特別研究報告書

文脈情報付き対訳辞書を用いた訳語選択

指導教員　　石田 亨 教授

京都大学工学部情報学科

中川 正博

平成24年2月2日

文脈情報付き対訳辞書を用いた訳語選択

中川 正博

内容梗概

インターネットユーザにはさまざまな言語や文化をもつ人々が存在しており，異なる言語を使う人々のコラボレーションにおいて，機械翻訳サービスは有用なサービスとなっている．

しかし，機械翻訳によって特定分野の専門的な文書を翻訳するとき，一般的な辞書のみを用いた翻訳では，その分野に特有の専門的な文脈に応じた翻訳ができない．この問題を解決する方法として，専門対訳辞書（以下，対訳辞書）を用いて専門用語を含む文を翻訳する機能を高める方法がある．対訳辞書とは，専門用語を翻訳元言語と翻訳先言語の対訳として登録しておくものである．対訳辞書を用いることによって，特定分野の文書に特化した機械翻訳を実現できる．しかし対訳辞書を用いた翻訳の際，対訳辞書中に同じ見出し語で異なる意味をもつ単語がある場合に適切な訳語の置き換えがされないことがある．

本研究では，このような対訳辞書内の同じ見出し語で異なる意味をもつ単語に対する訳語選択を支援するため，対訳辞書に文脈情報を付加することで，訳語候補に順位をつけて適切な訳語を選択できるようにする手法を提案する．文脈情報とは，既存の文書の文脈を考慮したときの単語間の意味的な結びつきの強さを表す評価尺度のことである．

対訳辞書内の同じ見出し語で異なる意味をもつ単語を適切に機械翻訳するために，文脈情報付き対訳辞書を作成し，それに基づき訳語選択する上で，以下の 3 つの課題がある．

1) 対訳辞書に付加するための文脈情報の内容選択
   対訳辞書に文脈情報を付加する上で考慮すべき点として，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対し適切に訳語選択するために，文脈情報としてどのような情報を用いるのかという問題がある．
2) 文脈情報を付加する際の計算時間
   翻訳をする際，同じ見出し語で異なる意味をもつ単語が翻訳元文書に登場するたびに文脈情報の計算を行うと，翻訳元文書の長さが長い場合に計算時間が増加する．
3) 訳語選択時に文脈情報を用いる手法

単に対訳辞書に文脈情報が存在するのみでは適切な訳語選択ができるようにはならない．そこで訳語選択を行う際に，文脈情報付き対訳辞書の文脈情報をいかなる手法で用いるのか検討する必要がある．

本研究では上記の課題に対し，文脈情報を既存の記事から抽出し用いることとした．そして，あらかじめ文脈情報を付加しておくことで，訳語選択のたびに文脈情報を計算する必要がないため訳語選択時の計算時間が短縮される．また，本研究の訳語選択の手法は，対訳辞書内の同じ見出し語で異なる意味をもつ名詞単語が翻訳元文書中に現れたとき，その単語と同じ文中に存在した他の名詞単語の訳語と訳語候補間の文脈情報のスコアの合計値が最大になるものを選択することとした．

上記を考慮し，文脈情報付き対訳辞書を作成し，対訳辞書内の同じ見出し語で異なる意味をもつ名詞単語に対する文脈を考慮した訳語選択により翻訳するシステムを実装した．また，実際にシステムを用いて対訳辞書内の約 3,000 語に文脈情報を付加し，対訳辞書中の同じ見出し語で異なる意味をもつ名詞単語に対する訳語選択の結果に与える影響を分析した．

本研究の貢献は以下の 3 点である．

1) 文脈情報を付加した対訳辞書の実現
   対訳辞書に付加する文脈情報として，既存の記事から抽出した文脈情報を用いるようにした．このことにより，対訳辞書内の同じ見出し語で異なる意味をもつ名詞単語に対して，文脈を考慮した訳語選択結果が反映されるため，翻訳の品質が高まる効果を得た．
2) 事前に文脈情報を計算することによる翻訳時の計算時間短縮
   あらかじめ対訳辞書内の単語を抽出し，同じ見出し語で異なる意味をもつ単語に対して文脈情報のスコアを計算し保存しておくことで，訳語選択のたびに文脈情報を計算する必要がなくなり，翻訳時間を短縮することに成功した．
3) 訳語選択時に付加された文脈情報を用いる手法の確立
   対訳辞書内の同じ見出し語で異なる意味をもつ名詞単語が翻訳元文書中に現れたとき，その文を形態素解析した上で，その名詞単語と同じ文中に存在した他の名詞単語の訳語と訳語候補間の文脈情報のスコアの合計値が最大になるものを訳語として選択するという手法を確立した．

Word Selection Using a Bilingual Dictionary with Context Information

Masahiro NAKAGAWA

Abstract

Machine translation (MT) services are useful for overcoming problems in collaboration between users from different language and cultural backgrounds. However, when MT is used in combination with a general dictionary to translate technical documents in a specific area, context-sensitive translation can be difficult. One way to solve the problem is to use a specialized bilingual dictionary, in order to increase the quality of translated sentences containing technical terms. In a bilingual dictionary, technical terms are registered in both the source and the target language. However, correct word selection may not be achieved only with a bilingual dictionary if the headword has multiple different translations.

This research aims to improve word selection for words with several translations in a bilingual dictionary, and proposes a method for correct word selection by adding context information to bilingual dictionaries.

The following three problems exist when words with several translations are translated by MT combined with a bilingual dictionary with context information:

1) Selecting the contents of context information added to a bilingual dictionary.

   There are some points to consider when context information is added to a bilingual dictionary. One of the points is what information should be used as context information for correct word selection of words with several translations in a bilingual dictionary.

2) Computation time while trying to add context information.

   If MT calculates context information whenever words with several translations appear, computation time increases in proportion to the length of the original documents.

3) Method using context information for word selection.

   Correct word selection cannot be achieved only by adding context information to a bilingual dictionary. It is necessary to consider how to use the

context information for word selection of words with several translations in a bilingual dictionary.

In order to solve these problems, this research suggests extracting context information from existing articles. Also, multiple calculations of context information on every word selection can be avoided by adding context information in advance. This reduces the calculation time. This research proposes an approach where the words, which have the highest context information score between translated words and the translations of the words appearing in the sentence where the original word appeared.

Considering the above, the system to select words with several translations in a bilingual dictionary was implemented by using a bilingual dictionary with semantic relatedness (SR) scores as the context information. Effects of the context-sensitive word selection for the words with several translations in a bilingual dictionary were analyzed by adding approximately 3,000 entries of context information to a bilingual dictionary.

Contributions of this research are:

1) Implementing a bilingual dictionary with semantic relatedness (SR) score as context information.

   Semantic relatedness based on tf/idf score is used as context information. Translation quality was increased by adding the SR scores as the context information to a bilingual dictionary.

2) Eliminating unnecessary computations by calculating the context information in advance.

   Unnecessary computations were eliminated by saving the context information scores of words with several translations calculated beforehand.

3) Establishing a method to use context information in word selection.

   Correct word selection is achieved with morphological analysis and by selecting the words that have the highest context information score between translated words and the translations of the words appearing in the same sentence.

文脈情報付き対訳辞書を用いた訳語選択

目次

第1章　はじめに

ますます進むグローバル化により，インターネットユーザは増加の一途をたどっている．そしてインターネットユーザには世界中のさまざまな言語や文化をもつ人々が存在している．しかし，ほとんどのインターネットユーザにとって，世界中で作成されるインターネットコンテンツの大半は理解できる言語で記述されたものではない．唯一の世界の公用語とされる英語のコンテンツであっても，やはり英語を公用語としないユーザにとって理解できるものではない．つまり，インターネット上に情報があるにも関わらず，言語を理解できないことに起因し，ユーザがその情報やインターネットコンテンツを理解できない場合が少なくない．

機械翻訳サービスは，ユーザが全く理解できない言語で記述されたコンテンツでも，自動的にそのコンテンツをユーザにとって理解できる言語に変換することを可能にするサービスである．上述のような，ユーザがインターネット上の情報を理解することができないことがある状況で，機械翻訳は極めて有用なサービスとなっている．なぜならば，機械翻訳サービスによって理解できなかったコンテンツをユーザが容易に理解可能になるからである．しかし，機械翻訳には次のような問題点がある．

機械翻訳によって特定分野の専門的な文書を翻訳するとき，一般的な機械翻訳では，一般語辞書に基づいた翻訳を行っている．そのため，ある分野に特有の専門的な文脈に応じた翻訳をすることが不可能である．つまり，人間が翻訳を行う際には，文脈を考慮することで専門的な単語や言い回しを適切に翻訳することが可能であるが，機械翻訳では専門的な単語や言い回しに対して，適切に翻訳できないという問題が生じる．

上述のような問題を解決する手法として，専門対訳辞書（以下，対訳辞書）を用いて専門的な用語を含む文を翻訳する機能を高める方法がある．対訳辞書とは，あらかじめ専門用語の翻訳元言語と翻訳先言語の対訳として登録しておくものである．つまり，ある単語 w に対してはある訳語 tw を用いて翻訳するという情報を記憶しておくものである．機械翻訳で専門的な用語を含む文を翻訳するとき，対訳辞書中の対訳を参照する手法を用いて，正しく専門的な用語の訳語を選択できるようになる効果がある．対訳辞書に登録された対訳が多ければ多いほど，多くの単語について適切に訳語選択できるようになる．

しかし対訳辞書を用いた機械翻訳において，対訳辞書内に同じ見出し語で異なる意味をもつものが存在する場合がある．そのような場合に，同じ見出し語で異なる意味をもつ単語が翻訳元文書に登場することがあるが，このとき次のような問題が生じる．対訳辞書のデータベースの同じ見出し語をもつレコードのうちから必ず一組のレコードを選択する必要があり，ユーザの意図する意味での訳語が選択されない場合があるという問題である．この問題に対し，特定分野の専門的な文書に対して対訳辞書を用いて機械翻訳する際，翻訳元文書中の同じ見出し語で異なる意味をもつ単語に対する訳語選択の品質の向上を支援することは有用である．

そこで本研究では，対訳辞書内の同じ見出し語で異なる意味をもつ単語の訳語候補に対して文脈に応じた順位をつけて，適切な訳語を選択可能にする手法を提案する．つまり本研究は，対訳辞書内に同じ見出し語で異なる意味をもつ単語対が登録された場合でも，文脈に応じた適切な訳語選択結果を得ることを目的とする．

上述の目的を果たすために本研究では，文脈情報付き対訳辞書を作成する手法を提案する．文脈情報付き対訳辞書とは，対訳辞書に文脈情報を付加したものである．また，文脈情報とは既存の文書の文脈を考慮した時の単語間の意味的な結びつきの強さを表す評価尺度のことである．

本稿では第 2 章で本研究の背景として対訳辞書を用いた機械翻訳における問題点を整理した上で，文脈情報を対訳辞書に導入する際の課題について述べる．第 3 章で対訳辞書に文脈情報を付加することの効果について述べ，文脈情報の内容について詳しく説明し，またその計算手法について述べる．第 4 章では文脈情報の訳語選択時の利用方法の具体的なアルゴリズムを説明する．第 5 章では本研究で実装した文脈情報として意味的関連性の情報を付加した対訳辞書を用いて訳語選択を行い，文を翻訳するシステムについて説明し，その品質を実験によって評価した結果について述べ，考察する．最後に第 6 章で本研究における成果や今後の課題についてまとめる．

第2章　背景

2.1　対訳辞書を用いた機械翻訳

対訳辞書を用いた機械翻訳は，特定分野の専門的な文書に対し，専門的な文脈に応じた翻訳を実現するために有用である．専門的な文書に対する対訳辞書を用いた機械翻訳の効果を示した例を図 2.1 に示す．具体的には，図 2.1 では対訳辞書の有無による機械翻訳の結果の違いを比較した例を示している．図 2.1 中の文は上から順番に原文，対訳辞書を用いずに機械翻訳をした結果の文，対訳辞書を用いて機械翻訳をした結果の文となっている．図 2.1 の対訳辞書ありの機械翻訳を行う際に用いた対訳辞書の抜粋を表 2.1 に示す．たとえば，病院のカテゴリの文書を翻訳したい場合に用いる対訳辞書には表 2.1 のように英語の「operation」に対して日本語の「手術」という単語が訳語として登録されている．

原文）My operation will start at 9:00.
⇒対訳辞書なし）私の 操作 は 9:00 から開始されます.
⇒対訳辞書あり）私の 手術 は 9:00 から開始されます.

