構文構造と意味構造を統合した
単語依存構造による
多言語対応日本語解析に関する研究

提出先　　大阪大学大学院情報科学研究科

提出年月　2021 年 1 月

田 中　貴 秋

本研究に関連する研究業績

学術論文

(1) 田中貴秋, 永田昌明. 日本語の文法機能タイプ付き依存構造解析. 自然言語処理. Vol. 26, No. 2, pp. 441–481, 2019.

(2) 浅原 正幸，金山 博，宮尾 祐介，田中 貴秋，大村 舞，村脇 有吾，松本 裕治. Universal Dependencies 日本語コーパス. 自然言語処理. Vol. 26, No. 1, pp. 3–36, 2019.

国際会議

(3) Takaaki Tanaka, Katsuhiko Hayashi and Masaaki Nagata. Hierarchical word structure-based parsing: A feasibility study on UD-style dependency parsing in Japanese, In *Proceedings of the 15th International Conference on Parsing Technologies (IWPT 2017)*, pp. 56–60, 2017.

(4) Takaaki Tanaka, Yusuke Miyao, Masayuki Asahara, Sumire Uematsu, Hiroshi Kanayama, Shinsuke Mori and Yuji Matsumoto. Universal Dependencies for Japanese, In *Proceedings of the Tenth International Conference on the Language Resources and Evaluation Conference (LREC 2016)*, pp. 1651–1658, 2016.

(5) Takaaki Tanaka and Masaaki Nagata. Word-based Japanese typed dependency parsing with grammatical function analysis. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP 2015)*, Vol.2, pp. 237–242, 2015.

(6) Takaaki Tanaka and Masaaki Nagata. Constructing a practical constituent parser from a Japanese treebank with function labels. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2013)*, pp. 108–118, 2013.

研究会・全国大会

(7) 田中貴秋, 荒瀬由紀, 永田昌明 鬼塚真. Universal Dependencies に基づく多言語間テキスト意味類似性測定. 第 34 回人工知能学会全国大会論文集, 3Q5GS902, 2020.

(8) 田中貴秋, 永田昌明, 荒瀬由紀, 鬼塚真. 意味グラフに基づくテキスト類似性尺度の提案. 第 32 回人工知能学会全国大会論文集, 3G1-02, 2018.

(9) 田中貴秋, 林克彦, 永田昌明. 日本語の単語依存構造のための長単位解析. 第 30 回人工知能学会全国大会論文集, 1K3-2, 2016.

(10) 田中貴秋, 永田昌明. 日本語のラベル付き依存構造解析の検討. 言語処理学会第 21 回年次大会発表論文集, pp. 1044–1047, 2015.

内容梗概

本論文では， 従来日本語の自然言語処理で扱われてきた独自体系の構文構造に基づく言語解析の課題を解消し，複数の異なる言語を横断して扱う多言語自然言語処理（多言語処理）への対応を容易にする新たな日本語解析の枠組みを提案する．本論文の貢献は，(1) 多言語処理に対応した新たな日本語構文解析の設計と大規模データ構築による有効性の検証，(2) 新たな日本語構文解析に適合した解析手法の考案，(3) 新たな構文解析と拡張した意味解析との組み合わせによる多言語対応のテキスト意味類似度測定方法の考案の 3 点である．ある言語で記述されたテキストは，構文的な構造や意味的な構造を内包しており，伝達する意味内容を理解して対話や翻訳等を行おうとすれば，言語解析を行ってこれらの構造を捉えることが必要になる．構文解析等の言語解析は対象とする言語に依存する特徴があることが多く，この言語ごとの違いが言語を横断した処理を行う場合の障壁となる．日本語の構文解析に関しては，文節という日本語特有の単位に基づく構造で表すのが通例であり，このことが日本語を含めた多言語処理を行う際に処理やデータ構造の共通化を図るのを難しくしていた．

本論文の第一の貢献は，日本語の構文解析と他の言語の構文解析との共通化を図るため，日本語の構文情報を適切に保持可能かつ多言語処理対応が柔軟に行える単語依存構造に基づく構文解析の枠組みを考案し，大規模なデータにより有効性を検証した点である．単語依存構造の設計として 6 タイプの依存構造スキーマを考案し，構文解析の全体精度に有利なスキーマ，述語項構造取得に有利なスキーマ等のスキーマの特性を確かめた．単語依存構造による構文解析の枠組みの有効性は，従来の文節依存構造解析と同等の精度を実現可能であることにより確認した．また，新たな構文解析は文節依存構造にはない文法機能の情報を高い精度で得ることが可能であり，従来独立した処理で行っていた述語項構造情報を高精度で獲得できる．

