格解析は、構文解析木から格構造と呼ばれる構造を作り出す処理である。この処理は意味的な情報を利用するので、意味解析と呼ぶ場合もある。格解析の結果は動詞を中心とした構造であり、動詞に対して他の要素がどのような格(動作主格、対象格、源泉格、道具格等)を示すかを明らかにする。格解析の際に中心的な役割を果たすのが、動詞の格フレーム辞書である。この辞書には、それぞれの動詞がどのような格を選択するのかが記述されている。動詞は、複数の意味(語義)を持っているが、格フレーム辞書ではどのような格要素がどのような助詞を伴って現れるか、またそれぞれの格に対して意味マーカと呼ぶ制限(人、物、量、数)を設ける。意味マーカは単語の表す意味を大まかに分類した体系である。前段階で複数候補が得られた場合、格解析の結果、矛盾する内容に関しては除外するなどの処理により、候補を絞れる場合がある。
変換処理は、格解析された文の格構造を、対象言語の格構造に変換する処理である。ここでの主たる処理は、訳語選択と、構文変換である。訳語を選択する場合には、格情報を元に動詞の選択を行う等の工夫が必要とされる。構文変換では、あらかじめ2つの言語間の構文的な変更規則を用意しておき、この規則を適用することで実現される。
生成処理は、変換が終了した構造から、訳語の語順を決定し、各種の調整(冠詞、複数形、受け身等)を行った上で、最終的な変換先言語の訳文を生成する。
以上が、自動翻訳を例とした、自然言語の解析処理の流れである。

◆ 1-4 自動点訳システムの構造
自動点訳システムは、前述の自然言語処理の分野の中では自動翻訳に近いといえる。しかし変換先の言語はまったく別の言語ではなく、点字という表現形式ではあっても、わかち書きされた表音に極めて近い日本語である。このため日英自動翻訳等と比べると、処理の複雑さは軽微といえる。とは言っても、日本語という非常に大きな母集団を扱う訳であるから、膨大な量の日本語の知識を投入することで始めて実現できる本格的なシステムが要求される。

◆ 1-5 EXTRA 自動点訳システム
EXTRA 自動点訳システムの処理は、自動翻訳が5ステップであったのに対して、主として次の2つの内部処理に大別できる。
・ 形態素解析
・ 生成処理
EXTRA 自動点訳システムの内部構造は形態素解析と、生成処理をほぼ同時に進めるような内部構造となっており、厳密な分離は難しいが、論理的には、この2つの処理が行われているといえる。
この内、生成処理については、音引き、分かち書き、数符などの点字固有の各種規則を適用する部分である。自動点訳の変換精度を考える場合、生成処理は比較的に定型的な処理であり、品詞情報が正確に求まれば、多くの場合に正確な生成処理が行える。したがって形態素解析の処理結果が、変換精度を大きく左右することになる。本報告書は、この点に着目し、次世代型の自動点訳システムにおいては、どのような形態素解析の処理方法がふさわしいかについて調査したものである。

□ 第2章 従来型の形態素解析システム
ここでは、1990年代前半までの従来型の手法に基づく形態素解析システムについて述べる。現在の EXTRA 自動点訳システムで用いられている構造は、従来型の形態素解析システムの枠内に収まっているといえる。従来型の形態素解析システムの課題については、そのまま EXTRA 自動点訳システムにも当てはまる部分が多い。

◆ 2-1 人手によるパラメータ推定と精度の限界
日本国内では、1982年から1986年にかけて京都大学長尾研究室を中心に、Mu プロジェクトと呼ぶ機械翻訳プロジェクトが行われ、日英および英日の2つの機械翻訳システムが実現された。このシステムはその後、形態素解析システム JUMAN に発展した。このシステムの形態素解析は、原言語単語辞書と品詞接続可能表を元に解析を行う方式である。この辞書と表はすべて人手により作成されたものであり、実現には非常に多くの労力と時間が必要であった。その後に作られた機械翻訳システムでも、同様の方法を踏襲したものが主流である。
原言語単語辞書と品詞接続可能表を参照する方法で形態素解析を行うと、多くの場合、複数の解析結果が得られる。一般に、この複数候補はラティス構造と呼ぶデータ構造で示されることが多い。複数解のうち、どの解析結果が適切かを決めるために、様々な方法が考案された。以下に挙げるのは、その主たる方法である。
最長一致法：文頭から長い形態素を優先して解を抽出する。

2 文節最長一致法：文頭から 2 文節ごとの長さが長い解を優先して解を抽出する。
形態素数最小法：形態素数が少ない解を優先する。
文節数最小法：文節数が少ない解を優先する。
コスト最小法：語と語の連続にコストを与えて、総コストの少ない解を優先する。
最長一致法、2 文節最長一致法、形態素数最小法、文節数最小法については、それぞれ有効な場面があるが、有効でない場合には適切な処理が行えないというアルゴリズム面での限界がある。一方、コスト最小法は、その他の方法ではカバーできないケースに関してもコストパラメータを適切に設定すれば、適切な解が得られる可能性があるという面で他の方式と比べて適用範囲が広いと考えられる。
コスト最小法では、語と語の接続の度合いはすべて人手でパラメータを与える方式で実現された。誤解析結果が見つかると、正しい解析結果になるようにそれぞれの語と語の接続の度合いを人手でチューニングする必要がある。コスト最小法は、適用できる範囲が広いというメリットはあるが、これを実現するための辞書のチューニングに膨大な労力が要求されるという問題点を抱えている。さらに、このパラメータは分析対象の文書の分野に依存するため、分野の違いを人手で変更するのが容易ではない。コスト最小法は有望な方法論だが、大量の辞書をメンテナンスしつつ、解析精度を向上させるのは大変な困難が伴った。
さらに、日本語の形態素解析には未定義語の処理に関する課題もある。未定義語が見つかった場合には、おなじカタカナ、ひらがな、漢字の連続を名詞の未定義語として扱う処理が一般的である。しかし、この方法では、たとえば「冬ソナ」「メル友」「ゆるパン」などの新語がうまく処理できないし、「ググる」のような動詞の新語も扱えない。未定義語に関しても人手で辞書登録を行えば、適切に形態素解析を行えるようになるが、人手を介さずに処理を行う方法が大きな課題といえる。

□ 第 3 章 今日的な形態素解析システム

近年(1990 年代後半以降)の、国内の形態素解析に関する自然言語研究では、前述のコスト最小法の人手によるパラメータ推定や、未定義語の人手によるメンテナンスをどのように省力化し、形態素解析システムの精度向上につなげるかが研究テーマの大きな潮流であったといえる。

