文字が含まれていないものも車と結びつくことで翻訳結果が正確になると考えられる．名前や生年月日などには不要な情報となる場合が多いため，表構造にほぼ確実に現れると思われるものは対訳データで固定化しておくのが良いと考える．

第8章 おわりに

近年，国際結婚数の増加や日本に来る留学者の増加などにより，外国人移住者が増加している．その際に移住者が日本に来て初めに壁となるものは言語である．日本語を学んでくる移住者も多数いるが，日本語は文字種が多く省略された言葉も多いため，日常会話を主に学んでくる者にとっては，日本の文書は読みづらいものである．通常，Google などが展開しているニューラル機械翻訳のサービスを用いるなどをして翻訳しながら記入にあたるが，日本の書類は本来単独では意味が変わってしまうものを単独のままや省略して表に含まれていたりするため，文書に特化した翻訳システムの研究をおこなった．

本研究は，表によってそれぞれ区切られているセルを，関係性を用いて修飾して文脈を生成し翻訳させることで，本来記入すべき内容の翻訳結果を導出させるアプローチでおこなった．

表中で幅の広いセルが幅の狭いセルを複数個囲んでいる場合，幅の広いセルと，翻訳対象のセル以外の幅の狭いセルが翻訳対象のセルを修飾すると仮定し，繋ぎ合わせて翻訳対象のセルに文脈として付与することで，翻訳精度が向上することを検証した．

本研究の貢献は以下の二点である．

セル間の関係の形式化

表のラベルの内容や特定の表構造に依存せずに表構造を解釈する必要があるため CSV ファイルから表構造を抽出するプログラムの実装をおこなった．

さらに，その構造から文脈を生成するために，計算処理可能な形式的表現でモデル化することができた．

関係に基づく文脈の付与

文脈を生成するために，形式的表現から文脈となる文字列に変換する必要があり，文脈生成ルールを構造のパターンごとに作成した．また，ルールをもとに翻訳することに成功した．

本研究で提案した文脈生成ルールは 2 つであり，まだ誤訳が生じる文書が多く見られた．書類を多く確認することで別の関係性を見つけることができれば，さらなる翻訳精度の向上が可能だと考える．また，本研究では上手く出力できなかった書類に対しての課題点として表構造の抽出方法が挙げられる．CSV ファイルから情報を取得し表構造を抽出したが，セル間の関係以外に文書から得ら

れる情報があれば，それらを用いることで翻訳を補い誤訳を減らせると考える．

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准教授に深謝申し上げます．また，普段からお世話になっている社会知能研究室の皆様に感謝の意を表します．

参考文献

[1] Kochi. T, Saitoh. T.: User-defined template for identifying document type and extracting information from documents, *IEEE*. Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR Cat. No.PR00318 (1999).

[2] 岡田伊策，齋藤稔，大和裕幸，稗方和夫，三浦慎也: 表構造解析とキーワード抽出で付与したメタデータを複合的に用いた表形式文書検索システムの開発，人工知能学会第二種研究会資料，2012巻，KST-16号，pp.02- (2012).

[3] 熊野明，平川秀樹: 対訳文書からの機械翻訳専門用語辞書作成，情報処理学会論文誌, Vol35 No.11，pp.2283 (1994).

[4] 玉置晴菜: 表構造に基づく文書の翻訳，立命館大学情報理工学部情報コミュニケーション学科卒業論文 (2019).

[5] 藤森千尋: スピーチプロダクションの測定方法：正確さ、流暢さ、複雑さ，関東甲信越英語教育学会研究紀要，18巻 pp.41-52 (2004).

付録

A.1 セル間の関係抽出プログラム

```
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:GoogleT
ranslateNMT',
                        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 = 'simple_table.csv'
# FILE_NAME = '入学願書２.csv'
# FILE_NAME = '営業証明書交付申請書１.csv'
# FILE_NAME = '営業証明書交付申請書２.csv'

#--- 実験に用いた CSV ファイル
# FILE_NAME = '安全運転管理者に関する届出書.csv'
# FILE_NAME = '住所変更届.csv'
# FILE_NAME = '身元保証人変更届.csv'
# FILE_NAME = 'キャンプ申込書.csv'
# FILE_NAME = 'ネオン管灯設備設置届.csv'
# FILE_NAME = '育児短時間勤務申請書.csv'
# FILE_NAME = '家屋貸付等申告書.csv'
# FILE_NAME = '家屋届出書.csv'
# FILE_NAME = '家屋非課税適用申告書.csv'
# FILE_NAME = '危険作業開始の届出書.csv'
# FILE_NAME = '危険物事故発生届出書.csv'
# FILE_NAME = '給与所得者異動届出書.csv'
# FILE_NAME = '軽自動車税申告書.csv'
# FILE_NAME = '指定居宅支援事業者指定申告書.csv'
# FILE_NAME = '事業所税更正の請求書.csv'
# FILE_NAME = '事業所税申告書.csv'
# FILE_NAME = '時間外勤務届.csv'
# FILE_NAME = '自宅療養証明書発行申請書.csv'
# FILE_NAME = '住宅用地.csv'
```

```
# FILE_NAME = '償却資産申告書.csv'
# FILE_NAME = '消火器着工届出書.csv'
# FILE_NAME = '水張検査申請書.csv'
# FILE_NAME = '税務証明交付申請書.csv'
# FILE_NAME = '駐車場使用申請書.csv'
# FILE_NAME = '土地非課税適用申告書.csv'
# FILE_NAME = '特別徴収切替届出書.csv'
# FILE_NAME = '避難はしご着工届出書.csv'
# FILE_NAME = '非常警報設備・器具着工届出書.csv'
# FILE_NAME = '稟議書1.csv'
# FILE_NAME = '稟議書2.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())

#--- イテレータの中身の長さを返す関数
def iterator_length(iterator):
```

```
    item_list = []
    for item in iterator:
        item_list.append(item)
    return len(item_list)

#--- 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)
```

A.2 文脈生成器及び出力

```
#--- 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
    #print('aaaaaaaaaaaaaaa{}'.format(source))

    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
    print("ID:{}, path:{}".format(i, path))

    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'
    parents = next(G.successors(i))

    if iterator_length(G.predecessors(i)) > 1:
        for kyodai in G.predecessors(parents):
            if kyodai != i:
```

```
                context = context + G.nodes[kyodai]['cell'].value + 'と'

    print(context)
    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('')
    for value in G.predecessors(i):
        print('ID:{}の子ノード:{}'.format(str(i), value))
plt.show()
```