本論文の第二の貢献は，新しい枠組みである日本語の単語依存構造解析を行うにあたり，形態素情報を保持する単語単位（短単位）と構文情報を保持する単語単位（長単位）からなる階層的単語構造に基づく方法を考案した点である．長単位の同定と依存構造の同定は独立の課題ではないため，それらを逐次的に行うと前段の処理の誤りが後段の処理に伝播する．本論文では，長単位の同定と依存構造の同定を同時に行うことにより，誤りの伝播を防止しつつ両者の整合性を高めることを可能にする構文解析の方法について考案した．長単位と短単位の階層的単語構造に基づく構文解析について逐次解析手法と提案手法である同時解析手法による検証を行い，同時解析手法は全体的な解析精度では逐次解析手法を僅かに上回り，意味解析に重要である述語項構造等の文法的関係の同定について有意な精度向上が確かめられた．階層的単語構造に基づく日本語の単語依存構造解析は，構文解析の多言語対応のみならず意味解析との対応にも寄与する．

本論文の第三の貢献は，新たな日本語の単語依存構造解析を含む構文解析と意味解析とを組み合わせることによりテキスト意味類似度を測定する方法を考案した点である．従来の日本語の文節依存構造による構文構造は述語項構造等の意味構造や他の言語との対応付けが困難なことが問題となっているが，本論文で提案する単語依存構造による構文構造は意味構造との親和性が高く構文構造を意味構造に変換することが可能である．機械翻訳の評価や剽窃検出などでは，複数のテキストの意味内容を比較し，同一の内容を表しているのか，異なっているならば相違点は何かを判別することが重要である．同一の意味内容を表現する自然言語の表現は無数に存在するため，テキストを比較する際には表層的な情報を直接比較するのではなく，一旦抽象的な表現に変換してから行うことが適切であると考えられる．本論文の手法は，単語依存構造による構文構造と述語項構造を中心とした構造に変換した意味構造の対応付けを行うため意味内容の類似点，相違点を提示することができる．提案手法は，ベンチマークデータでの評価結果では表層的な情報のみを使う手法に全体的な精度では及ばないが，同様に意味表現の対応付けを行うベースライン手法より高い精度が得られることが確かめられた．また，提案手法はテキストを抽象度の高い意味表現に変換してから比較を行うため，異なる言語間の類似度測定を同一言語同士の比較の場合とほぼ同様の方法で行うことが可能である．このように単一手法で日本語を含めた多言語間の類似度測定が実現できたことには，本論文の第一の貢献である日本語の単語依存構造の枠組みの確立が大きく寄与している．

以上のように本論文では，従来の文節依存構造による日本語の構文解析に代わって，単語依存構造による構文解析を導入し多言語処理への対応を容易にする新たな日本語解析の枠組みを提案した．第 1 章で日本語解析の多言語化に関する研究の背景と目的について述べ，第 2 章で多言語に対応する新たな日本語の単語依存構造解析の枠組みの提案と大規模データ構築，解析器への適用による有効性の検証を行い，第 3 章で階層的単語構造に基づく解析手法の提案とその有効性の検証を行った．第 4 章では日本語の単語依存構造解析を多言語間を含むテキスト意味類似度評価に適用することにより，新たな単語依存構造解析は従来の文節依存構造解析と同等以上の精度を保持した上で，多言語処理への適用性の向上および意味解析と組み合わせた処理の有効性を示した．第 5 章で本研究の成果についてまとめる．

目 次

第1章　序論

1.1　研究の背景と目的

昨今インターネットを中心として，様々な言語で記述された膨大な量の電子的なテキストデータが日々生み出されるようになり，単一の言語だけを対象とするのではなく複数の異なる言語を横断して扱う多言語自然言語処理 (多言語処理) の重要度が高まっている．あらゆる言語を対象にした多言語処理を実現する観点から言えば，個別の言語に依存した特有の方式で処理するのではなく共通の枠組みに基づいて処理を行うのが理想である．しかし，人類がコミュニケーションに使用している言語は成り立ちや文化的背景などの違いから多様性が高く，文を構成する文法的な構造である構文構造だけでも単一の枠組みで表現することは簡単ではない．ウェブサイト上のデータ量が上位の言語に着目したとき，日本語は他の上位の言語である英語やロシア語等[1]と言語族が異なりかつ解析の体系が独自に発展しているため，他の言語と連携した処理を行うことは容易ではない．この独自体系に基づく解析処理により生じている多言語処理の障壁を解消することは，日本語話者にとって有用なだけでなく膨大な日本語の資源を多言語処理の中で有効に活用するための大きな貢献になると考えられる．本論文では，日本語の解析の独自性により生じている多言語処理対応の障壁を解消することにより，言語依存性が低く多言語処理へ対応可能な日本語解析の枠組みを提案する．