◆ 3-1 コーパスデータに基づく学習型のパラメータ推定

コスト最小法の語と語の接続コストパラメータを人手で与える方法論は大変な労力を要する欠点がある。また、扱う文書の分野によって、適切なパラメータが異なるという問題も吸収できない。そこで、あらかじめ人手で、わかち書きや品詞分析を行ったコーパス（文例）を大量に用意し、このデータをプログラムに与えて自動的に計算を行い、接続コストパラメータを求める、いわゆる学習型のパラメータ推定方法が提唱された。
パラメータを統計的手法により分析し自動推定する方法についてはいくつかの案が提唱されたが、中でも有力なのが隠れマルコフモデル(Hidden Markov Model, HMM)を用いた方法である。隠れマルコフモデルは、自然言語の文字や単語そのものを状態とみて、これをマルコフ・モデルとみなす。外部で観測されるのは単語の列であるが、その奥に内部状態としての品詞の遷移があると考える。「隠れ」とはこのように外部からは見えない内部状態が存在することを意味する。状態間の遷移は単語・品詞間の条件付き確率で表される。
コーパスデータを隠れマルコフモデルで処理すると、単語・品詞間の条件付き確率が求められる。この確率値を用い、入力の単語列に対して、総出現確率を最大にするような品詞列が、一番確からしい形態素解析結果といえる。このような確率的モデルによる正解率は、人手で規則を与える場合と同等あるいはそれ以上の精度があるとの研究結果(15)(16)がある。
パラメータの自動推定は、品詞の決定が難しい英語のような自然言語で最初に発達したが、日本国内でも日本語に対する有効性が明らかになってきた。さらに、学習型のパラメータ推定で必要となる大量の形態素解析済みのコーパスデータに関しても国内での整備が進んでおり(後述)、これらのコーパスデータを用いた形態素解析システムの研究が進んでいる。

◆ 3-2 形態素解析システムの具体例

ここでは、主たる日本語形態素解析システムの具体例を挙げる。

◇ 3-2-1 JUMAN

JUMAN は前述のように京都大学の長尾真氏の研究室を中心に開発された形態素解析システムであり、1992 年に公開された。現在、京都大学の黒橋貞夫氏の研究室にて改良が続いている。現在の最新版は Version 6.0 にあたり、以下の URL にて公開されている。
http://nlp.kuee.kyoto-u.ac.jp/nl-resource/juman.html

形態素解析のしくみとしては、コスト最小法が用いられている。語と語の接続の度合いのコストはすべて人手でパラメータを与える方式で実現されている。

◇ 3-2-2 ChaSen(茶筌)

ChaSen(茶筌)は奈良先端科学技術大学院大学の松本裕治氏の研究室で開発され、1997 年に公開された形態素解析システムである。JUMAN Version 2.0 を元にして開発が行われ、解析速度と使い勝手の向上を目指している。ChaSen の性能は 15 万語/秒と言われ、新聞記事 1 年分をおよそ 3 分で解析できるスピードである。[山下達雄 1997](17)のようにコーパスを元に統計的手法でパラメータ(確率パラメータ)を得て、茶筌を動作させる研究も行われた。なお、このような ChaSen の隠れマルコフモデルを用いたコスト推定モジュールは奈良先端科学技術大学院大学の外部に対しては現在公開されていない。ChaSen の未知語の取り扱いに関してはハードコーディングされており、自由に定義できない制約がある。

現在以下の URL の[公開ツール・データ]-[自然言語処理ツール]のリンクよりダウンロードできる。

http://cl.aist-nara.ac.jp/

現在の ChaSen の最新バージョンは 2.4.2 である。ChaSen は、日本語辞書として情報処理推進機構によって開発されている IPA 辞書もしくは UniDic を用いる。現在非公開であるが、NAIST-Japanese-dic, NAIST-Chinese-dic(中国語)を用いることもできる。

日本語辞書の UniDic は、国立国語研究所が規定した短単位という揺れがない斉一な単位で設計されている点が大きな特徴である。また、語彙素・語形・書字形・発音形の階層構造を持ち，表記の揺れや語形の変異にかかわらず同一の見出しを与えることがでる。さらに、アクセントや音変化の情報を付与することができ，音声処理の研究に利用できる。UniDic に関する情報は次の URL より入手できる。

http://www.tokuteicorpus.jp/dist/

◇ 3-2-3 MeCab(和布蕪)

MeCab(和布蕪)は京都大学情報学研究科と NTT コミュニケーション科学基礎研究所の共同研究プロジェクトを通じて工藤拓氏が開発し、2002 年に公開した形態素解析システムである。辞書とコーパスに依存しない汎用的な設計と処理速度が JUMAN や ChaSen よりも高速な点が大きな特徴である。日本語辞書として前述の UniDic, IPA 辞書, JUMAN を用いることができる。

MeCab では Conditional Random Fields, CRF という方法を用いてコスト値を推定する。ChaSen のコスト推定モジュールは隠れマルコフモデルによるものであったが、日本語での実用面での困難さがあった。一方の、MeCab では CRF を採用したことで、品詞体系、単語長、辞書の変更に対して柔軟かつ低コストで対応できるようになったのが、大きな特徴である。さらに、CRF は隠れマルコフモデルの 1/3 程度の学習用コーパスで同程度の性能が得られることが明らかとなっている。これは新しい分野に対するパラメータを作成する場合に、少ない分量のコーパスで対応できることを示している。また、ChaSen のコスト推定モジュールは非公開であったのに対し、MeCab では、コスト推定モジュールが標準添付されている。

ChaSen では未知語は固定的な処理しか行えなかったが、MeCab は未知語に対する処理を自由に定義できる構造となっている。また、MeCab の基本的な構造として、字種に基づくわかち書きを行う。この字種そのものの定義や、各字種に対するわかち書きの定義、分かち書きされたものに対してどのように品詞を割り当てるかといった処理をユーザ定義できる、非常に柔軟な構造になっている。さらに、字種を表現するための内部構造は Unicode が用いられている。未知語処理のパラメータも CRF により推定できるシステムとなっている。

◇ 3-2-4 Rosette

Rosette 日本語形態素解析システムは、世界的な規模で様々な言語に関する自然言語処理の製品の開発を行っている米国の Basis Technology が開発した。Google、楽天、Amazaon などで使用されていることで有名である。商用的には、最も成功している日本語形態素解析システムといえる。形態素解析システムの内部構造は非公開である。

◆ 3-3 高度言語情報融合フォーラムによる大規模コーパス

近年の計算機の処理能力の向上により、大量のコーパスデータを処理して、学習型でパラメータを推定するシステムは十分に実用的な段階になってきた。このような取り組みを行う場合に、大量のデータを持った大規模コーパスの整備が必須である。大規模なコーパスを整備するのは、それなりに多くの労力を要する地道な作業が必要になる。大規模コーパスは、形態素解析にとどまらず、音韻解析、音声合成、音声対話システム、自動翻訳、テキスト要約など様々な分野で必要とされている。

一つの小さな組織で、大規模コーパスの整備を進めるのは効率が悪いため、産学官をあげて共同で大規模コーパスを整備しようという機運が高まり、高度言語情報融合フォーラム(ALAGIN)という組織が設立された。