図 2.1: 対訳辞書の有無による機械翻訳結果の相違

表 2.1: 図 2.1 の翻訳時の対訳辞書抜粋部分

英語	日本語
…	…
operation	手術
…	…

図 2.1 からわかるように，機械翻訳において対訳辞書を用いることによって「operation」という英単語が，病院というカテゴリに関して，専門的な意味の日本語訳である「手術」という単語に翻訳される．このように対訳辞書を用いることによって，特定分野の専門的な文書の専門的な文脈に応じた機械翻訳が実現される．

実際に，このような対訳辞書を用いた機械翻訳サービスとして「京大翻訳！[1)]」がある.「京大翻訳！」は京大に特化した翻訳を得意としたツールである.「京大翻訳！」による機械翻訳結果の例を図 2.2 に示す．図 2.2 中の文は上から順番に原文，通常の機械翻訳の結果,「京大翻訳！」による機械翻訳の結果である．また，図 2.2 の「京大翻訳！」の機械翻訳を行う際に用いられている対訳辞書の抜粋を表 2.2 に示す．図 2.2 から分かるように，通常の機械翻訳では，原文の「東大路通り」のような京大付近の通りの名前を認識することができず,「Tokyo University road」と誤訳になる．そこで，表 2.2 のように対訳辞書に登録しておくことで,「京大翻訳！」では「東大路通り」の訳語として適切な「Higashioji St.」を選択することを可能とする．また,「京大翻訳！」において用いられている対訳辞書には，約 15,000 レコードの対訳が登録されている．

原文）私は 東大路通り を渡った。
⇒通常の機械翻訳）I crossed Tokyo University road.
⇒「京大翻訳！」）I crossed Higashioji St.

図 2.2: 通常の機械翻訳と「京大翻訳！」による結果の相違

表 2.2: 図 2.2 の翻訳時の対訳辞書抜粋部分

日本語	英語
…	…
東大路通り	Higashioji St.
…	…

しかし，対訳辞書を用いた機械翻訳にも次のような問題がある．機械翻訳が用いる対訳辞書中に同じ見出し語で異なる意味をもつ対訳が存在するときに，文脈を考慮した訳語選択がされず，ユーザの意図した翻訳結果が得られない場合がある．具体的には，対訳辞書の内部において同じ見出し語で異なる意味をもつ対訳がある場合，そのような単語に対する訳語は共に専門的な文脈に対応

[1)] http://www.s-coop.net/smarttrans.html

する目的で登録されたものである．そのため，一般語辞書の内部の訳語よりも優先されるが，対訳辞書内部の同じ見出し語で異なる意味をもつ単語の訳語候補に関して互いの間の優先順位は決定しかねる．しかし，翻訳のためには何らかの規則にしたがって訳語を選択する必要がある．そこで言語グリッド [1] の辞書連携翻訳サービスでは，対訳辞書のデータベースのレコード ID が若い順番に優先順位をつける方法が用いられている．しかし，このような方法では文脈に応じた訳語選択がされないことがあり，ユーザの意図した翻訳結果が得られず問題となる．たとえば，京都に特化した翻訳を作成する際に利用することが可能な，既存の対訳辞書として，NICT 作成の Wikipedia 日英京都関連コーパス付属の専門用例対訳集がある．この対訳辞書には約 50,000 のレコードが存在し，このうち，同じ見出し語で異なる意味をもつ単語は約 3,000 レコード存在する．この辞書を用いて辞書連携翻訳した際に上述の問題が発生する機械翻訳の具体例を図 2.3 に示す．また，図 2.3 の機械翻訳を行う際に用いた対訳辞書の抜粋部分を表 2.3 に示す．この対訳辞書内において「Ako」という英単語に対し，「阿衡」（役位の名前）という日本語が訳語として登録されている．また同じ対訳辞書内に別のレコードとして「Ako」という英単語に対し,「赤穂」（地名）という日本語が登録されている．

原文）He was born as a son of a statesman of the Ako clan in Harima Province in 1832.
⇒結果）彼は 1832 年に播磨藩 阿衡 一族の政治家の息子として誕生しました.
正解文）1832 年、播磨 赤穂 藩の子弟として生まれる。

図 2.3: 対訳辞書を用いた機械翻訳の誤訳例

図 2.3 のように言語グリッドの辞書連携サービスでは，機械翻訳をするとき原文に「Ako」というフレーズが存在すると，すべての「Ako」に対して「阿衡」が訳語として選択される．なぜならば，機械翻訳の際に機械翻訳システムが原文を形態素解析し，原文の単語単位で対訳辞書をレコード ID 順に調べていき，該当レコードを見つけると，レコード走査を中止し，その訳語を選択するから

表 2.3: 図 2.3 の翻訳時の対訳辞書抜粋部分

英語	日本語
…	…
Ako	阿衡
…	…
Ako	赤穂
…	…

である．つまり,「Ako」の例の場合，レコード ID 順に関して上位レコードの「阿衡」のみが選択される候補となり，下位レコードの「赤穂」は選択されないからである．以上のように，対訳辞書を用いた機械翻訳の際に用いる対訳辞書内に同じ見出し語で異なる意味をもつ単語がそれぞれ対訳として登録されている場合，適切な訳語選択がされない場合があるという問題が存在する．

このような問題を解決するために，その文ごとの文脈に基づき，文脈を考慮した訳語選択を行う必要がある．このような文脈の情報，つまり文中の他の単語との意味的な結びつきの強さを表す評価尺度を文脈情報と呼ぶ．文脈情報を用いて，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対して適切な訳語を選択することが実現可能であるならば，上述のような問題を考慮せずとも対訳辞書を作成することができる．さらに，同じ見出し語で異なる意味をもつ単語を含む対訳辞書を用いて，機械翻訳をすることが可能になる．

また，対訳辞書を作成する方法は，熊野らによる「対訳文書から機械翻訳専門用語辞書作成」の研究 [2] で確立され，現在までにさまざまな手法が提案されてきている．その手法はさまざまであり，統計情報を用いる方法 [3] や共起後集合の類似度を用いる方法 [4] などが考案されている．そのような手法に基づき，100 %近くの適合率で対訳コーパスから自動で対訳辞書を作成する研究 [5] もなされている．

よって本研究では，既存の対訳コーパスから生成された，対訳辞書を用いて実験および評価を行う．

2.2 文脈情報の導入における課題

対訳辞書を用いた機械翻訳時に，対訳辞書の同じ見出し語で異なる意味をもつ単語に対して，ユーザの意図した訳語選択結果を得ることができない問題がある．このような問題を解決するために，対訳辞書内の同じ見出し語で異なる意味をもつ単語と同文中の他の語との文脈を考慮して機械翻訳をする必要がある．つまり，問題解決のために，文脈情報を用いて文脈に沿った適切な訳語選択を行うことが必要である．そこで本研究では，対訳辞書に文脈情報を付加し，文脈情報付き対訳辞書を作成することで，このような問題を解決することを目標とする．また，文脈情報を付加した対訳辞書の概要を表 2.4 に示す．

英語	日本語	文脈情報
…	…	…
英語 A	日本語 a	a に関する文脈情報
…	…	…
英語 A	日本語 b	b に関する文脈情報
…	…	…

表 2.4: 文脈情報付き対訳辞書の概要

表 2.4 のように，対訳辞書内において英語 A に対して，日本語 a および日本語 b が存在するとする．このとき，訳語候補 a および b に対するそれぞれの文脈情報を用いることで，文脈を考慮した訳語選択が実現可能となる．

実際に文脈情報付き対訳辞書を作成し，用いるためには以下のような課題がある．

1) 文脈情報の取得方法

   具体的にどのような手法で文脈情報を取得し，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対する訳語を適切に選択できるのかについて考える必要がある．しかし文脈情報があったとしても，それが訳語選択に応用できないものでは意味がない．そこで対訳辞書を用いた機械翻訳による翻訳の際，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対する訳語選択において，有用である文脈情報を取得しなければならない．

2) 文脈情報の計算手法

文脈情報の取得方法が決定すると，次はその計算手法を考慮する必要がある．すべての対訳に文脈情報を付加すると対訳辞書が大きなものである場合に膨大な計算時間となる．そこで，訳語選択の際の計算時間を減少させる工夫が必要である．

3) 訳語選択での文脈情報の具体的な利用方法
対訳辞書内の同じ見出し語で異なる意味をもつ単語が翻訳元文書に出現した際に，対訳辞書に付加しておいた文脈情報を用いて訳語選択を行う．その際，具体的にどのような手法により訳語選択を行うのかを考える必要がある．つまり，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対する訳語選択と，対訳辞書に付加された文脈情報をどのように結びつけて利用するかという方法を考えなければならない．

上記の課題を考慮して，本研究では文脈情報付き対訳辞書を作成し，それを用いて機械翻訳するシステムを作成する．そのことにより，特定分野の専門的な文書における専門的な文脈の用語に対して，対訳辞書中で同じ見出し語で異なる意味をもつ単語があった場合でも適切に訳語を選択することを実現する．

次章では，より具体的に文脈情報や文脈情報付き対訳辞書について説明する．

第3章　文脈情報付き対訳辞書

3.1　対訳辞書への文脈情報の付加

前章で述べたように，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対する訳語を適切に選択することができない問題がある．本研究では，この問題を解決し，同じ見出し語で異なる意味をもつ単語に対する訳語選択品質の向上を目的とし，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対する文脈に沿った訳語選択を行う手法を提案する．

文脈情報を用いて訳語選択および機械翻訳を行う研究はさまざまなものが行われている．文脈情報を用いてピボット翻訳をサポートする研究 [6] がなされている他，結合価文法に基づいて訳語選択を行う手法 [7] や単語間の意味的な距離に基づいて訳語選択を行う手法 [8] がある．しかし，これらの文脈情報を用いて訳語選択を行う手法はすべて，訳語選択のたびに計算を行わねばならない．なぜならば，すべての単語に対して，文脈情報を用いて訳語選択を行うことを目的としているからである．

本研究では文脈情報を用いた訳語選択を行う対象を対訳辞書内の同じ見出し語で異なる意味をもつ単語に限定する．このことによって，事前に計算された文脈情報を付加しておくことで，同じ見出し語で異なる意味をもつ単語が翻訳元文書に出現した際に，機械翻訳が文脈情報の計算をせずとも，付加された文脈情報を参照する操作のみで，訳語を選択し，翻訳することが可能になる．つまり，文脈情報の必要な単語が限定されるため，対訳辞書に文脈情報を付加することで，翻訳の際に文脈情報を計算するのではなく，計算済みの文脈情報を取得するのみで良くなる．したがって，対訳辞書を用いた機械翻訳を行う際，文脈情報計算する必要がないため，機械翻訳の時間を短縮することが可能になる．

3.2 文脈情報の内容および取得方法

対訳辞書を用いた機械翻訳において翻訳元文書に，対訳辞書内の同じ見出し語で異なる意味をもつ単語が存在した際に，そのような単語の訳語選択を行うための文脈情報としてどのような情報を用いるのかという課題がある．そこで本研究では，文脈情報を既存の文書から抽出した文中の単語間の意味的な結びつきの強さを表す評価尺度と定義する．

本研究が提案する対訳辞書へ文脈情報を付加するシステムの構成図を図 3.1 に示す．

このシステムは，辞書連携翻訳のための対訳辞書に，2 つの単語の訳語間の文脈情報を付加し，文脈情報付き対訳辞書を作成するシステムである．このシステムでは，まず Wikipedia 全記事を形態素解析し，作成した Wikipedia の記事ごとの名詞に関するインデックス情報を，文脈情報のスコアを算出する処理の入力として用いる．また，対訳辞書内の同じ見出し語で異なる意味をもつ単語の訳語と，一般語辞書の単語の間の文脈情報を対訳辞書に付加する．以上のような処理過程により，文脈情報付き対訳辞書が生成される．

具体的な文脈情報付き対訳辞書の一例を表 3.1 に示す．

対訳辞書内において同じ見出し語で異なる意味をもつ単語 A や C には訳語候補それぞれに，翻訳先言語の一般語との間の文脈情報が付加されている．また，B のような見出し語と訳語が一対一で対応しているものに関しては，本研究で問題としている同じ見出し語で異なる意味をもつ単語の訳語選択とは異なり，通常の対訳辞書を用いた翻訳で翻訳できるため，文脈情報を計算する必要がない．したがって，B には文脈情報は付加しなくてもよい．なお，この辞書