テキストデータは表層的には一次元の文字列の羅列であるが，文字列間の依存関係から構成される文法的な構造（構文構造）と，構文構造により表現される意味的な構造（意味構造）を包含している．一方，言語に依存しない多言語処理への主要なアプローチとして，テキストが本来持っている構文構造や意味構造等の構造情報を無視してテキストデータに存在する表層的情報の統計的な類似性を利用する方法がある．この方法は，大量のテキストデータが利用可能になったことと計算機の処理能力が大幅に向上したことを背景に，情報検索や機械翻訳等の様々なタスクで高い性能を実現している．ただし表層的情報を用いる方法は，単語の出現頻度の傾向や共起する単語の傾向が類似している等，大量のデータから得られる表層的情報に関する統計的な傾向から類似する現象が推定可能な場合には有効に働くが，テキストが表現している構造情報を捉える必要がある問題には限界

[1]いずれも 2019 年の統計において上位 10 位以内に位置する (https://w3techs.com/technologies/history_overview/content_language/ms/y).

(E1)	He saw a sailor with a telecsope.
(J1)	彼は望遠鏡で船員を見た.
(J2)	彼は望遠鏡を持った船員を見た.
(J3)	彼は望遠鏡によって船員を見た.

図 1.1: 類似した単語が含まれる文の例.

がある．例えば，自然言語処理の重要なタスクの一つとして 2 つのテキストの表現する内容を比較することを考える．図 1.1 において，英語の文 (E1) に意味的に対応する日本語の文を (J1)-(J3) の中から選択しようとするとき，いずれの文も “he”, “saw”, “sailor”, “telescope” に対応する語「彼」,「見た」,「船員」,「望遠鏡」が全て含まれており，出現単語の類似性のような表層的情報のみでは差が出ない．これらの違いを区別するためには構造的な情報が不可欠となる．また，英語の文 (E1) は構造的な曖昧性を含む．すなわち文中の “with a telescope” という前置詞句の修飾している対象が動詞 “saw” であるか，名詞 “sailor” であるかという曖昧性が存在する．この曖昧性を解消しなければ，対応する構文構造および意味構造を決定してテキストが表現する内容を正しく解釈することができない．英語の文の場合と同様に日本語の文においても,「望遠鏡」という語が動詞「見た」や名詞「船員」と関係があるのか，関係があるならばどのような関係なのかという構造的情報を捉えて比較することが必要になる．このように，テキストが本来伝達しようとしている意味内容を捉えて高度な処理を行うためには，テキストを構成する単語，各単語が文法的制約下で構成される構文構造，構文構造により表現されている意味構造を解析する必要がある．

日本語と英語のように異なる言語で記述されたテキストを比較する場合には構文構造や意味構造が言語に依存しない共通の枠組みに則っていることが望ましいが，実際には言語それぞれの特性が異なるため独自の枠組みが用いられることが多い．本論文の対象である日本語の場合，構文の基本要素を「文節」という独自の単位として定義し文節に基づく依存構造を構文構造情報として扱うことが主流である．文節は韓国語等の少数の例外[2]を除いて英語や中国語等の他の多くの言語では対応する単位がなく，文節を構造情報の基本単位とすると多言語間で構造情報の比較を行う場合に障壁となる．

本研究では，テキストの表現する意味内容を文法的な構造，意味的な構造を解析した上で比較する．特に機械翻訳等の翻訳の評価，多言語情報検索や剽窃検出を念頭において，異なる言語で記述されたテキストの間であっても同一の言語で記述されたテキスト間と共通の方法で構造の解析および構造の比較を行えることを目指す．前述のように言語は成り立ちや文化的背景などから多様性が高く，文法的な構造だけでも単一の枠組みで表現する

[2]文節と類似する単位「語節」が存在する.

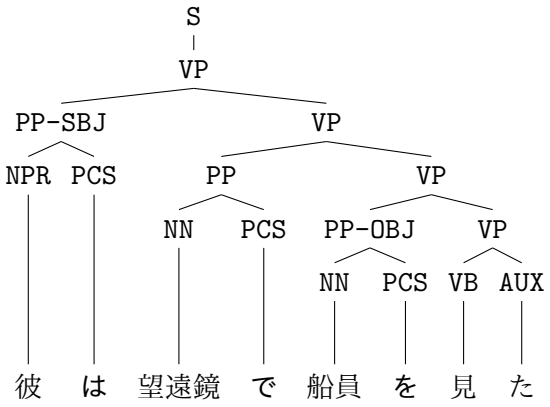


図 1.2: 句構造による構文構造表示の例.