こちらでは会員に対して、研究目的でコーパス、日本語辞書、データベース、音声データ、対話データなどの配布を行っている。

□ 第4章 次世代型自動点訳システムに望まれる形態素解析
ここでは、自動点訳において望まれる形態素解析機能が何かについて明らかにする。

◆ 4-1 動作環境
近年、自動点訳システムはWindowsパソコン以外の利用も要求されるようになってきている。考えられる動作環境としては、次のようなものがある。
・Windows 32bit 環境/64bit 環境
・Windows CE
・Linux (主として組み込みシステム)
・UNIX
・Mac OS X
・iOS(iPhone OS)
つまり、POSIXで規定される範囲のAPIのみを用いて実現している等の、マルチプラットホームで動作できる設計が、自動点訳システムの形態素解析システムにとって重要である。
また、使用できるメモリやCPUリソースなどの要求も厳しいといえる。具体的には、64M程度の使用メモリで高速に形態素解析が動作できることが望ましい。さらにサーバで自動点訳を動作させる場合には、スレッドセーフな構造も要求される。
また、近年の動作環境の傾向として、マルチコアCPUの利用が一般的となってきた。形態素解析の処理に於いて、マルチコアを生かして並列処理を行う等の工夫も、マシン性能を十分に引き出すという点で重要である。

◆ 4-2 形態素解析に望まれる点
次世代自動点訳システムの形態素解析に望まれる点として、専門辞書の実現、メンテナンスの労力低減、Unicode対応などが挙げられる。
前述のように、近年の形態素解析研究は、人手によるコスト付与から、コーパスを用いた自動的なパラメータ推定の方法論に大きくシフトしている。専門辞書に関しては、専門分野のコーパスを作成し自動学習させることで、手動でパラメータを付与するのと比べて、比較的少ない労力で辞書を作成できる。また、メンテナンス労力の面では、解析誤りの文を人手で正しい形態素解析結果に修正し、これを学習コーパスとして自動学習させることで、比較的容易に誤りが軽減されたパラメータのチューニングが実現できる可能性がある。

◆ 4-3 MeCab
本報告書でまとめた形態素解析システムの最先端の研究成果がMeCabというオープンソースの形で公開されているのは着目すべき点である。MeCabは内部Unicodeで実現されているという点でも、自動点訳システムの形態素解析システムとしても申し分ない。また、MeCabは他分野でも現在活躍中の形態素解析システムであるので、他分野での修正結果を取り込み、さらに形態素解析精度を向上できる可能性があり、社会実装性の面でも優れている。
結論として、MeCabは、これまでの日本国内の形態素解析研究の総決算ともいえるシステムであり、自動点訳システムの形態素解析システムとして利用できる可能性が非常に大きいといえる。

□ 第5章 参考文献
(1) 自然言語処理 長尾真編 岩波書店 ISBN4-00-010355-5
(2) チョムスキー入門(生成文法の謎を解く) 町田健 光文社新書 ISBN4-334-03344-X
(3) 自然言語処理の基礎 吉村賢治 サイエンス社 ISBN4-7819-0956-6
(4) アナロジーによる機械翻訳 佐藤理史 共立出版株式会社 ISBN4-320-02854-6
(5) テキストデータの統計科学入門 金明哲 岩波書店 ISBN4-00-005702-8
(6) フリーソフトでつくる音声認識システム 荒木雅弘 森北出版株式会社 ISBN4-627-84711-8
(7) 確率的言語モデル 北研二 東京大学出版会 ISBN4-13-065404-7
(8) 自然言語処理入門 西田豊明 オーム社 ISBN4-274-07468-4
(9) 自然言語処理ことはじめ 荒木健治 森北出版株式会社 ISBN4-627-82851-9
(10) 自然言語処理 天野真家,石崎俊,宇津呂武仁,成田真澄,福本淳一 オーム社 ISBN4-274-20465-4

(11) 自然言語解析の基礎 田中穂積 産業図書 ISBN4-7828-5127-8
(12) パーソナルコンピュータによる機械翻訳プログラムの制作 上野俊夫 ラッセル社 ISBN4-947627-00-X
(13) 自然言語処理入門 黒川利明,東条敏 近代科学社 ISBN4-7649-0143-9
(14) 生物配列の統計(核酸・タンパクから情報を読む) 岸野洋久,浅井潔 岩波書店 ISBN4-00-006849-0
(15) 隠れマルコフモデルによる日本語形態素解析システムのパラメータ推定 竹内孔一 1995
(16) 隠れマルコフモデルによる日本語形態素解析のパラメータ推定 竹内孔一,松本裕治 1997
(17) コスト最小法と確率モデルの統合による形態素解析 山下達雄,松本裕治 1997

B-4 形態素解析エンジン MeCab を使った自動点訳エンジンの開発と評価

MeCab は工藤拓によって開発された、オープンソースの形態素解析エンジンである。
この MeCab を元にして点訳エンジンを作成し、能力を検証した。

点訳の分かち書きの規則は、大きく二つの規則によって成り立っている。
すなわち、自立語は前との間にマスを空け、付属語は前の語との間を続けるという「分かち書きの規則」と、長い名詞は意味のまとまりで区切るという「切れ続きの規則」の二つである。

このうち、「分かち書きの規則」については、MeCab などの形態素解析によって自立語と付属語の判断を行うことができると考えられる。
「切れ続きの規則」については、種々の例外があるものの、主にコーパスによって判断を行うことができると考えられる。

◆MeCab を使った自動点訳エンジンの処理内容

具体的には、MeCab の出力に対し、以下の後処理を行った。

助詞、助動詞、接尾語は前を続ける。
非自立の動詞、名詞、形容詞も前を続ける。
接尾語は種類により、前を空けるか続けるか判断する。これは接尾語は原則は前の語との間を続けるが、点訳規則により、「さん」「氏」「君」は前の語との間にマスを空けるためである。

それ以外の語(主に自立語)は前を空ける。

接頭詞は後を続ける。

記号類は点字の規則にしたがって処理する。

数字、漢数字は読み下し処理を行う。

出力結果に対し、点訳コーパスによる修正を行った。

◆ 実験手順

手順は以前の点訳コーパスの評価の場合と同じである。
20 の文書について点訳をおこない、公開されている自動点訳エンジンのうち、EXTRA と比較を行った。

◆ 結果

1. 分かち書き誤り数
EXTRA 60
MeCab を使った点訳エンジン 106

○分かち書きの誤り例
EXTRA の誤り例
伊豆の国市 いずのくにし いずの くにし
穴掘り あなほり あな ほり
タラの木の たらのきの たらの きの

MeCab を使った点訳エンジンの誤り例
視床下部から ししょーかぶから(正) ししょー かぶから(誤)

クモ膜下腔を　くもまくかくーを(正)　くもまくか　くーを(誤)
アミノバイタルフィールド　あみの　ばいたる　ふぃーるど(正)　あみのばいたるふぃーるど(誤)