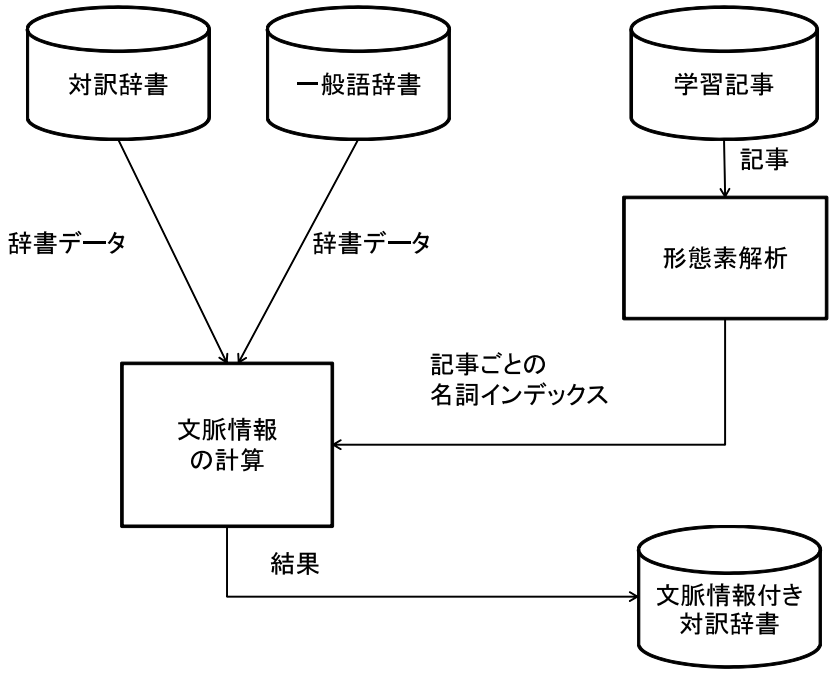

図 3.1: 対訳辞書に文脈情報を付加するシステム構成図

表 3.1: 文脈情報付き対訳辞書の例

見出し語（英）	訳語（日）	一般語 α（日）	一般語 β（日）	…（一般語）
A	a	a と α の文脈情報	a と β の文脈情報	…
A	b	b と α の文脈情報	b と β の文脈情報	…
B	c			
C	d	c と α の文脈情報	c と β の文脈情報	…
C	e	d と α の文脈情報	d と β の文脈情報	…
…	…	…	…	…

の行数は，対訳辞書のレコード数である．また，列数は入力として用いた一般語辞書の一般語の数 +2 となる．この 2 という数字は図中の左の見出し語と訳語の列である．また本研究の実装したシステムにおいては，入力として用いた一般語辞書の一般語の数は約 80,000 語である．

3.3 文脈情報の計算手法と計算時間

本研究では，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対する適切な訳語選択を支援するための，文脈情報付き対訳辞書を作成する．

また，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対する適切な訳語選択を次のように実現する．対訳辞書内の同じ見出し語で異なる意味をもつ単語のそれぞれの訳語ごとに，その単語と翻訳元文書で共起した一般語との文脈情報の総和を比較する．そして，文脈情報の総和が最大となる訳語を，対訳辞書内の同じ見出し語で異なる意味をもつ単語の訳語として選択する．このとき，文脈情報の計算に関する問題として次のものがある．文脈情報は翻訳先言語の既存の記事のすべてを調べて計算を行うため，計算にかかる時間が長くなる．また，翻訳時に文脈情報の計算を逐一行っていては，翻訳にかかる時間が長くなる．

そこで本研究では，あらかじめ対訳辞書内の同じ見出し語で異なる意味をもつ単語を抽出し，同じ見出し語で異なる意味をもつ単語の訳語に関して，一般語辞書内の翻訳先言語の単語との間の文脈情報を計算する．またその文脈情報の計算結果を付加しておくことで，対訳辞書内の同じ見出し語で異なる意味をもつ単語の訳語選択をする際に，文脈情報の計算を行うことなく，付加された文脈情報を参照する操作のみで済ませることを可能とする．つまり，同じ見出し語で異なる意味をもつ単語を含む文に対して文脈情報を用いて訳語選択し，翻訳時の計算時間を短縮することを可能とする．

たとえば，名詞 A と名詞 B の文脈情報が必要になった際にそれを計算し，文脈情報を得る方法では，文脈情報が必要となるたびに計算をしなくてはならない．しかし，本研究で作成するシステムではあらかじめ文脈情報を付加しておくため，名詞 A と名詞 B の文脈情報が必要になった際にそれを参照する操作のみで済むため，上述のように計算時間を短縮することを可能とする．

本章では，対訳辞書内の同じ見出し語で異なる意味をもつ単語を含む文に対する機械翻訳における，適切な訳語選択を行うための文脈情報について詳しく

述べた．具体的には，対訳辞書へ文脈情報を付加することについての効果に関する説明をし，その手法について述べた．また，文脈情報とは既存の文書から抽出した単語間の意味的なつながりの強さを表す評価尺度であるが，その文脈情報の計算方法の説明を計算時間に触れながら詳しく述べた．

次章では，対訳辞書を用いた機械翻訳をする際に，翻訳元文書に対訳辞書内の同じ見出し語で異なる意味をもつ単語が出現した場合，対訳辞書に付加された文脈情報を用いて，適切な訳語選択を行うアルゴリズムを詳しく述べていく．

第4章　訳語選択と文脈情報付き対訳辞書

前章では，対訳辞書に文脈情報を付加することで，文脈を考慮した適切な訳語選択を行う際の文脈情報の取得方法や計算方法について述べた．

本章では，具体的に対訳辞書内において同じ見出し語で異なる意味をもつ単語に対する文脈情報付き対訳辞書を用いた訳語選択の手法について述べる．つまり，対訳辞書内の同じ見出し語で異なる意味をもつ単語が出現する翻訳元文書に対する，機械翻訳をする際の訳語選択において，対訳辞書に付加された文脈情報を利用する方法について述べる．

文脈情報付き対訳辞書を用いた訳語選択のアルゴリズムを図 4.1 に示す．

原文を s，対訳辞書内の同じ見出し語で異なる意味をもつ単語のレコード群を BD とする．また，単語 x と単語 y の文脈情報の値を $CI(x, y)$ と表す．

このとき，アルゴリズムは図 4.1 にしたがう．

また，図 4.1 のアルゴリズムのフローチャートを図 4.2 に示す．$R(k_s)$ は k_s に対する本システムを介した訳語選択結果を表す．

図 4.2 中の「訳語選択」の部分では，同じ見出し語で異なる意味をもつ単語 k_s の訳語候補の集合 tK_s の各要素に対し，その要素と tO_s の各要素の文脈情報の値を取り出し，その合計値を比較することで tK_s の各要素のうちその合計値が最大となるものを訳語として選択するという操作が行われている．これはアルゴリズムの 6) および 7) の部分に相当する．

この部分の操作について具体的に表 4.1 の文脈情報付き対訳辞書の例を用いて説明する．同じ見出し語で異なる意味をもつ単語の訳語候補の集合 tK_s の要素が $\{tk_1, tk_2\}$ であり，同じ見出し語で異なる意味をもつ単語以外の名詞単語の訳語 tO_s の要素が $\{to_1, to_2\}$ のときを考える．このとき，tk_1 に関する

1) s を形態素解析し，s 中のすべての名詞 W_s=$\{w_1, w_2, ..., w_n\}$ を取得
2) BD を調べ，W_s を同じ見出し語で異なる意味をもつ単語 $k_s = \{w_i | w_i \in BD, 0 \leq i \leq n\}$ および，それ以外の名詞単語 $O_s = \{o_1, o_2, ..., o_j\}$ に分類
3) k_s が存在しないなら 8) へ
4) BD を用いて，k_s の訳語候補群 $tK_s = \{tk_1, tk_2, ..., tk_h\}$ を獲得
5) 同様に一般語辞書から，O_s のすべての訳語群 $tO_s = \{to_1, to_2, ..., to_m\}$ を獲得
6) tK_s の要素ごとに tO_s のすべての要素との間の文脈情報の値 (CI) の合計 $\sum_j^m CI(tk_l, to_j)(1 \leq l \leq h)$ を計算
7) CI の合計値を最大とする tk_l を k_S の訳語として選択する
8) 次の文へ

図 4.1: 文脈情報付き対訳辞書を用いた訳語選択アルゴリズム

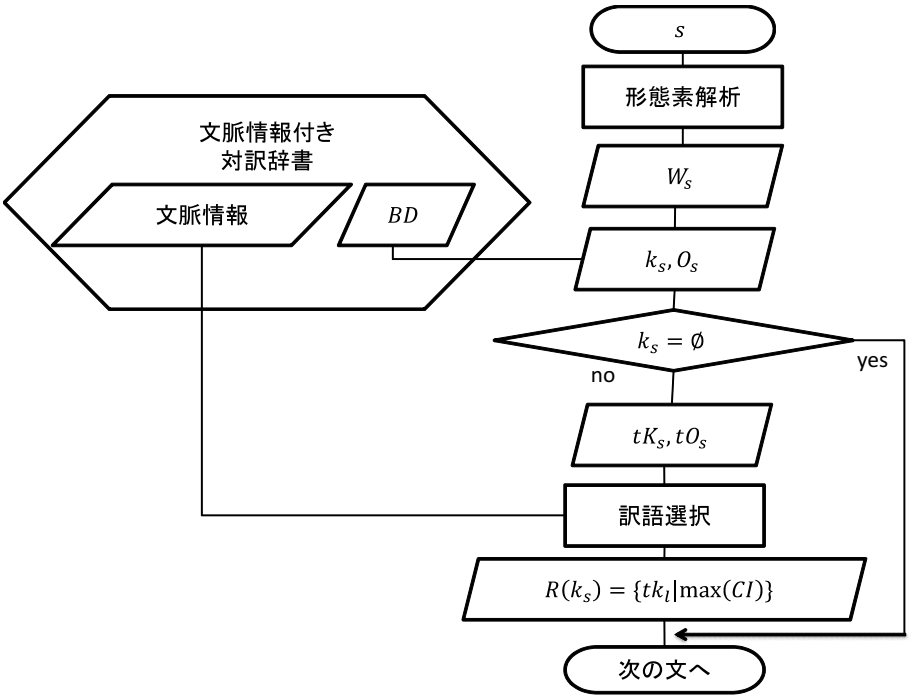

図 4.2: 文脈情報付き対訳辞書を用いた訳語選択アルゴリズムのフローチャート

文脈情報の合計値 $CI(tk_1, to_1) + CI(tk_1, to_2)$ と tk_2 に関する文脈情報の合計値 $CI(tk_2, to_1) + CI(tk_2, to_2)$ を比較し，合計値の大きいものを訳語として選択するというような操作が行われている．表 4.1 の例では，それぞれの値は次のようになる．tk_1 に関する文脈情報の合計値=0.3+0.2=0.5，tk_2 に関する文脈情報の合計値=0.4+0.3=0.7 である．つまり，この例においては，より合計値の高い tk_2 を訳語として選択する．

表 4.1: 訳語選択の際に用いる文脈情報の例

	to_1	to_2
tk_1	0.3	0.2
tk_2	0.4	0.3

第 5 章　実装と評価

前章では，文脈情報付き対訳辞書を用いた訳語選択のアルゴリズムについて説明した．

本章では，本研究において作成した文脈情報付き対訳辞書を用いた機械翻訳システムについて説明する．このシステムは，対訳辞書を用いた機械翻訳の問題点を解決することを目的として作成したものである．つまり，機械翻訳を行う際，対訳辞書内において同一の見出し語で異なる意味をもつ単語が翻訳元文書に存在した場合に，文脈情報付き対訳辞書を用いて適切な訳語選択を行うシステムについて述べる．さらに，システムを用いた評価実験について述べ，考察をする．

5.1　システムの実装

本研究では，既存の対訳辞書に文脈情報を事前に付加した文脈情報付き対訳辞書を作成し，機械翻訳サービスと連携し翻訳を行う．その際に，翻訳元文書内に対訳辞書内の同じ見出し語で異なる意味をもつ単語が存在した場合でも適切な訳語選択による機械翻訳を行うシステムを実装した．なお，機械翻訳には言語グリッドのサービスである，辞書連携翻訳サービスを用いた．

文脈情報付き対訳辞書を用いた訳語選択を行うシステムの構成図を図 5.1 に

示す．このシステムでは翻訳元文書の原文を形態素解析し，その中の名詞を抽出する．その後，文脈情報付き対訳辞書から辞書データを受け取り，抽出した名詞を同じ見出し語で異なる意味をもつ単語とその他の単語に分ける．そして，文脈情報付き対訳辞書の文脈情報のスコアを用いて，前章で述べたアルゴリズムにしたがい，訳語候補ごとに同文中に存在した名詞との文脈情報の合計値を比較する．そして，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対する訳語を選択する．その後，選択された訳語を用いて機械翻訳を実行する．