1.1 分かち書き誤りのうち、解析の誤りと判断されるもの
EXTRA 8
MeCab を使った点訳エンジン 19

○　解析の誤り例
EXTRA の誤り例
土からすっぽ抜けるし
つちから　すっぽぬけるし(正)　つち　からすっぽ　ぬけるし(誤)

これらが集まり内筋周膜で
あつまり　ないきん　しゅーまくで(正)　あつまりないきんしゅーまくで(誤)

絹糸のようなつやを
きぬいとのよーな　つやを(正)　きぬいとのよーなつやを(誤)

MeCab を使った点訳エンジンの誤り例
長々しい　「しい」が動詞の「しいる」と誤って認識
不自由なれど　「ど」が接頭語の「ど」と誤って認識
行なえなかった　「なかっ」が形容詞の「ない」と誤って認識

2. 読み誤り数
EXTRA 186
MeCab を使った点訳エンジン 244

○　読み誤り例
EXTRA の誤り例
お代の　おだいの(正)　およの(誤)
間ぎわには　まぎわにわ(正)　あいだぎわにわ(誤)
面映い　おもはゆい(正)　めんええい(誤)
天下人　てんかびと(正)　てんかじん(誤)

MeCab を使った点訳エンジンの誤り例
間に　あいだに(正)　まに(誤)
墨字と　すみじと(正)　ぼくじと(誤)
金が　かねが(正)　きんが(誤)
膠原繊維の　こーげんせんいの(正)　にかわ　げんせんいの(誤)

◆　考察
MeCab は形態素解析エンジンであるため、読み分けについてはもともと考慮されていない。必然的に読み誤りは多くなると考えられる。また、未知語が多かったために読み誤りが増加したという要因もあると考えられる。

分かち書きの誤りのうち、形態素的な解析の誤りについては、今回は EXTRA の方が誤りが少なかった。これは、両方の結果とも点訳コーパスに よる修正を行っており、読み誤りがあると点訳コーパスが働かないことも理由の一つではないかと推測している。読みが正しければ、仮に形態素解析的な解析の誤りが存在しても、コーパスによる修正で修正されることもある。

総合的には EXTRA のほうが良い結果を示したが、個別の文章では、携帯小説のような、ひらがなが多い文書では、MeCab の方が良好な結果を示した。

解析の誤り以外の分かち書きの誤りについては、以下のものが存在する。
a.特殊な専門用語や固有名詞であるため、点訳コーパスにヒットしなかった
b.コーパスに続ける用例と切る用例の両方が存在するため、点訳コーパスにヒットしたが、修正されなかった。
c.読み誤りとの複合により、点訳コーパスにヒットしなかった。

B-5 点字コーパスによる分かち書き誤り検出と修正

概要

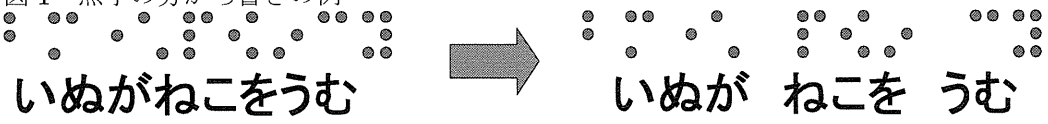
点字コーパスによる点訳分かち書き誤り検出、修正実験結果を報告する。修正対象とする点訳ファイルのある一部分を取り出し、その一部分が大量の点訳ファイル(点訳コーパスと呼ぶ)の中に何件ヒットするかを調べる。そして、そのヒット件数から、分かち書きの誤りを判別し、修正する。実験をおこない、本研究開発システムにより、自動点訳された点訳ファイルを修正したところ、分かち書き誤りの数は31.56%減少した。また、形態素解析エンジンMeCabを使用した自動点訳エンジンを試作し、現在公開されている自動点訳エンジンとの比較を行った。

1.はじめに

1.1 点字について

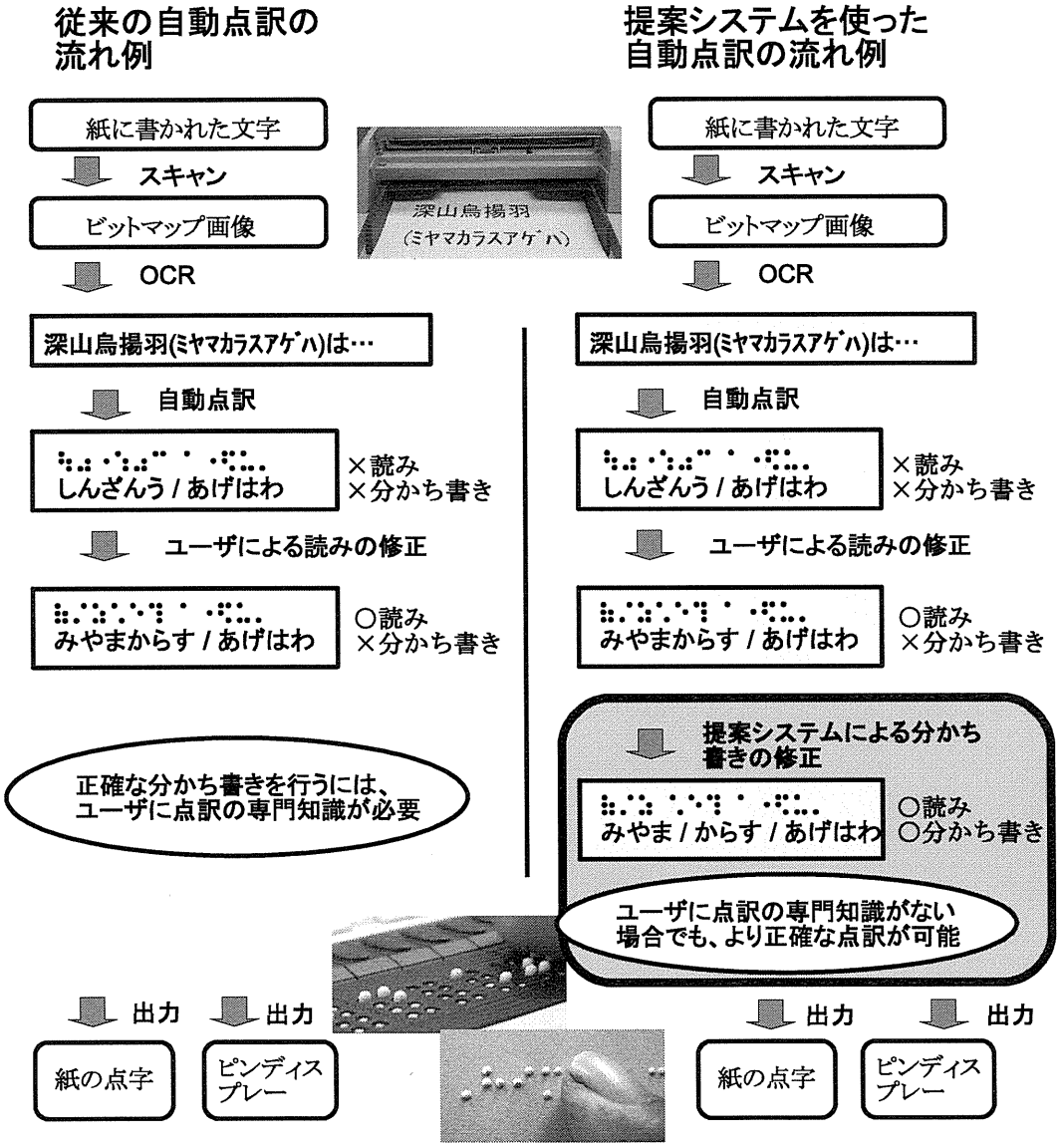
点字は視覚障害者が用いる字である。世界中で一般的な6点式点字は、1文字は6点の有無によって構成されているため、63文字しか表現できない。日本語点字では、漢字は表現できず、カナだけしかない。カナだけでマス空けをせずに書くと分かりにくいし、意味が複数とれることもある。例えば「いぬがねこをうんだ」は「犬が猫を産んだ」「犬がね子を産んだ」の2通りの解釈ができる。そこで、適当な場所でマスを空ける。点字の実際の配列を図1に示す。以下、点字のマス空けは``/"で表し、マスを空けることを分かち書き行う、と表現する。「いぬが / ねこを / うんだ」と分かち書きされば、読みやすく、誤解される可能性も減少する。ある文字列の内部で分かち書きを行わないことを「続ける」と表現し、分かち書きを行うことを「切る」と表現する。また、一般的な点字表記法にならい、点字はひらがなで表記するが、ウ列長音は``ー"(長音符)で表記し、格助詞の``は"は発音の通りに``わ"と表記する。

図 1　点字の分かち書きの例



1.2 点訳作業の流れ

図 2　本研究開発システムを使用した場合の自動点訳の流れ図



従来の自動点訳を使った点訳作業の流れ例を図 2 左側に示す。
一般的には以下のような手順で行われている。

- 前処理　紙の原稿の場合は、スキャン 、OCR による認識。ワードプロセッサ形式等の電子的な原稿の場合は、ファイルの解析、テキスト情報、レイアウト情報の抽出。
- 変換　　自動点訳ソフトによる点訳。
- 後処理　ユーザによる読み、分かち書きの修正。
- 出力　点字プリンタによる紙への出力、または点字を機械的に表示するピンディスプレーへの出力。

1.3 点訳の分かち書きの困難さについて

点字の「分かち書き」の意味とその困難さを説明するため、非課税、非常識などの「非」について考える。これは後に来る要素に依存して、「切る」場合と、「続ける」場合の両方が存在する。点訳のてびき[1]には、

・原則として、「接頭語・接尾語・造語要素と自立語との間は続けて書く。」

とある。例として、「非合法」、「非人情」があげられている。しかし、備考1として、

・「接頭語・接尾語・造語要素であっても、意味の理解を助ける場合には、発音上の切れ目を考慮して区切って書いてよい」

とある。例として、「非人道的」があげられている。点字は表音文字であるから、話し言葉の切れ目に準じるのだ、とは理解することはできる。では「非公式学校」や「非営利団体」はどうであろうか。「非公式」というひとつの単語としてとらえるか、「非」「公式」というふたつの単語としてとらえるかというのは、解釈する人によるといわざるをえない。それぞれの熟語の例を図3に示す。ここでは「非」を例にとって紹介したが、「純」「新」など、ほかにも接頭語的要素は、多数存在している。

図 3　接頭語「非」の後に続く熟語の例

「非」のすぐ後ろを続ける熟語
非常口　非課税　非通知

「非」のすぐ後ろを切る熟語
非人道的　非効率的　非上場企業

「非」のすぐ後ろを切るか続けるかの判断が非常に困難な熟語
非公式学校　非営利団体

また、接頭語的要素ではないが、「みぎはんしん」は途中で分かち書きをしないが、「みぎ / はんぶん」は途中で分かち書きをする、など点字初心者にとって、規則の不可解さについて考えさせられるような例は多数存在する。

1.4 問題点

点訳の知識がなくても、通常の日本語の知識があれば、読み下しの修正は行うことができる。しかし、分かち書きについては、点訳の知識がない人には、正確に行うことは難しい。このような、読み下しは正しく行えるが、分かち書きは完全には正しく行うことができないユーザーが、自動点訳の上記点訳作業の流れの後段階の修正をおこなう場合の問題点について考える。

ここで特に問題となるのは、自動点訳の段階での読み下しが間違っていた場合である。この部分は、分かち書きも間違っている可能性が高い。

従来の点訳システムでは、読み下しを修正した部分については、再度分かち書きをおこなうことができず、この部分については全てユーザーの点訳の知識に頼ってしまっている。

例として、「深山烏揚羽」(ミヤマカラスアゲハ)を点訳する場合について考える。正しくは、「みやま / からす / あげは」と分かち書きを行う。

ところが、自動点訳ソフトのエクストラで点訳すると、間違って「しんざんう / あげは」と点訳してしまう。ユーザーが、もし正しい読み下しを知っていた場合は、「しんざんう」を「みやまからす」と修正することができる。しかし、「みやまからす」がひと続きで「みやまからす」であるのか、途中で分かち書きを行って「みやま / からす」となるのかは修正を行うユーザーに点字の分かち書きの専門知識を必要とする。

2. 本研究開発システム

本研究開発システムを使った自動点訳の流れを図2右側に示す。

本研究開発システムは、点訳された文から、その分かち書きの誤りを判別し、修正するものである。自動点訳システムは全体としてみると、かな漢字交じり文を点訳ファイルに変換するものであるが、本研究開発システムが直接関わる部分については、入力は点訳であり、出力も点訳である。

このシステムは自動点訳作業の流れのうち、自動点訳後、ユーザーが読みを修正した後に、さらに分かち書きの誤りを判別し、修正する場面で使用することを想定している。分かち書きを完全には正しく行えないユーザーが読み下しの修正を行った場合、特に有効である。さきほどの問題点で挙げた「みやまからす」については、もしユーザーが間違えて、分かち書きをせずに、ひと続きで「みやまからす」としていても、正しく途中で分かち書きを行って「みやま / からす」と修正することができる。

具体的には、修正対象とする点訳ファイルのある一部分を取り出し、その一部分が大量の点訳ファイル(点訳コーパスと呼ぶ)の中に何件含まれているかを調べる。この含まれた件数をヒット件数とする。そして、そのヒット件数から、分かち書きの誤りを判別する。