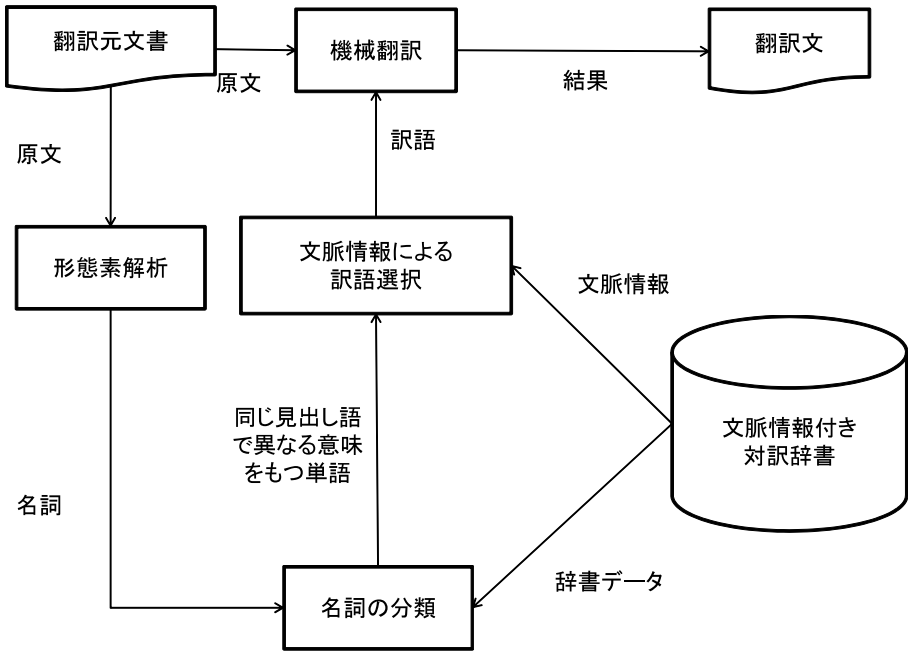

図 5.1: 文脈情報付き対訳辞書を用いた訳語選択システム構成図

また，文脈情報付き対訳辞書を用いた機械翻訳の流れをシーケンス図として図 5.2 に示す．順を追ってこの流れを述べる．まず，翻訳元文書の原文を形態素解析し名詞を抜き出す．そして，対訳辞書と比較することで，抜き出した名詞を対訳辞書内の同じ見出し語で異なる意味をもつ単語とその他の単語に分類し，その訳語を抽出する．もし対訳辞書内の同じ見出し語で異なる意味をもつ単語が存在しない場合は，そのまま原文を機械翻訳に渡し，翻訳する．また，もし対訳辞書内の同じ見出し語で異なる意味をもつ単語が存在した場合は，対訳辞書に付加された文脈情報の値を取り出す．つまり，対訳辞書内の同じ見出し語で異なる意味をもつ単語の訳語候補それぞれとその他の名詞の間の文脈情報の値

を合計し，その値がもっとも大きくなった訳語候補を訳語として選択する．そして，機械翻訳でその訳語選択結果を利用して翻訳を行う．

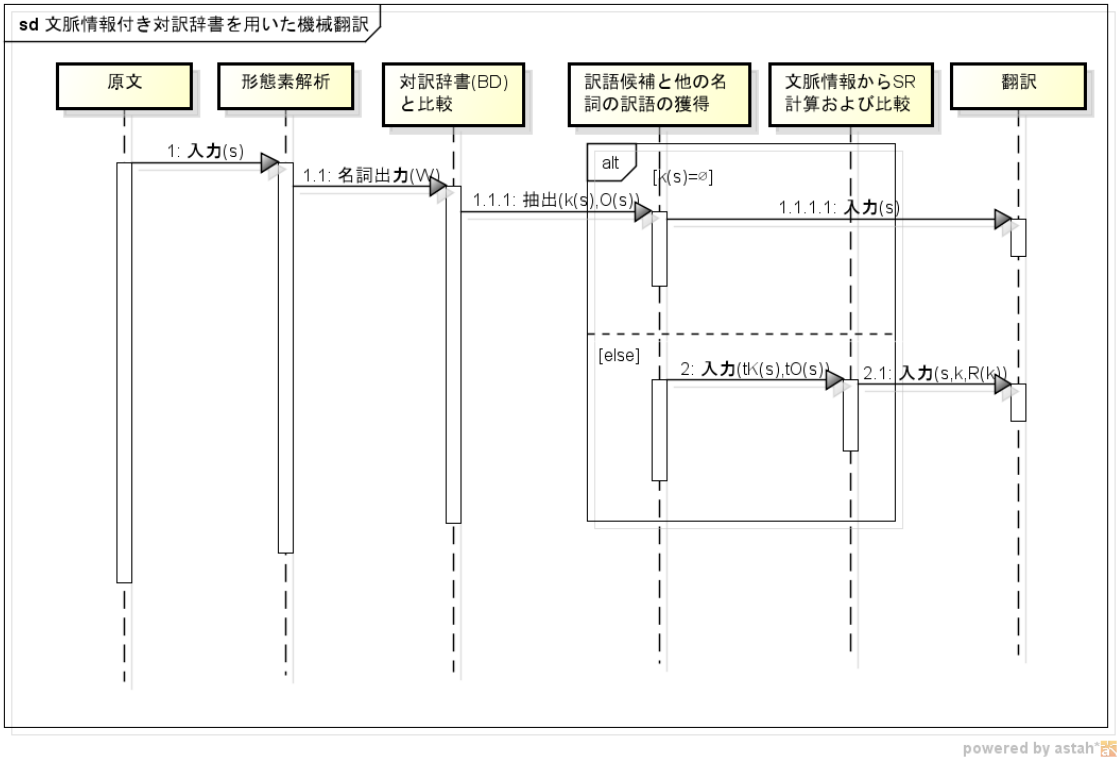

図 5.2: 文脈情報付き対訳辞書を用いた機械翻訳シーケンス図

つまり，対訳辞書内の同じ見出し語で異なる意味をもつ単語を含む文書に対してでも適切に対訳辞書を用いた機械翻訳を行うこと可能にするために，本研究では図 3.1 に示した既存の対訳辞書に文脈情報を付加するシステムと，図 5.1 に示した文脈情報付き対訳辞書を用いた訳語選択を行いつつ機械翻訳を行うシステムを作成した．

5.2 文脈情報としての意味的関連性

本研究で作成したシステムを評価する実験を行うにあたって，文脈情報のスコアの評価尺度の計算手法を具体的にする必要がある．

適切な訳語選択を行うためにさまざまな文脈情報の内容が考えられる．また，文脈情報に基づいて訳語選択を行う手法についてはさまざまな研究がなされている．

その中でも本研究では，システムの評価実験を行うにあたって，Wikipedia 記事から単語間の意味的関連性を計算する理論 [9] に基づき，意味的関連性を計算する手法を定式化し，文脈情報として利用できるようにした．つまり，文脈情報として Wikipedia 記事に基づく意味的関連性（SR）という評価尺度を用いた．この理論を用いた理由は，Wikipedia は今日さまざまなカテゴリごとにもっとも記事数の多い Web 百科事典であるからである．具体的な数値では Wikipedia には現在，英語で約 3,400,000 記事，日本語で 700,000 記事存在する．

意味的関連性（SR）を計算するために，重み付き制約充足問題 [10] に基づいて定式化することを考える．

以下では，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対する訳語選択に用いる意味的関連性を求めるための，重み付き制約充足問題に基づいた定式化を行う方法について述べる．このとき，以下の要件に基づいて意味的関連性に基づく訳語選択を定式化する．

1) 原文書中の同一見出し語で異なる意味をもつ名詞単語 w の訳語候補は，対訳辞書での w のすべての名詞の訳語
2) 文中の訳語間には意味的関連性がある
3) 訳語間の意味的関連性の総和を最大とするような，原文書中の同じ見出し語で異なる意味をもつ名詞単語に対する訳語を解とする．

２つの単語間の意味的関連性を求めることについて考える．このとき，上述の要件に基づいて，Wikipedia 記事から単語間の意味的関連性を計算する理論に基づき意味的関連性の値を計算する．

ある単語 x_i と x_j の意味的関連性を求めるとする．このとき，x_i が翻訳先言語の Wikipedia 記事に出現した回数を用いて，tf/idf 法に基づくスコアを計算し，x_i と各記事に対する意味的な関連性の強さ（重み）を決定する．そして，各記事に対して重みづけされた訳語のベクトル $v_{x_i} = (v_{x_i1}, v_{x_i2}, ..., v_{x_im})$（$m$ は翻訳先言語の Wikipedia の記事数）を得る．

このとき，v_{x_ik} は $v_{x_ik} = (1 + logtf(i,k))log\frac{m}{l}$ として計算できる．これは，x_i が m 記事のうちの第 k 番目の記事中に $tf(i,k)$ 回出現しており，さらに翻訳先言語の Wikipedia 記事全体の m 記事のうち l 記事において x_i が出現していることを表している．

このような計算を翻訳先言語の Wikipedia 記事に対して行うことで，ある単語 x_i に対する訳語のベクトル v_{x_i} を算出することができる．そして v_i と v_j の間

のコサイン相関値を計算することで，0 以上 1 以下の値を取る 2 つの訳語間の意味的関連性の値を定量的に求めることが可能になる．よって x_i と x_j の間の意味的関連性の値 $SR_{ij}(x_i, x_j)$ は以下のように表すことができる．

$$SR_{ij}(x_i, x_j) = \frac{v_{x_i1}v_{x_j1} + v_{x_i2}v_{x_j2} + ... + v_{x_im}v_{x_jm}}{\sqrt{v_{x_i1}^2 + v_{x_i2}^2 + ... + v_{x_im}^2}\sqrt{v_{x_j1}^2 + v_{x_j2}^2 + ... + v_{x_jm}^2}}$$

この関数 SR を用いることで，任意の 2 つの単語間の意味的関連性を定量的に計算し，比較することが可能になる．

ここまでの計算手法をまとめたものを図 5.3 に示す．

任意の 2 つの名詞単語：

$$x_i,\ \ x_j$$

ある名詞単語の訳語のベクトル：

$$v_{x_i} = (v_{x_i1}, v_{x_i2}, ..., v_{x_im})$$

各ベクトル要素：

$$v_{x_ik} = (1 + logtf(i,k))log\frac{m}{l}$$

(x_i が m 記事のうちの第 k 番目のインデックスとして記事中に $tf(i,k)$ 回出現しており，さらに翻訳先言語の Wikipedia 記事全体の m 記事のうち l 記事において x_i が出現)

任意の 2 つの名詞単語間の意味的関連性：

$$SR_{ij}(x_i, x_j) = \frac{v_{x_i1}v_{x_j1} + v_{x_i2}v_{x_j2} + ... + v_{x_im}v_{x_jm}}{\sqrt{v_{x_i1}^2 + v_{x_i2}^2 + ... + v_{x_im}^2}\sqrt{v_{x_j1}^2 + v_{x_j2}^2 + ... + v_{x_jm}^2}}$$

図 5.3: 意味的関連性の計算手法

上記の方法で，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対しての適切な訳語選択を実現するための文脈情報としての意味的関連性の値を計算することが可能になる．

対訳辞書に付加された文脈情報としての意味的関連性のスコアを用いた訳語選択の具体例を図 5.4 に示す．また，図 5.4 の翻訳をする際に必要となる文脈情報付き対訳辞書の一部分を表 5.1 に示す．図 5.4 の原文を翻訳するとき，この文には同じ見出し語で異なる意味をもつ単語である「Ako」が存在する．また，同文中の名詞単語として「son」,「statesman」,「clan」および「Harima」がある．このとき,「Ako」に対する訳語候補「阿衡」および「赤穂」から文脈情報付き対訳辞書を用いて適切な訳語を選択するための具体的な処理を説明する．表 5.1 の左側は対訳辞書で，右側は文脈情報としての意味的関連性の値の表である．

例文）He was born as a son of a statesman of the Ako clan in Harima Province in 1832.
本システム翻訳結果）彼は 1832 年に 播磨 藩 赤穂 の 一族 の 政治家 の 息子 として誕生しました.