2.1 点字専用全文検索エンジンについて

2.1.1 索引ファイルの作成について

大量の点訳ファイルから全文検索を高速に行うため、あらかじめ対象点訳ファイルから点字文字列を抽出して、索引ファイルにまとめている。索引のキーには、「点字の分かち書き」を単位として、登録した。これをキー点字文字列と呼ぶ。検索を高速にするため、キー文字列の後続の文字列も索引ファイルに含めた。キー文字列に関連づけられたレコードに、キー点字文字列に後続する14文字の点字文字列を記録した。これを後続点字文字列と呼ぶ。この後続点字文字列には、空白文字(スペース)は含まれていないが、1文字につき1ビットからなる、空白文字情報が付加されている。また、キー点字文字列に、先行する文字列も8文字が記録されている。これを先行点字文字列と呼ぶ。先行点字文字列は点訳文字列の修正には直接機能しないが、人間が用例を判断する際の助けとなる。例えば、「北アメリカ」は点字表記辞典によると「きたあめりか」と続けるが、コーパスで「きたあめりか」を検索すると、「きた　あめりか」と切られている用例もわずかながら見つかる。先行点字文字列も同時に表示すれば、これが「ひがしから　きた　あめりか　かんたいわ」と用例を表示することができ、「きた」は名詞の「北」ではなく、動詞の「来た」であることが分かる。

空白文字情報と、点字文字列情報を分けて格納しているのは、以下の理由からである。これは、検索にヒットするかどうかを考慮せずに一致判別するためである。例えば、「生老病死」は点字表記辞典によると、「しょー　ろー　びょー　し」とすべてマス空けする。もし「しょーろーびょーし」とすべて続けた、正しくない点字文字列が入力された場合でも、「しょー　ろー　びょー　し」にヒットさせたいので、このようになっている。しかし、マス空けの情報も格納していて、修正時にヒットしたかどうかの判定には、入力語に対応する本文語の末尾が、分かち書きで終わっているもののみ、ヒットしたものとしている。

読点、句読点があれば、そこで完全に文章が区切れているものと判断し、それ以降は後続文字列には含めない。先行点字文字列に含まれている場合は、読点、句読点より前は削除する。また、分かち書きは、後続する読点、句読点には影響されないので、キー点字文字列、後続点字文字列には、語の末尾に付属する読点、句読点などの情報は含めないものとする。

例として、「きょーわ　よい　てんきです。」という点字文章に、索引を登録することを考える。まず、「きょーわ」をキー点字文字列として、レコードを追加する。レコードには、先行点字文字列はなし、後続点字文字列は「よいてんきです」が格納され、分かち書き情報には、「よい」の後と、「てんきです」の後が、空いているという情報が格納される。次に、「よい」をキー点字文字列として、レコードを追加する。このレコードには、先行点字文字列は「きょーわ」、後続点字文字列は「てんきです」が格納され、分かち書き情報には、「てんきです」の後が、空いているという情報が格納される。

コーパス中に、ある単語は複数存在する場合が多いので、それぞれのキー文字列には、複数のレコードが対応している。最後に、それぞれのレコードを後続点字文字列の内容によってソートしておく。このことにより、検索時に二分木探索が可能になり、時間の短縮に役立っている。

2.1.2 検索について

N文字からなる点字文字列について検索する場合は、その点字文字列の1文字目から、j文字目($1 \leqq j \leqq N$)までの部分点字文字列を取得する。その部分文字列をキー点字文字列とし、キー点字文字列に関連したレコードを検索する。j=Nのとき、レコードはすべてヒットしたものとする。$j \neq N$のとき、j+1文字目からN

文字目までの部分点字文字列と、レコードに記録されている後続点字文字列の1文字目から(N－j＋1)文字目までが等しいかどうか判定する。このとき、後続点字文字列に含まれるの分かち書きは無視して、比較する。

たとえ等しい場合でも、後続点字文字列の分ち位置情報を確認し、比較した点字文字列の末尾が分ち書きされている場合に、ヒットしたものとする。

これは例えば、「映画館」は分かち書きをせずに、「えいがかん」と表記する。しかし、「映画鑑賞」であれば、「えいが　かんしょー」と分かち書きを行う。つまり、「えいがかん」を「えいが　かんしょー」にヒットさせないように、このような処理を行っている。

2.2 提案手法の欠点について

形態素解析や、自立語、付属語の判別をせずに点訳ファイルの内容をそのままコーパスに登録し、対象となる文書についてもそのままコーパスを参照している。

2.3 コーパスに対する検索

本研究開発システムでは、大量の点訳ファイル(点訳コーパス)から検索を高速に行うため、あらかじめ対象点訳ファイルから点字文字列を抽出して、索引ファイルにまとめている。[6]

2.4 分かち書きの修正アルゴリズム

分かち書きの間違いは2種類ある。

1.本来であればマス空けをせずに続けるべき場所をマス空けしてしまった間違い。

2.本来であればマス空けをするべき場所でマス空けをしなかった間違い。

まず、1.について考える。修正前の点字文字列の、すべてのマス空けについて、マスを空ける場合と、空けない場合の両方についてヒット件数を求める。そのヒット件数の比が1:10以上であれば、修正すべきものと判断し、マス空けを削除する。

次に、2.について考える。修正前の点字文字列の、あるマス空けによって囲まれた部分点字文字列について、マス空けをしなかった場合と、部分点字文字列中のある位置でマス空けをした場合とのヒット件数の比を求める。その比が1:10以上になるようなマス空けが存在すれば、修正すべきものと判断し、マス空けを追加する。

例として、点字文字列の「とってかえす」、「らんど / まーく」について、分かち書き修正を試みた場合の途中経過を図4に示す。

図4　分かち書きの修正アルゴリズム

例1.
対象点字文字列：**「とってかえす」**

コーパスに対する検索結果:

「とってかえす」:　37件 －(1)
「とって / かえす」: 672件 －(2)

→ (1) : (2) = 1 : 18.16 ＞ 10
「とって / かえす」と修正

例2.
対象点字文字列：**「らんど / まーく」**

コーパスに対する検索結果

「らんど / まーく」:　53件 －(1)
「らんどまーく」:　909件 －(2)

→ (1) : (2) = 1 : 17.15 ＞ 10
「らんどまーく」と修正

2.5　読みからの自動点訳

しかし、ある点字文字列 B がまったくコーパスにヒットしない場合は、どのように分かち書きを判別したらよいであろうか。これに対しては、かな漢字変換の文節数最少法[7]に似た手法により、分かち対応する。文節数最少法では、解析結果の文節数を最少にする解析を採用するが、本システムでは、それぞれがコーパスにヒットするように解析結果をいくつかに分割し、その分割数が最少になる分割を採用する。

この最少分割問題を定義すると以下のようになる。

点字 L 文字から構成される入力点字文字列をB_Lとする。

入力点字文字列のうち、i番目の点字からj番目の点字までの部分点字文字列をB_j^iとする。以下、部分点字文字列を単に単語と表現し、複数の単語の集合を単語列とする。

単語をwとするとき、w をコーパスから検索したときのヒット件数を hit(w)とする。

単語列を W=w_1, w_2.....,とする。

単語列 W に含まれる単語の数を|W|とする。

単語列 W の各単語w_kを接続した文字列を str(W)とする。

入力文字列B_Lの単語分割は、

$$\widehat{W} = \underset{W:str(W)=B_L}{\operatorname{argmin}} |W|$$

となるような$\widehat{W}$を求める問題と定義される。

これは Viterbi アルゴリズムによって計算することができる。

(1)初期化

N(0)=0

for i =1 to L do, N(i)=∞

(2)繰り返し:x =1～L
N(x)=min $f(x,i)$
I(x)= $\mathrm{argmin}_i f(x,i)$
W(x)= $B_j^{I(x)}$

$f(x,i)$ = N(i -1)+1 if hit(B_x^i) > 0
= ∞ if hit(B_x^i) = 0

(3)終了
M=|$\widehat{W}$|=N(L)
$\widehat{w_M}$=W(L)
i (M)=I(L)

(4)バックトラック
$\begin{pmatrix} \widehat{w_k} = \mathrm{W}(i(k+1)-1) \\ i(k) = \mathrm{I}(i(k+1)-1) \end{pmatrix}$ for k =M-1,M-2,...1
となる。

単語列 それぞれのコーパスに対するヒット件数 hit(w_k)は 1 以上で、なおかつ分割数 M が最少になるように分割するが、そのような分割方法が複数ある場合、ヒット件数の総積

$$\prod_{k=1}^{M} \mathrm{hit}(w_k)$$

が最大となるような分割を採用する。
分割されたそれぞれの単語は途中に分かち書きを含む可能性があるため、最後に検索にヒットした単語の中から、語中の分かち書きについて多数決を行い、最も件数の多かった分かち書きを採用する。
例えば、「常識に欠けます」という文を点字で読み下すと「じょーしきにかけます」となる。この点字文字列の分かち書きを判定する過程を図 5 に示す。
この点字文字列はそのままではコーパスにはヒットしない。
それぞれがコーパスにヒットするように最少分割すると、「じょーし / きにかけます」と、「じょーしきに / かけます」の二通りに解釈できる。
前者は「じょーし」のヒット件数が 8372、「きにかけます」のヒット件数が 9、前者のヒット件数の積は 7.5×10^4、後者は「じょーしきに」のヒット件数が 4048,「かけます」のヒット件数が 3192、後者のヒット件数の積は約 1.3×10^7であり、後者のヒット件数の積のほうが大きいため、後者の「じょーしきに / かけます」を採用する。

図 5「読み」からの自動点訳のアルゴリズム

分かち書きされていない点字文字列
「じょーしきにかけます」の分かち書き

1. 部分点字文字列を点訳コーパスより検索

じょーしきにかけます　(0件)
じょー　(25614件)
じょーし　(8339件)
じょーしき　(4393件)
じょーしきに　(4045件)
うし　(5311件)
しき　(34279件)
しきに　(5429件)
きに　(440182件)
きにかけます　(9件)
かけ　(68824件)
かけます　(3193件)
ます　(9494件)

2. 件数>0で、最少分割数となる解を求める

じょーし / きに / かけ / ます　(4分割)　×
じょーし / きに / かけます　(3分割)　×
じょーしきに / かけ / ます　(3分割)　×
じょーし / きにかけます　(2分割)　○　- (1)
じょーしきに / かけます　(2分割)　○　- (2)

3. 分割数が同じ場合はヒット件数の総積の最大となる解を採用する

(1) じょーし (8339件) / きにかけます (9件)
8339 × 9 = 75051
(2) じょーしきに (4045件) / かけます (3193件)
4045 × 3193 = 12915685

(1)<(2) よって
(2)の「じょーしきに / かけます」を採用

2.6　点字の仮名遣い、連濁について
例えば「腹の内が」は「はらの / うちが」だが、誤って分かち書きを行わなかった場合、点訳の仮名遣いのルールにより、ウ列長音は``ー"(長音符)を使って表記するため、「はらのーちが」となっているはずである。このような点字文字列を検索する場合、長音を「う」に直してから、コーパスを検索している。また、連濁をおこすと考えられる語は、連濁についても考慮して、辞書を参照している。「たまり醤油」は「たまり / しょーゆ」と「たまりじょーゆ」の両方について、コーパスを検索している。

2.7　“副作用"とその低減について

「もう学校には行かない」は点訳では、「もー / がっこーにわ / いかない」とマス空けされる。「もーがっこー」をコーパスから検索すると、途中で分かち書きをする「もー / がっこー」のヒット件数が 56 件で全体の 2% 、分かち書きをしないで一続きで書く「もーがっこー」のヒット件数は 2869 件で全体の 98% である。2.5 で示した通り、この件数の比は 1:10 以上であるので、本研究開発システムは間違って修正を行ってしまう。つまり、「もう学校には行かない」という文を本研究開発システムにかけると、「盲学校には行かない」という意味に変換されてしまう。
「もう学校には」など、ある程度``副作用"が発生する単語を予測し、修正から除外することも考えられるが、完全に副作用をなくすことは難しい。修正後、修正箇所を人が見てチェックする必要がある。
"副作用"を低減するためには、これを予測する必要がある。これには、正しいとわかっている点訳ファイルに対し、あえて修正をおこない、どの単語が「誤って修正されてしまった」かを記録することにより、自動的に予測することができると考えられる。
サピエ図書館に登録されている 500 タイトルの点訳ファイルについて、
"副作用"を検出するための、あえて誤り修正をおこなった結果、以下のような"副作用"が検出された。

- いた美和 [正:いた / みわ 誤:いたみわ(痛みわ)]
- いたメモの [正:いた / めもの 誤:いためもの(炒め物)]
- この蓑 [正:この / みの 誤:このみの(好みの)]
- (手紙を)したため、 [正:したため 誤:した / ため((動作を)したため、)]
- 美奈さんわ [正:みな / さんわ 誤:みなさんわ(皆さんわ)]
- 蒸すこと [正:むす / こと 誤:むすこと(息子と)]

2.8 数字の取り扱い
数字はすべて、同じものとして取り扱う。分かち書きの規則上、数の大きさが分かち書きに影響をあたえることはないからである。例えば、「第 1 次」、「第 2 次」…「第 n 次」などはすべて同じ分かち書きである。索引ファイルの作成時に、数字は便宜上、すべて 1 に置換し、登録してある。参照時にも、数字は便宜上 1 に置換し、参照する。

2.9 分かち書き辞典としての応用
「デオキシリボ核酸」など、点字の熟練者でも、分かち書きに迷う語は多い。「点字表記辞典」[8]が出版されているものの、すべての名詞や地名を網羅することはできない。その点、コーパスを参照することにより、いままでに点訳されたことがある語であれば、どのくらいの割合でどのように分かち書きされているかが判別できる。例として、図 6 に「でおきしりぼかくさん」の分かち書きを検索した場合の結果を示す。