図 5.4: 「Ako」に関する訳語選択の例文の翻訳結果

表 5.1: 図 5.4 の例文を翻訳する際に必要な文脈情報付き対訳辞書抜粋部分

対訳辞書		意味的関連性の文脈情報				
英語	日本語	息子（son の訳語）	政治家（statesman の訳語）	一族（clan の訳語）	一門（clan の訳語）	播磨（Harima の訳語）
Ako	阿衡	0.05	0.40	0.15	0.05	0.05
Ako	赤穂	0.15	0.10	0.25	0.15	0.35

「Ako」と同文中に存在した「son」,「statesman」,「clan」および「Harima」のそれぞれの訳語と,「阿衡」および「赤穂」に関する文脈情報（意味的関連性のスコア）が必要となる．表 5.1 から,「阿衡」と「息子（son の訳語）」,「政治家（statesman の訳語）」,「一族（clan 訳語）」,「一門（clan 訳語）」および「播磨（Harima の訳語）」のそれぞれの訳語の間の意味的関連性の合計値は 0.05+0.40+0.15+0.05+0.05=0.70 となる．同様に「赤穂」と「息子（son の訳語）」,「政治家（statesman の訳語）」,「一族（clan 訳語）」,「一門（clan 訳語）」お

よび「播磨（Harima の訳語）」のそれぞれの訳語の間の意味的関連性の合計値は 0.15+0.10+0.25+0.15+0.35=1.00 となる．つまり，この例文では「Ako」の訳語として，意味的関連性のスコアの合計値のより大きな「赤穂」が選択される．

以上のように意味的関連性を文脈情報として利用し訳語選択を実現する．

5.3 文脈情報付き対訳辞書の評価

本研究では，文脈情報付き対訳辞書を用いた機械翻訳システムを作成した．具体的には，対訳辞書内の同じ見出し語で異なる意味をもつ単語が，翻訳元文書に存在した場合も，そのような単語に対する適切な訳語選択を行いつつ，機械翻訳を行うシステムである．本節では，作成したシステムによる訳語選択品質の評価を行う．

5.3.1 実験方法

以下の手順で作成したシステムを評価するための実験を行った．

1) 対訳辞書として，既存の対訳コーパスである NICT 作成の「Wikipedia 日英京都関連コーパス[1)]」に付属の専門用語対訳集を用いる．
2) 本研究作成システムによって，対訳辞書中において同じ見出し語で異なる意味をもつ単語に対して，文脈情報を付加する．
3) 文脈情報を付加した見出し語の内，Wikipedia 内で出現頻度の高いものから順に 50 語を抽出する．抽出した 50 語の見出し語について，実験を行うための例文を 50 文獲得する．この例文はそれぞれの見出し語を 1 つずつ含み，かつ他にも名詞単語を含むものとし，Weblio の例文検索システムを用いて獲得する．
4) それぞれの例文に対して，本研究で作成した文脈情報付き対訳辞書を用いた機械翻訳を行う．
5) また，比較のために通常の辞書連携翻訳システムを用いて機械翻訳を行う．この辞書連携翻訳システムでは，同じ見出し語で異なる意味をもつ単語の訳語選択をする際，辞書内の ID が上位のレコードを選択する．
6) それぞれの同じ見出し語で異なる意味をもつ単語に関する訳語選択の結果を比較し，分析する．また本実験では翻訳元言語を英語，翻訳先言語を日本語とする．

[1)] http://alaginrc.nict.go.jp/WikiCorpus/

また,「Wikipedia 日英京都関連コーパス」に付属の対訳辞書は約 50,000 レコードの対訳が存在し，そのうち同じ見出し語で異なる意味をもつものは約 3,000 レコードである．

5.3.2 評価結果

本節では評価実験の結果について述べる．実験を行った 50 文の内，同じ見出し語で異なる意味をもつ単語に対して文脈に応じた適切な訳語が選択される機械翻訳が行われた文の数をそれぞれの手法ごとに調べた．その結果を表 5.2 に示す．

表 5.2: 実験結果

	正しく訳語選択された文の数
本研究作成システム	45（文）
通常の辞書連携翻訳	25（文）

本研究作成システムによって，同じ見出し語で異なる意味をもつ単語を含む 50 文の内，45 文で適切な訳語選択により機械翻訳された．また，同じ見出し語で異なる意味をもつ単語に対して，対訳辞書内の ID の上位のものを選択する辞書連携翻訳では，25 文で適切な訳語選択により機械翻訳された．

この結果について，次節で詳しく考察する．

5.3.3 考察

作成したシステムに対する評価実験の結果について考察する．

本研究作成システムは通常の辞書連携翻訳システムに比べ，適切な訳語選択を行う割合が 40 %高いという結果となった．つまり，対訳辞書内に同じ見出し語で異なる意味をもつ単語が存在した場合の辞書連携翻訳をする際に，本研究で作成したシステムは従来のものよりも有用であるということが分かった．

また，本研究作成システムによって翻訳された 50 文に対する結果を次のように分類して考察をする．

1) 対訳辞書内の同じ見出し語で異なる意味をもつ単語に対して，適切な訳語が選択されつつ機械翻訳が行われた文
2) 対訳辞書内の同じ見出し語で異なる意味をもつ単語に対して，適切な訳語が選択されずに機械翻訳が行われた文

1) の適切な訳語選択の結果が反映されて機械翻訳された例文については，本研究で作成したシステムがうまく動作したと言える．つまり，このような例文では，例文中の対訳辞書内の同じ見出し語で異なる意味をもつ名詞単語と，同文に存在した他の名詞単語との文脈情報の値が正しく評価され，文脈に沿った訳語選択が行われたと判断することが可能である．具体的には，適切である訳語が適切でない訳語に比べて，意味的関連性の文脈情報の合計値に関して高いスコアを得たため，適切な訳語選択がされた．図 5.4 の例文はこのパターンの一例である．

2) の適切な訳語選択の結果が反映されずに機械翻訳された例文については，文脈に沿った適切な訳語よりも，そうでない訳語の方が意味的関連性の文脈情報の合計値においてより高いスコアを得た．これは文脈情報を抽出した元記事データにおいて，適切でない訳語が同文に存在した他の名詞の訳語との結びつきが強いことが原因であり，適切でない訳語の一般度が高いことに起因する．単語の一般度が高いということはつまり，文脈情報を抽出した元記事中のさまざまな記事においてその単語が登場するため，多くの単語と結びつきが強いということである．このパターンの一例を図 5.5 に示す．また，図 5.5 の翻訳をする際に必要となる文脈情報付き対訳辞書の一部分を表 5.3 に示す．図 5.5 の原文を翻訳するとき，この文には同じ見出し語で異なる意味をもつ単語である「yo」が存在する．また，同文中の名詞単語として「part」,「days」，および「labor」がある．このとき,「yo」に対する訳語候補「葉」および「庸」から文脈情報付き対訳辞書を用いて適切な訳語を選択するための具体的な処理は以下である．また表 5.3 の左側は対訳辞書で，右側は文脈情報としての意味的関連性の値の表である． 「yo」と同文中に存在した「part」,「days」および「labor」のそれ

表 5.3: 図 5.5 の例文を翻訳する際に必要な文脈情報付き対訳辞書抜粋部分

対訳辞書		意味的関連性の文脈情報			
英語	日本語	一部 （part の訳語）	部分 （part の訳語）	日 （days の訳語）	労務 （labor の訳語）
yo	葉	0.25	0.20	0.30	0.05
yo	庸	0.10	0.05	0.15	0.30

ぞれの訳語と,「葉」および「庸」に関する文脈情報（意味的関連性のスコア）

例文）The yo was a part of the soyocho from which it was possible to be exempted; this annual mandatory 20 days of labor could be substituted with 40 days of the zatsuyo.
本システム翻訳結果）葉は、それが、免除することについて可能であった soyocho の一部でした； この年間義務的 20日 の労務は、zatsuyo の 40 日によって代用できました。
正解文）庸が年間 20 日の労役の義務であるのに対し 40 日の雑徭により庸が免除される。

図 5.5: 「yo」に関する訳語選択の例文の翻訳結果

が必要となる．表 5.3 から,「葉」と「一部（part の訳語）」,「部分（part の訳語）」,「日（days 訳語）」，および「労務（labor の訳語）」のそれぞれの訳語の間の意味的関連性の合計値は 0.25+0.20+0.30+0.05=0.80 となる．同様に「庸」と「一部（part の訳語）」,「部分（part の訳語）」,「日（days 訳語）」，および「労務（labor の訳語）」のそれぞれの訳語の間の意味的関連性の合計値は 0.10+0.05+0.15+0.30=0.60 となる．つまり，この例文では「yo」の訳語として，意味的関連性のスコアの合計値のより大きな「葉」が選択される．しかし，文脈を考慮した正解の訳語は「庸」である．つまり,「yo」の訳語として「葉」および「庸」が存在するが,「葉」の方が一般的な語であり，さまざまな語との意味的関連性のスコアが高いため，適切でない「葉」が訳語として選択される．これが 2) のパターンの具体例である．また，本研究で作成したシステムでは 2) のパターンであっても，例文によっては上記の問題が発生せずに，適切な訳語選択を行うことが可能となる場合がある．たとえば，表 5.2 から分かるように,「労務（labor の訳語）」との意味的関連性のスコアのみを比較すると,「葉」よりも「庸」の方が高い．つまり，同文に存在する名詞単語が「labor」のみの例文であれば，本システムでは原文中の「yo」に対する訳語は「庸」となる．

上述のように，対訳辞書内の同じ見出し語で異なる意味をもつ単語の訳語候補に一般度の高いものが存在する場合を考える．そのような場合に，本研究作成システムによる，文脈を考慮した適切な訳語選択を行う機能が適用できない例文が存在するという問題が生じる．そこでこのような問題を解決することが今後の課題として考えられる．

第6章　おわりに

本稿では，まず機械翻訳において対訳辞書を用いることで，専門的な文脈に適した訳語を選択できるようになるということについて説明した．また，多くの対訳を登録するとより多くの専門的な文脈に適した訳語を選択できるようになり，対訳辞書を用いた機械翻訳はより良いものとなる．しかし，そのように対訳を追加していくと，同じ見出し語でありながら異なる意味をもつ対訳が追加される場合がある．

本研究では，そのような場合に生じる，対訳辞書内で同じ見出し語で異なる意味をもつ単語に対する訳語選択を含めた機械翻訳を支援するために，文脈情報付き対訳辞書を用いた手法を提案した．その際に，以下のことを課題として挙げ，それぞれを解決した．

1) 文脈情報の内容と取得方法
   具体的にどのような文脈情報に基づくことによって，対訳辞書内の同じ見出し語で異なる意味をもつ単語に対して訳語を適切に選択できるのかという課題があったが，既存の記事から名詞インデックスを取得し，それをもとに算出するという手法を提案した．
2) 文脈情報の計算手法
   文脈情報の内容が決定すると，次はその計算手法を考慮する必要がある．すべての対訳に文脈情報を付加すると文脈辞書が大きなものである場合に膨大な計算時間となる．そのため，対訳辞書を事前に調査し，対訳辞書内の同じ見出し語で異なる意味をもつ単語にのみ文脈情報を付加することで，計算時間を短縮することに成功した．
3) 文脈情報の訳語選択への具体的な利用方法
   対訳辞書内の同じ見出し語で異なる意味をもつ単語が翻訳元文書に出現した際に，対訳辞書に付加しておいた文脈情報から具体的にどのような手法で訳語選択を行うのかという課題に対し，文脈情報付き対訳辞書と訳語選択を結び付けて機械翻訳するためのアルゴリズムを提案した．

実際に，既存の対訳辞書の対訳辞書内の同じ見出し語で異なる意味をもつ単語の訳語に対して，一般語との間の文脈情報の値を計算し，文脈情報を対訳辞書に付加するシステムを作成した．また，文脈情報付き対訳辞書を用いて機械翻訳を行う際に，対訳辞書内の同じ見出し語で異なる意味をもつ単語が翻訳元

文書にあった場合に，そのような単語に対する適切な訳語選択を可能にし，その結果を取り入れて機械翻訳するシステムを作成した．

さらに，作成したシステムの評価を行うための実験を行った．対訳辞書内で同じ見出し語で異なる意味をもつ単語を含む例文を機械翻訳し，適切な訳語選択の実現された割合を調査した．本研究作成システムによる訳語選択結果と，通常の辞書連携翻訳システムによる訳語選択結果を比較した．通常の辞書連携翻訳システムでは，同じ見出し語で異なる意味をもつ単語の訳語選択時に，レコードの ID が上位のものを選択する．この比較実験の結果，テストした 50 文の内，本研究作成システムでは 45 文で適切な訳語選択が行われた．また，同じテスト文に対し通常の辞書連携翻訳システムでは，25 文で適切な訳語選択が行われた．以上の結果から，本研究作成システムは通常の辞書連携翻訳システムと比較して，対訳辞書内の同じ見出し語で異なる意味をもつ単語の訳語選択に関して，高い品質で翻訳できることが分かった．

また本研究作成システムによって，適切な訳語選択の結果が反映されて機械翻訳された例文については，本研究で作成したシステムがうまく動作したと言える．つまり，このような例文では，例文中の対訳辞書内の同じ見出し語で異なる意味をもつ名詞単語と同文に存在した他の名詞単語との文脈情報の値が正しく評価され，文脈に沿った訳語選択が行われたと判断することが可能である．しかし，適切な訳語選択の結果が反映されずに機械翻訳された例文については，本研究で作成したシステムでの訳語選択結果が誤っていた．その原因は，文脈に沿った適切な訳語よりも，そうでない訳語の方が文脈情報の合計値においてより高いスコアとなったことである．つまり，文脈情報を抽出した元の記事データにおいて，同文に存在した他の名詞の訳語との間の結びつきに関して，適切でないものの方が強く結びついていた．これは，そのようにして選択された適切でない訳語の一般度が高いことに起因する．また，本研究で作成したシステムでは，同じ見出し語に対する訳語選択を行う文でも，例文によっては上記の問題が発生せずに，適切な訳語選択を行うことが可能である場合がある．

今後の課題としては次のことが考えられる．上述のように対訳辞書内の同じ見出し語で異なる意味をもつ単語の訳語候補に，一般度の高いものが存在した場合に，本研究作成システムにより文脈を考慮した適切な訳語選択を行う機能が適用できない例文があるという問題を解決することが考えられる．

また，本研究の成果により，対訳辞書内に同じ見出し語で異なる意味をもつ

単語が存在したとき，文脈を考慮した訳語選択を実現可能である．そのため，見出し語の重なりを気にすることなく対訳辞書を作成することができるようになる．これにより，より大きなカテゴリに関する対訳辞書を用いた特定分野に特化した機械翻訳が実現可能となる．

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石田亨教授に厚く御礼申し上げます．また，日頃から多くのご助言とご協力をいただきました稲葉利江子特定講師に感謝申し上げます．また，技術的な面をサポートしてくださりました中島悠特定助教に感謝申し上げます．そして最後に，普段からお世話になっている石田・松原研究室の皆様に心より感謝いたします．

参考文献

[1] Ishida, T.: Language grid: An infrastructure for intercultural collaboration, *Applications and the Internet, 2006. SAINT 2006. International Symposium on*, IEEE, pp. 96–100 (2006).