図 6 分かち書き辞典としての応用例

●分かち書きの割合:
58%(51件) **でおきし / りぼかくさん**
35%(18件) **でおきしりぼ / かくさん**
5%(3件) **でおきしりぼかくさん**

●用例:
○でおきし / りぼかくさん:
とわ でおきし りぼかくさん ぬくれいっく あ
ある でおきし りぼかくさん すなわち

○でおきしりぼ / かくさん
つまり でおきしりぼ かくさん とくゆーの はいれつ
ーた でおきしりぼ かくさん ていおん さっきん しゅ

○でおきしりぼかくさん
かくさんるい、 でおきしりぼかくさん りぼかくさん 3 びたみん
ＤＮＡ でおきしりぼかくさん かんていで ひんしゅを

3.実験

3.1 コーパスに使用した点訳ファイル

点字図書館などの団体である、全国視覚障害者情報提供施設協会が運営している「ないーぶネット」(現在は「サピエ図書館」に移行)に登録されていた点訳ファイルのうち、一部の 34217 タイトルの点訳ファイルをコーパスの作成に使用した。2001 年に新たな点字表記法[9]が制定されているが、予備実験では、2001 年版より前の点字表記法に基づく点訳ファイルも混在しており、これが悪影響を与えることが分かった。そのため、あらかじめ 2001 年版点字表記法に基づく点訳ファイルのみを選んで使用した。
点字表記法が 2001 年版に基づくかどうかの判断は、以下のように行った。
2001 年版では、サ変動詞「する」の扱いが変わり、それ以前の表記法では原則として前の語に続け、2001 年版では前の語との間はマスを空ける。これを判別に利用した。具体的には、点訳ファイルから、全ての「する」を数え、「する」の前が空いていない場合の割合が 3% 以下であれば、2001 年版に基づいているとみなす。ただし、「対する」「制する」「属する」などは、2001 年版に基づいていても、前の語と続けるので、これらは予め除外した。

3.2 誤りの定義

分かち書きの誤りについては、正解は続けているにもかかわらず、対象が切れている「切りすぎ」と、正解点訳では切れているにもかかわらず、対象点訳では切れていない「切り忘れ」の両方がある。「切りすぎ」と「切り忘れ」が隣接している場合、例えば「回転待ち行列」が正解点訳は「かいてんまち / ぎょーれつ」であるが、対象点訳では「かいてん / まちぎょーれつ」となっているような場合、単純に「切りすぎ」1 個と「切り忘れ」1 個とはせず、「切り位置間違い」1 個とカウントした。「切りすぎ」「切り忘れ」「切り位置間違い」は合計して「分かち書きの誤り」とした。
また、読みの誤りについては、正解点訳の単語を単位とし、それに対応する対象点訳が 1 文字でも間違っている場合、1 個の間違いとした。読みが誤っている部分については、分かち書きも誤っている可能性は高い。読みと分かち書きが両方間違っている場合、「読みの誤り」に分類した。

3.4 実験 1. 自動点訳における先行研究の後処理としての提案

[3]のまえがきにおいて、兵頭らは、「特に自動点訳後の後編集を支援する機能の充実が望まれている」と指摘した。本研究開発システムは、この後編集に関するシステムについての提案である。本研究開発システムは、自動点訳特有の分かち書き誤りのうち、かなりの部分を修正することが可能である。例として、[3]において、考察された、分かち書き誤りの分類を本研究開発システムにより修正することを試みる。

- 「点字規則の不備による誤り(複合語以外)」は、「山田さん」などの人名の接尾語「さん」の前は区切るが、「歯医者さん」や「炭鉱夫さん」などの普通名詞についた「さん」の場合は、続けるという規則による誤りである。本研究開発システムにより、「歯医者さん」は修正できたが、「炭鉱夫さん」は修正できなかった。これは「炭鉱夫さん」という単語がコーパスに全く含まれていなかったためである。
- 「文節解析の誤り」は、辞書情報の不足などにより、正しく文節解析が行われず誤った例である。「注意して歩かんからやろ」が(注意して / 歩 / かんからや / ろ)と間違って解析されていたものは修正できなかったが、「21 世紀に向けてこの素晴らしい」が(21 世紀に / 向け / てこの / 素晴らしい)と間違って解析されていたものについては、本研究開発システムにより、(向け / てこの)という部分を(向けて / この)と正しく修正することができた。
- 「点字規則の不備による誤り(複合語以外)」についても、本研究開発システムにより、「どーし」は「どー / し」、「どー / こー」は「どーこー」と正しく修正できた。
- 「固有名詞等の未登録語による誤り」はコーパスに出現した固有名詞(歴史上の人物や有名な人物など)については、修正することが可能であるが、ニュースに出てくる人名などで、まったく未知の人名については、修正することができない。
- 「カタカナ表記語の切れ続きに関する誤り」については、コーパスにまったく含まれていないほど特殊な用語はともかく、一般的な語では修正できる可能性は高い。
- また、[5]において橘らが指摘している「する / とつぎのような」も正しく「すると / つぎのよーな」と修正できた。

3.5 実験 2. 熟練していない点訳者による点訳の修正

本研究開発システムは点訳の初心者に対して特に有効であると考えられる。しかし点訳初心者が作った点訳ファイルを、まとまった数で入手することは難しい。それは、点訳初心者は点訳グループに所属する点訳者のうちのごく一部分であり、また、点訳初心者が作成した点訳ファイルは、多くの場合、熟練点訳者によって修正された後に公開されるからである。そこで、ある刑務所において、訓練の一環として訓練生(受刑者)が作成した点訳ファイルに本研究開発システムを適用する実験を行った。この訓練生に点訳経験者はおらず、いずれも刑務所での点訳経験は 1 年未満である。

3.5.1 手順

ある刑務所において、訓練生が点訳し、その後訓練生同士でチェックした段階の点訳ファイルに対し、本研究開発システムを用いて誤り修正を行った。熟練点訳者が修正したものを正解とし、本研究開発システムによる修正前と修正後で誤りの数を比較した。

3.5.2 結果

本研究開発システムによる修正の結果を表 1 に示す。

表 1　熟練していない点訳者による点訳に対する
本研究開発システムによる修正

修正前誤り数(個)	修正前誤り数(個)	減少率(%)
16	8	50

点訳コーパスによる自動修正前は 16 個の誤りが存在したが、修正後は、8 個に減少した。すべての分かち書き個所の数は 8748 であり、点訳コーパス修正後の誤りの個数の、すべての分かち書き個所に対する割合は約 0.1 パーセントである。

自動修正により減少した 8 個の誤り数の内訳を、表 2 に示す。正しい修正が 10 個で、誤った修正(副作用)が 2 個である。差し引きで 8 個減少した。