[2] 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, Vol. 35, No. 11, pp. 2283–2290 (1994).

[3] 高尾哲康, 富士秀, 松井くにお: 対訳テキストコーパスからの対訳語情報の自動抽出, 情報処理学会研究報告. 自然言語処理研究会報告, Vol. 96, No. 87, pp. 51–58 (1996).

[4] 梶博行, 相薗敏子: 共起語集合の類似度に基づく対訳コーパスからの対訳語抽出, 情報処理学会論文誌, Vol. 42, No. 9, pp. 2248–2258 (2001).

[5] 北村美穂子, 松本裕治: 対訳コーパスを利用した対訳表現の自動抽出, 情報処理学会論文誌, Vol. 38, No. 4, pp. 727–736 (1997).

[6] Tanaka, R., Murakami, Y. and Ishida, T.: Context-based approach for pivot translation services, *International Joint Conference on Artificial Intelligence (IJCAI-09)*, pp. 1555–1561 (2009).

[7] 金出地真人, 徳久雅人, 村上仁一, 池原悟: 結合価文法による動詞と名詞の訳語選択能力の評価, 情報処理学会研究報告, pp. 119–124 (2003).

[8] 小嶋秀樹, 伊藤昭: 文脈依存的に単語間の意味距離を計算する一手法, 情報

処理学会論文誌, Vol. 38, No. 3, pp. 482–489 (1997).

[9] Gabrilovich, E. and Markovitch, S.: Computing semantic relatedness using wikipedia-based explicit semantic analysis, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, Vol. 6, Morgan Kaufmann Publishers Inc., pp. 1606–1611 (2007).

[10] Bistarelli, S., Montanari, U. and Rossi, F.: Semiring-based constraint satisfaction and optimization, *Journal of the ACM (JACM)*, Vol. 44, No. 2, pp. 201–236 (1997).

付録:ソースコード

以下では，今回作成したプログラムのソースコードを記載する．A.1 の WordSelection.java をメイン関数とする．また，各プログラムファイルの処理をシステム構成図の処理と対応させて図 A.1，図 A.2 および図 A.3 に示す.

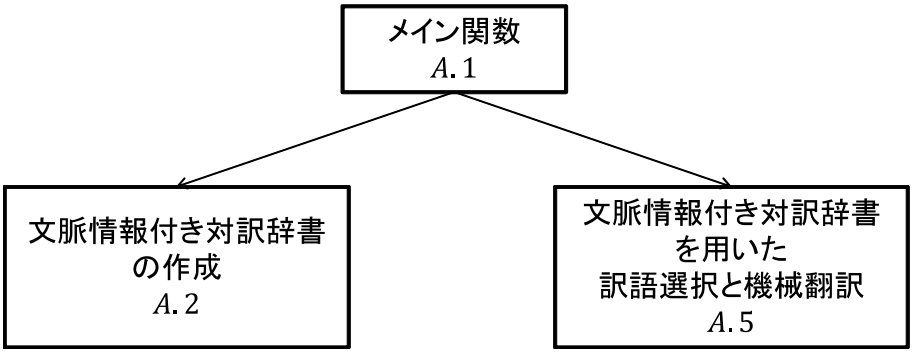

図 A.1: システム構成図（メイン関数周辺）

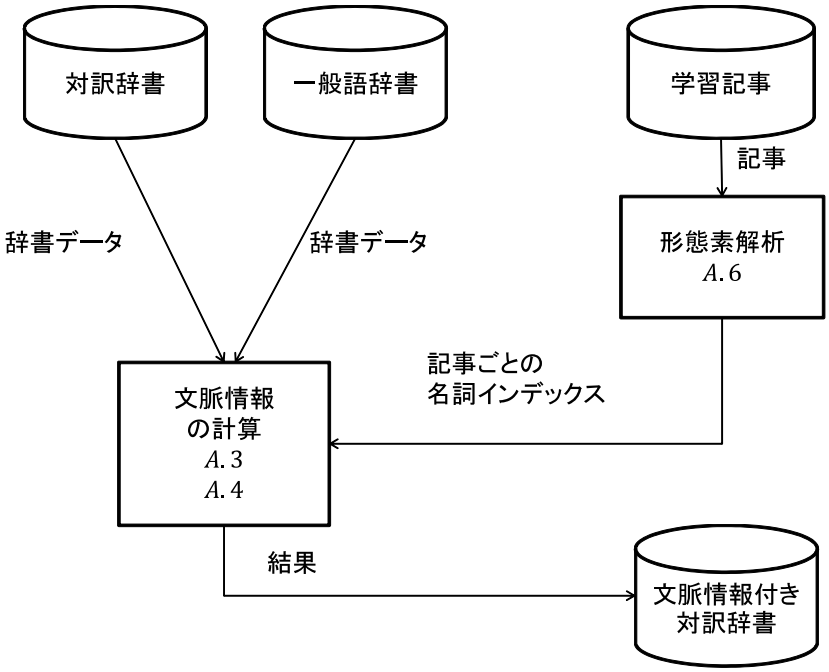

図 A.2: システム構成図（CleanerClient.java の詳細）

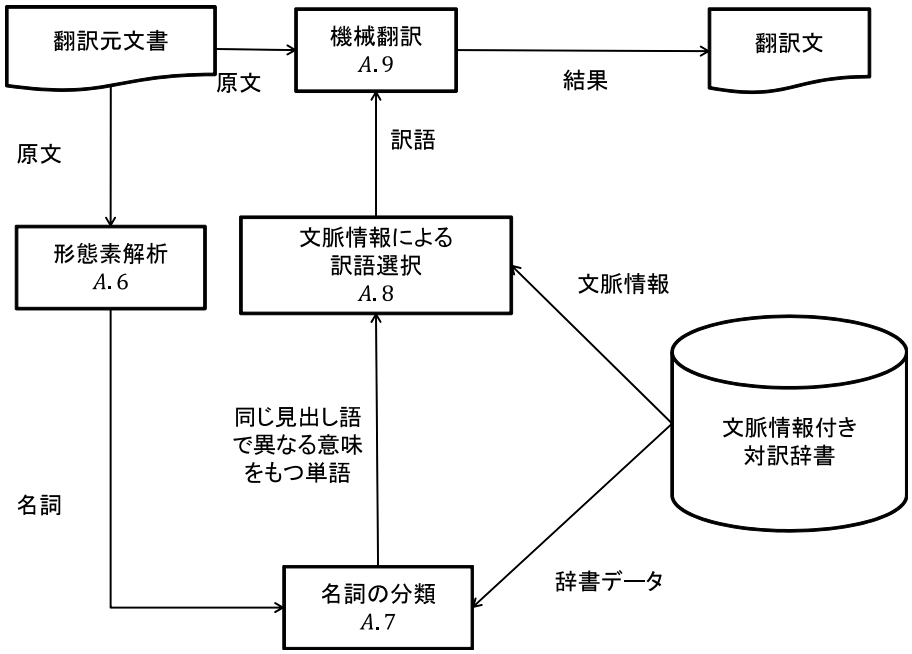

図 A.3: システム構成図（SelectionClient.java の詳細）

A.1 WordSelection.java

```
package main;
import java.io.File;
public class WordSelection {
    public static void main(String[] args) {
        // 対訳辞書から同字異義語を取り出し，文脈情報として SR 表を作成
する
        CleanerClient cc = new CleanerClient();
        cc.cleanAndCalculateSRTable();
        // 原文を読み取り，複数訳語候補をもつ単語の訳語選択を行い，翻
訳する
        SelectionClient sc = new SelectionClient();
        sc.manageInputFile();
    }
}
```

A.2 CleanerClient.java

```
package main;
import java.io.File;
public class CleanerClient {
    public void cleanAndCalculateSRTable() {
        /* ファイル宣言等 */
        File inputDict =
        new File("data/SRcal/beforedict/beforedict.txt");
        File outputOverlappingDict = new File(
                "data/SRcal/afterdict/overlappingdict.txt");
        File outputOthersDict =
        new File("data/SRcal/afterdict/othersdict.txt");
        File outputGeneralNoun = new File(
                "data/SRcal/afterdict/generalnouns.txt");
        File sr = new File("data/SRcal/sr/sr.csv");
        /* 辞書分割開始 */
         DictMaker dictMaker = new DictMaker();
         System.out.println("辞書作成開始\n");
         dictMaker.makeDict(inputDict, outputOverlappingDict,
         outputOthersDict);
         System.out.println("辞書作成完了\n");
        /* SR 事前計算開始 */
        System.out.println("SR 計算開始");
        SRTableMaker table = new SRTableMaker();
        table.exec(sr, outputOverlappingDict, outputOthersDict,
                outputGeneralNoun);
        System.out.println("SR 計算終了");
    }
    // ディレクトリ内のファイルを取得する
    private File[] getInputFiles(File inputRootDir) {
        return inputRootDir.listFiles();
    }
```

```
}
```

A.3 DictMaker.java

```
package main;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;
public class DictMaker {
    // 同字異義語のみの辞書を作る
    public void makeDict(File inputDict, File outputOverlappingDict,
            File outputOthersDict) {
        List<String> allRecords = new ArrayList();
        List<String> overlappingRecords = new ArrayList();
        List<String> otherRecords = new ArrayList();
        System.out.println("辞書読み込み開始\n");
        allRecords = readRecords(inputDict);
        System.out.println("辞書読み込み完了\n");
        System.out.println("同じ見出し語調査中\n");
        overlappingRecords = getOverlappingRecords(allRecords,
        overlappingRecords);
        overlappingRecords = normalize(overlappingRecords);
        System.out.println("同じ見出し語調査完了\n");
        otherRecords = removeOverlappingRecords(allRecords,
        overlappingRecords);
        System.out.println("同じ見出し語辞書とその他の辞書書き込み中
\n");
        writeFile(overlappingRecords, outputOverlappingDict);
        writeFile(otherRecords, outputOthersDict);
```

```
}
// overlapping.txt を csv に変換する
private void makecsv(File outputOverlappingDict) {
    try {
        BufferedReader br = new BufferedReader(new FileReader(
                outputOverlappingDict));
        String line = "";
        File csv = new File("data/SRcal/afterdict/overlap.csv");
        BufferedWriter bw = new BufferedWriter(new FileWriter(csv));
        while ((line = br.readLine()) != null) {
            String[] ary = line.split("\t", 2);
            String ja = ary[0];
            String en = ary[1];
            String newline = ja + "," + en;
            bw.write(newline);
            bw.newLine();
        }
        br.close();
        bw.close();
    } catch (IOException e) {
        System.out.println(e);
    }
}
// csv を txt に変換する
private void changetype(File outputOverlappingDict) {
    try {
        File csv = new File("data/SRcal/afterdict/overlapp.csv");
        // CSV データファイル
        BufferedReader br = new BufferedReader(new FileReader(csv));
        String line = "";
        BufferedWriter bw = new BufferedWriter(new FileWriter(
                outputOverlappingDict));
```

```
            while ((line = br.readLine()) != null) {
                String[] ary = line.split(",", 2);
                String ja = ary[0];
                String en = ary[1];
                String newline = ja + "\t" + en;
                bw.write(newline);
                bw.newLine();
            }
            br.close();
            bw.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
    // 1行ずつ書き込む
    private void writeFile(List<String> records, File outputDict) {
        try {
            BufferedWriter bw = new BufferedWriter(
            new FileWriter(outputDict));
            for (String record : records) {
                bw.write(record);
                bw.newLine();
            }
            bw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    private List<String> normalize(List<String> Records) {
        List<String> formalArticleRecords = new ArrayList();
        for (String formalArticleRecord : Records) {
            if (!formalArticleRecords.contains(formalArticleRecord)) {
```

```
            formalArticleRecords.add(formalArticleRecord);
        }
    }
    return formalArticleRecords;
}
private List<String> removeOverlappingRecords(
List<String> allRecords, List<String> overlappingRecords) {
    for (String overlapping : overlappingRecords) {
        int j = allRecords.size();
        for (int i = 0; i < j; i++) {
            if (allRecords.get(i).equals(overlapping)) {
                allRecords.remove(i);
                j--;
                i--;
            }
        }
    }
    return allRecords;
}
private List<String> getOverlappingRecords(
List<String> allRecords, List<String> overlappingRecords) {
    for (int i = 1; i < allRecords.size(); i++) {
        System.out.println(i);
        String[] recordsAry = allRecords.get(i).split("\t");
        String enRecord = recordsAry[1];
        for (int j = i + 1; j < allRecords.size(); j++) {
            String[] recordsAry2 = allRecords.get(j).split("\t");
            String enRecord2 = recordsAry2[1];
            if (enRecord.equals(enRecord2)) {
                overlappingRecords.add(allRecords.get(i));
                overlappingRecords.add(allRecords.get(j));
            }
```

```
        }
    }
    return overlappingRecords;
}
// 見出し語を調査する
private List<String> serchArticleHeadWords(
List<String> allRecords,
        File enjaArticle) {
    List<String> articleRecords = new ArrayList();
    for (String articleRecord : allRecords) {
        String articleRecordAry[] = articleRecord.split("\t");
        String jaRecord = articleRecordAry[0];
        try {
            FileReader in = new FileReader(enjaArticle);
            BufferedReader br = new BufferedReader(in);
            String line;
            while ((line = br.readLine()) != null) {
                if (line.contains(jaRecord)) {
                    articleRecords.add(articleRecord);
                }
            }
            br.close();
            in.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
    return articleRecords;
}
//ファイル読み込み
private List<String> readRecords(File inputDict) {
    List<String> allHeadWords = new ArrayList();
```

```
        FileReader in;
        try {
            in = new FileReader(inputDict);
            BufferedReader br = new BufferedReader(in);
            String line;
            while ((line = br.readLine()) != null) {
                allHeadWords.add(line);
            }
            br.close();
            in.close();
        } catch (IOException e) {
            System.out.println(e);
        }
        return allHeadWords;
    }
}
```

A.4 SRTableMaker.java

```
package main;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import solve.HC;
public class SRTableMaker {
    private ArrayList<String> overlappingWords = new ArrayList();
    private ArrayList<String> othersWords = new ArrayList();
```

```
private ArrayList<String> generalWords = new ArrayList();
private ArrayList<String> allWords = new ArrayList();
// ベクトル生成および SRTable の作成を行う
public void exec(File sr, File outputOverlappingDict,
        File outputOthersDict, File outputGeneralNoun) {
    this.overlappingWords = makeNounList(outputOverlappingDict,
            this.overlappingWords);
    this.othersWords =
    makeNounList(outputOthersDict, this.othersWords);
    this.generalWords = readGeneralWords(outputGeneralNoun,
            this.generalWords);
    this.allWords.addAll(this.generalWords);
    this.allWords.addAll(this.othersWords);
    operateVector(this.overlappingWords);
    operateVector(this.allWords);
    int num1 = this.overlappingWords.size();
    int num2 = this.allWords.size();
    Double[][] score = new Double[num1][num2];
    score =
    makeSRTable(this.overlappingWords, this.allWords, score);
    makeTableFile(sr, score);
}
// ベクトル計算を始める
private void operateVector(ArrayList<String> words) {
    for (String word : words) {
        makeVectorFile(word);
    }
}
// ベクトルファイル作成
public void makeVectorFile(String noun) {
    File vectorFile = new File("data/vector/" + noun + ".txt");
    if (!vectorFile.exists()) {
```

```
            ArrayList<Double> vectors = new ArrayList();
            System.out.println(noun + "のベクトル生成中");
            vectors = calculateVector(noun);
            try {
                BufferedWriter bw =
                new BufferedWriter(new FileWriter(
                        vectorFile));
                for (int i = 0; i < vectors.size(); i++) {
                    bw.write(vectors.get(i).toString());
                    bw.newLine();
                }
                bw.close();
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }
    // 実際にベクトルを計算する
    private ArrayList<Double> calculateVector(String noun) {
        double df = 0;
        double allnum = 0;
        ArrayList<Double> tfscore = new ArrayList<Double>();
        ArrayList<Double> tfidfscore = new ArrayList<Double>();
        ArrayList<Double> vectors = new ArrayList<Double>();
        double tfidfscoresize = 0;
        // tf と df の計算
        for (int articleNum = 1; articleNum <= 124; articleNum++) {
            String number = String.format("%03d", articleNum);
            try {
                FileReader in = new FileReader("data/jawikidata"
                        + "/jawiki-article" + number + ".txt");
                BufferedReader br = new BufferedReader(in);
```

```
            String tmp;
            while ((tmp = br.readLine()) != null) {
                double tf = 0;
                if (!(tmp.matches(""))) {
                    String[] cut = tmp.split("&&&\\s", 2);
                    if (cut.length >= 2) {
                        String[] nouns = cut[1].split(" ");
                        if (nouns != null) {
                            for (String n : nouns) {
                                if (n.matches(noun)) {
                                    tf++;
                                }
                            }
                        }
                        allnum++;
                        if (tf > 0) {
                            df++;
                            tf = 1 + Math.log(tf);
                        }
                        tfscore.add(tf);
                    }
                }
            }
            br.close();
            in.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
    // ベクトルの計算
    for (double tfvalue : tfscore) {
        double vector = 0;
```

```
            if (df != 0) {
                vector = tfvalue * Math.log(allnum / df);
            } else {
                vector = 0;
            }
            vectors.add(vector);
        }
        return vectors;
    }
    // SR表の作成
    private Double[][] makeSRTable(ArrayList<String> overlappings,
            ArrayList<String> others, Double[][] score) {
        int i = 0;
        int j = 0;
        for (String overlapping : overlappings) {
            for (String other : others) {
                System.out.println(overlapping + ":" +
                other + ":" + i + j);
                score[i][j] = calculateSR(overlapping, other);
                j++;
            }
            i++;
            j = 0;
        }
        return score;
    }
    // SR  の計算
    public double calculateSR(String overlapping, String other) {
        List<Double> vec1 = new ArrayList<Double>();
        List<Double> vec2 = new ArrayList<Double>();
        vec1 = readVector(overlapping);
        vec2 = readVector(other);
```

```
    double vec1size = 0;
    double vec2size = 0;
    vec1size = calculateSize(vec1);
    vec2size = calculateSize(vec2);
    double multisize = 0;
    multisize = vec1size * vec2size;
    // vectorの内積を計算する
    double inner = 0;
    for (int k = 0; k < vec1.size(); k++) {
        inner += vec1.get(k) * vec2.get(k);
    }
    double srValue = 0;
    if (multisize <= 0) {
        srValue = 0;
    } else {
        srValue = inner / multisize;
    }
    return srValue;
}
// ベクトルファイルから読み込む
private List<Double> readVector(String noun) {
    ArrayList<Double> vec1 = new ArrayList<Double>();
    try {
        FileReader in =
        new FileReader("data/vector/" + noun + ".txt");
        BufferedReader br = new BufferedReader(in);
        String line;
        while ((line = br.readLine()) != null) {
            double value = Double.valueOf(line);
            vec1.add(value);
        }
        br.close();
```

```
            in.close();
        } catch (IOException e) {
            System.out.println(e);
        }
        return vec1;
    }
    // ベクトルの大きさを返す
    private double calculateSize(List<Double> vec1) {
        double tmpvec1size = 0;
        double vec1size = 0;
        for (double vector : vec1) {
            tmpvec1size += vector * vector;
        }
        vec1size = Math.sqrt(tmpvec1size);
        if (tmpvec1size == 0) {
            vec1size = 0;
        }
        return vec1size;
    }
    // 実際にSR表をファイルに出力する
    private void makeTableFile(File sr, Double[][] score) {
        try {
            BufferedWriter bw =
            new BufferedWriter(new FileWriter(sr));
            // 1行目書き込み
            bw.write("SR" + ",");
            for (int i = 0; i < this.allWords.size() - 1; i++) {
                bw.write(this.allWords.get(i) + ",");
            }
            bw.write(this.allWords.get(this.allWords.size() - 1));
            bw.newLine();
            // 1行目終わり
```

```
            // 2行目以降
            for (int i = 0; i < this.overlappingWords.size(); i++) {
                bw.write(this.overlappingWords.get(i) + ",");
                for (int j = 0; j < this.allWords.size() - 1; j++) {
                    Double srValue = score[i][j];
                    String str = "Null";
                    if (!(srValue == null)) {
                        str = srValue.toString();
                    }
                    bw.write(str + ",");
                }
                Double lastsr = score[i][this.allWords.size() - 1];
                bw.write(lastsr.toString());
                bw.newLine();
            }
            // 書き込み終わり
            bw.close();
        } catch (FileNotFoundException e) {
            // Fileオブジェクト生成時の例外捕捉
            e.printStackTrace();
        } catch (IOException e) {
            // BufferedWriterオブジェクトのクローズ時の例外捕捉
            e.printStackTrace();
        }
    }
    // 名詞一覧を作成する
    private ArrayList<String> makeNounList(File allNounWords,
            ArrayList<String> words) {
        try {
            FileReader in = new FileReader(allNounWords);
            BufferedReader br = new BufferedReader(in);
            String line;
```

```
            while ((line = br.readLine()) != null) {
                words.add(line.split("\t")[0]);
            }
            br.close();
            in.close();
        } catch (IOException e) {
            System.out.println(e);
        }
        return words;
    }
    private ArrayList<String> readGeneralWords(File outputOthersDict,
            ArrayList<String> generals) {
        try {
            FileReader in = new FileReader(outputOthersDict);
            BufferedReader br = new BufferedReader(in);
            String line;
            while ((line = br.readLine()) != null) {
                generals.add(line);
            }
            br.close();
            in.close();
        } catch (IOException e) {
            System.out.println(e);
        }
        return generals;
    }
}
```

A.5 SelectionClient.java

```
package main;
import java.io.BufferedReader;
import java.io.BufferedWriter;
```

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import jp.go.nict.langrid.client.ws_1_2.TranslationClient;
public class SelectionClient {
    // 入出力ファイル管理
    public void manageInputFile() {
        File inputDir = new File("data/trans/original/");
        File outputDir = new File("data/trans/result/");
        for (File inputFile : getInputFiles(inputDir)) {
            File outputFile = new File(outputDir.getPath()
                    + "/"
                    + inputFile.getName().substring(0,
                            inputFile.getName().length() - 4) +
                             ".txt");
            selectTransWord(inputFile, outputFile);
        }
    }
    private File[] getInputFiles(File inputRootDir) {
        return inputRootDir.listFiles();
    }
    private void selectTransWord(File inputFile, File outputFile) {
        List<String> nouns = new ArrayList<String>();
        // List<String> keywords = new ArrayList<String>();
        String keyword = "";
        List<String> others = new ArrayList<String>();
        List<String> tothers = new ArrayList<String>();
        List<String> candidates = new ArrayList<String>();
```

```
List<String> selectionResults = new ArrayList<String>();
// 原文を読み込む
String orig = readOriginalSentences(inputFile);
System.out.println("----------------------------------"
        + inputFile.getName() +
         "----------------------------------");
System.out.println("原文：" + orig + "\n");
// 形態素解析し，原文から名詞を取得しリストに入れる
NounSearcher ns = new NounSearcher();
nouns = ns.getNouns(orig, nouns);
// 訳語候補の抜出・同文共起した名詞単語の抜出（日本語化）
DictAnalyser da = new DictAnalyser();
keyword = da.getEnKeywords(nouns);
System.out.println("keyword:" + keyword + "\n");
if (keyword == "") {
    System.out.println("keyword がありませんでした");
    return;
}
others = da.removesamerecord(nouns, keyword);
tothers = da.gettrans(others);
System.out.println(others);
System.out.println("同文共起した名詞の訳語群:" +
 tothers + "\n");
// 訳語候補からの SR に基づく選択
candidates.clear();
candidates = da.getCandidates(keyword, candidates);
System.out.println(keyword + "の訳語候補：" + candidates);
String result = null;
// SR 表から文脈情報を取り出す
TableManager tm = new TableManager();
result = tm.selectWordfromTable(tothers, candidates);
double highScore = 0;
```

```
        highScore = tm.getMax();
        selectionResults.add(result);
        System.out.println("\n" + "keyword:「" + keyword +
         "」に対する訳語選択結果："
                + selectionResults + "\n");
        String resultWithSR = "";
        String normalResult = "";
        // 置き換えて翻訳
        BDTranslationClient tc = new BDTranslationClient();
        try {
            resultWithSR = tc.translate(orig, keyword, result);
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            normalResult =
            tc.translate(orig, keyword, candidates.get(0));
        } catch (Exception e) {
            e.printStackTrace();
        }
        makeResultFile(outputFile, orig, keyword, candidates, result,
                resultWithSR, normalResult, highScore);
        System.out.println("\n\n");
    }
    // 結果をファイルに書き込む
    private void makeResultFile(
            File outputFile, String orig, String keyword,
            List<String> candidates, String result,
            String resultWithSR,
            String normalResult, double highScore) {
        try {
            FileWriter fw = new FileWriter(outputFile);
```

```
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write("（原文）");
            bw.newLine();
            bw.write(orig);
            bw.newLine();
            bw.newLine();
            bw.write("（文脈情報に基づいた翻訳結果）");
            bw.newLine();
            bw.write(resultWithSR);
            bw.newLine();
            bw.newLine();
            bw.write("（上位のレコードを優先した翻訳結果）");
            bw.newLine();
            bw.write(normalResult);
            bw.newLine();
            bw.newLine();
            bw.write("keyword「" + keyword + "」に対する訳語候補" +
             candidates);
            bw.newLine();
            bw.write("文脈情報に基づくハイスコアの候補：" + result +
             "スコア：" + highScore);
            bw.newLine();
            bw.write("最上位のレコード：" + candidates.get(0));
            bw.newLine();
            bw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    // 原文を読み込む
    private String readOriginalSentences(File inputFile) {
        String sentence = null;
```

```
        FileReader in;
        try {
            in = new FileReader(inputFile);
            BufferedReader br = new BufferedReader(in);
            sentence = br.readLine();
            br.close();
            in.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return sentence;
    }
}
```

A.6 NounSearcher.java

```
package main;
import static jp.go.nict.langrid.language.ISO639_1LanguageTags.en;
import java.util.ArrayList;
import java.util.List;
import jp.go.nict.langrid.language.Language;
import jp.go.nict.langrid.ws_1_2.morphologicalanalysis.Morpheme;
import triple.MultilingualMorphologicalAnalysisClient;
//原文から名詞を抽出する
public class NounSearcher {
    private Language[] languages = new Language[1];
    private static MultilingualMorphologicalAnalysisClient
    analysisClient;
    public List<String> getNouns(String orig, List<String> nouns) {
        languages[0] = en;
        analysisClient =
        new MultilingualMorphologicalAnalysisClient();
        Morpheme[] mp =
```

```
            analysisClient.analyze(languages[0], orig);
            for (int j = 0; j < mp.length; j++) {
                if (mp[j].getPartOfSpeech().matches("noun.common")
                        || mp[j].getPartOfSpeech().matches("noun")
                        || mp[j].getPartOfSpeech().matches("noun.proper"))
                {
                    nouns.add(mp[j].getLemma());
                }
            }
            nouns = normalize(nouns);
            return nouns;
        }
        //リスト内の重複削除
        private List<String> normalize(List<String> articleRecords) {
            List<String> formalArticleRecords = new ArrayList();
            for(String formalArticleRecord : articleRecords){
                if(!formalArticleRecords.contains(formalArticleRecord)){
                    formalArticleRecords.add(formalArticleRecord);
                }
            }
            return formalArticleRecords;
        }
    }
```

A.7 DictAnalyser.java

```
package main;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
```

```
import java.io.ObjectInputStream;
import java.util.ArrayList;
import java.util.List;
public class DictAnalyser {
    List<String> noRecords = new ArrayList<String>();
    public String getEnKeywords(List<String> nouns) {
        String keyword = "";
        for (String noun : nouns) {
            if (containinOverlapping(noun)) {
                keyword = noun;
            }
            if (!containinGeneralDict(noun)
            && !containinTechDict(noun)) {
                this.noRecords.add(noun);
            }
        }
        return keyword;
    }
    // 同字異義語かどうかの真偽値を返す
    private boolean containinOverlapping(String noun) {
        boolean bool = false;
        File inputFile =
        new File("data/SRcal/afterdict/overlappingdict.txt");
        FileReader in;
        try {
            in = new FileReader(inputFile);
            BufferedReader br = new BufferedReader(in);
            String line;
            while ((line = br.readLine()) != null) {
                String ja = line.split("\t")[0];
                String en = line.split("\t")[1];
                if (en.equals(noun)) {
```

```
                bool = true;
                break;
            }
        }
        br.close();
        in.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return bool;
}
// 一般語辞書に含まれているかの真偽を返す
private boolean containinGeneralDict(String noun) {
    String JEDict = "data\\trans\\generalDict\\GeniusJE";
    String Tri =
    "data\\trans\\generalDict\\triple_jed_all_arraydata";
    ArrayList jeDict = new ArrayList();
    try {
        ObjectInputStream in =
        new ObjectInputStream(new FileInputStream(Tri));
        in = new ObjectInputStream(new FileInputStream(JEDict));
        jeDict = (ArrayList) in.readObject();
    } catch (ClassNotFoundException e) {
        System.out.println(e);
    } catch (IOException e) {
        System.out.println(e);
    }
    for (int i = 0; i < jeDict.size(); i++) {
        Object test = jeDict.get(i);
        String str = test.toString();
        String strAry[] = str.split(", ");
        String generalJa = strAry[0].substring(
```

```
            1, strAry[0].length());
            String generalEn = strAry[strAry.length - 1].substring(
            0,strAry[strAry.length - 1].length() - 1);
            if (noun.equals(generalEn)) {
                return true;
            }
        }
        return false;
    }
    // 対訳辞書に含まれているかの真偽を返す
    private boolean containinTechDict(String noun) {
        boolean bool = false;
        File inputFile = new File(
        "data/SRcal/beforedict/beforedict.txt");
        FileReader in;
        try {
            in = new FileReader(inputFile);
            BufferedReader br = new BufferedReader(in);
            String line;
            while ((line = br.readLine()) != null) {
                String ja = line.split("\t")[0];
                String en = line.split("\t")[1];
                if (en.equals(noun)) {
                    bool = true;
                    break;
                }
            }
            br.close();
            in.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    return bool;
}
// 重複したレコードを削除する
public List<String> removesamerecord(List<String> nouns,
 String keyword) {
    nouns.remove(nouns.indexOf(keyword));
    for (String norecord : noRecords) {
        nouns.remove(nouns.indexOf(norecord));
    }
    return nouns;
}
// 訳語を得る
public List<String> gettrans(List<String> others) {
    List<String> tothers = new ArrayList<String>();
    String JEDict = "data\\trans\\generalDict\\GeniusJE";
    String Tri =
    "data\\trans\\generalDict\\triple_jed_all_arraydata";
    ArrayList jeDict = new ArrayList();
    try {
        ObjectInputStream in = new ObjectInputStream(
        new FileInputStream(Tri));
        in = new ObjectInputStream(new FileInputStream(JEDict));
        jeDict = (ArrayList) in.readObject();
    } catch (ClassNotFoundException e) {
        System.out.println(e);
    } catch (IOException e) {
        System.out.println(e);
    }
    for (String other : others) {
        // int count = 0;
        // String onlyOne = null;
        for (int i = 0; i < jeDict.size(); i++) {
```

```
            Object test = jeDict.get(i);
            String str = test.toString();
            String strAry[] = str.split(", ");
            String generalJa = strAry[0].substring(
            1, strAry[0].length());
            String generalEn = strAry[strAry.length - 1].substring(
            0,strAry[strAry.length - 1].length() - 1);
            // if (other.equals(generalEn)) {
            // count++;
            // onlyOne = generalJa;
            // }
            // }
            // if (count == 1) {
            // tothers.add(onlyOne);
            // }
            if (other.equals(generalEn)) {
                tothers.add(generalJa);
            }
        }
    }
    // 同文共起した単語の訳語として対訳辞書のものも含める
    File inputFile = new File(
    "data/SRcal/beforedict/beforedict.txt");
    FileReader in;
    try {
        in = new FileReader(inputFile);
        BufferedReader br = new BufferedReader(in);
        String line;
        while ((line = br.readLine()) != null) {
            String ja = line.split("\t")[0];
            String en = line.split("\t")[1];
            for (String other : others) {
```

```
                if (en.equals(other)) {
                    tothers.add(ja);
                }
            }
        }
        br.close();
        in.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return tothers;
}
// 訳語候補を得る
public List<String> getCandidates(String keyword,
 List<String> candidates) {
    File inputFile = new File(
    "data/SRcal/afterdict/overlappingdict.txt");
    FileReader in;
    try {
        in = new FileReader(inputFile);
        BufferedReader br = new BufferedReader(in);
        String line;
        while ((line = br.readLine()) != null) {
            String ja = line.split("\t")[0];
            String en = line.split("\t")[1];
            if (en.equals(keyword)) {
                candidates.add(ja);
            }
        }
        br.close();
        in.close();
    } catch (IOException e) {
```

```
            e.printStackTrace();
        }
        return candidates;
    }
}
```

A.8 TableManager.java

```
package main;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;
public class TableManager {
    private double max;
    //訳語候補のハイスコアを返す
    public double getMax() {
        return this.max;
    }
    // SRtableを参照しスコアの最も高い訳語候補を得る
    public String selectWordfromTable(List<String> tothers,
            List<String> candidates) {
        List<Double> srScores = new ArrayList<Double>();
        double srScore = 0;
        for (String candidate : candidates) {
            srScore = 0;
            for (String tother : tothers) {
                srScore += fetchScore(tother, candidate);
            }
```

```
            srScores.add(srScore);
            System.out.println(candidate + "のスコア:" + srScore);
        }
        String result = null;
        result = candidates.get(elect(srScores));
        return result;
    }
    // スコアが最大値となる候補のナンバーを返す
    private int elect(List<Double> srScores) {
        double max = srScores.get(0);
        int maxNum = 0;
        if (srScores.size() == 1) {
            return 0;
        }
        for (int i = 1; i < srScores.size(); i++) {
            if (srScores.get(i) > max) {
                this.max = srScores.get(i);
                maxNum = i;
            }
        }
        return maxNum;
    }
    // Tableから指定のスコアを抜出しリストに格納
    private double fetchScore(String tother, String candidate) {
        double score = 0;
        try {
            File csv = new File("data/SRcal/sr/sr.csv");
             // CSVデータファイル
            BufferedReader br = new BufferedReader(
            new FileReader(csv));
            // 最終行まで読み込む
            int gyosuu = 1;
```

```
        String line = "";
        int cNum = 1;
        int tNum = 0;
        // 1行目
        line = br.readLine();
        for (int i = 0; i < line.split(",").length; i++) {
            if (tother.equals(line.split(",")[i])) {
                tNum = i;
            }
        }
        // 2行目から
        while ((line = br.readLine()) != null) {
            if (candidate.equals(line.split(",")[0])) {
                cNum = gyosuu;
            }
            gyosuu++;
        }
        br.close();
        File csv2 = new File("data/SRcal/sr/sr.csv");
        // CSVデータファイル
        BufferedReader br2 = new BufferedReader(
        new FileReader(csv));
        String line2 = "";
        for (int i = 0; i < cNum; i++) {
            line2 = br2.readLine();
        }
        line2 = br2.readLine();
        String tmpscore = line2.split(",")[tNum];
        score = Double.parseDouble(tmpscore);
        br2.close();
    } catch (FileNotFoundException e) {
        // Fileオブジェクト生成時の例外捕捉
```

```
            e.printStackTrace();
        } catch (IOException e) {
            // BufferedReader オブジェクトのクローズ時の例外捕捉
            e.printStackTrace();
        }
        return score;
    }
}
```

A.9 BDTranslationClient.java

```
package main;
import static jp.go.nict.langrid.language.ISO639_1LanguageTags.en;
import static jp.go.nict.langrid.language.ISO639_1LanguageTags.ja;
import ishidajTranslator.IshidaJTranslatorWithTemporalDict;
import java.net.URL;
import java.util.Collection;
import jp.go.nict.langrid.client.ws_1_2.*;
import jp.go.nict.langrid.commons.cs.binding.BindingNode;
import jp.go.nict.langrid.ws_1_2.bilingualdictionary.Translation;
public class BDTranslationClient {
    //言語グリッドの一時辞書機械翻訳サービスを呼び出し，翻訳結果を返す
    public String translate(String orig, String keyword, String trans)
            throws Exception {
        Translation[] dict = new Translation[1];
        dict[0] = new Translation(keyword, new String[] { trans });
        IshidaJTranslatorWithTemporalDict j2 = new IshidaJTranslatorWith
        TemporalDict(
        "http://langrid.nict.go.jp/service_manager/invoker/
        kyotou.langrid:TranslationCombinedWithBilingualDictionary");
        j2.setTranslator("KyotoUJServer");
        String result = j2.translate("en", "ja", orig, dict);
        return result;
```

```
    }
}
```