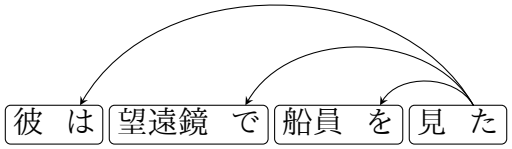


図 1.3: 文節依存構造による構文構造表示の例

ことは簡単ではないが，ほとんどの言語が共通に有している単語という単位を基礎として構文情報を表現しようとする枠組みとして，Universal Dependencies (UD) [1] がある．UD は単語単位という比較的小さい単位を基礎とすることで，多くの言語に適用可能な枠組みを実現している．本論文では，UD を含む単語単位の構文構造を基礎として日本語の解析を多言語に適応する枠組みを検討し，実際に解析器上でモデルを構築して解析を行い解析精度面など実行可能性の検証を行う．構造情報を用いた多言語処理として，テキスト間の意味的な類似性を測定するタスクに適用しその有用性について検証する．

また，言語による表現は多様性が高く同一の言語の同様の意味内容であっても用いられる文法的な構造が同一であるとは限らない．さらに多言語間ではそれぞれの言語的特徴が異なるため，文法的な情報のみで対応付けを行うことは十分ではない．図 1.1 の例で，文 (E1) の “with a telescope” が文 (J2) の「望遠鏡を持った」と意味的に対応付けられた場合についても，それぞれの言語での文法的役割は英語の文では「前置詞句による名詞の修飾」，日本語の文では「連体修飾節による名詞の修飾」であり異なる．これらが同様の意味内容を表現していると解釈するためには，文法的な構造からさらに抽象的な構造に変換して比較する必要がある．本論文では言語独自の自然言語処理に必要な情報を保持しつつ意味構造との統合がしやすく，かつ他の言語との対応付けが行いやすい単語依存構造による構文構造を，これまで単語依存構造があまり使用されてこなかった日本語に適用する試みと検証を行う．

表 1.1: 本論文の句構造木で使用する主な記号（終端記号 / 非終端記号）.

	句構造木上の記号	品詞 / 句	語句の例
終端記号	NN	名詞	犬，車
	NPR	代名詞	彼女，そこ
	VB	動詞	見る，ある
	ADJ	形容詞	赤い，早い
	ADV	副詞	すっかり，ちょうど
	PCS	格助詞	が，を，に
	AUX	助動詞	た，だ, たい
非終端記号	NP	名詞句	昨日聞いた話
	VP	動詞句	駅まで歩いた
	PP	後置詞句	鳥が, 徒歩で
	ADJP	形容詞句	とても美しい
	ADVP	副詞句	少し早く

1.2 構文解析と意味解析

本節では，本論文の中心的な課題である構文解析および意味解析について概観し，第 2 章以降の前提となる事項を説明する．これらの解析は，テキストが表現している構造を把握するために必要となる自然言語処理における基礎的な処理である．

構文解析は，各言語の制約である文法に従って文を組み立てる際の構造（構文構造）を解析する．構文構造を表現する代表的な方法には句構造と依存構造がある．句構造は，文法的な機能を担う単位である句（名詞的な機能を持つ名詞句など）を単位とした木構造である．句の単位は各言語により異なる．依存構造は，ある単位間の依存関係を集積することにより構成される構造であり，単語や句（文節など）が単位として用いられる．

図 1.2 は，日本語の文 (J1) の構文構造を句構造（木）で表現した例である．中間ノードに付与されたラベルは非終端記号と呼ばれ，配下の部分木の句の文法機能を表している．例えば非終端記号 VP は動詞句，PP は後置詞句を表しており，それぞれ配下の句が動詞を中心（主辞）にした句，後置詞（助詞）を中心にした句であることを示している．表 1.1 に本論文の句構造木で用いられる主な記号を示す．図 1.2 の句構造は,「望遠鏡で」という句が「船員を見た」という句と直接文法的関係があるという依存構造と同様の情報を持っているだけでなく，それぞれの句が後置詞句 (PP)，動詞句 (VP) という役割を担っているという情報を持っている．このように一般に句構造は詳細な文法的な情報を保持することができる．

一方，依存構造は各部分構造にあたる情報を持たないため句構造に比べてシンプルな表

現となる．図 1.3 は，同じく日本語の文 (J1) について文節を単位とした依存構造による構文構造の例である．句構造とは違い文節自体や文節間の依存構造に文法機能のラベルは付加されていない．この例のように文節間の依存構造には詳細なラベルが付加されないことが多いが，英語等で使用されている単語を単位とした依存構造では単語間の依存関係のラベルとして文法機能が付加されることが一般的である．図 1.4 は，英語と日本語の単語を単位とした依存構造による構文構造の例である．依存関係をもつ単語間を矢印で結ぶことで構造を表す．この矢印の元になっている語を矢印の先になっている語の「主辞」(head) と呼び，矢印の弧の上の文字列は 2 語間の文法的関係（文法機能）を示している[3]．図 1.4 の上 2 段の英語の依存構造 (E1)a, (E1)b において，前節で述べたような “with a telescope” の関係する構文的な曖昧性は，“telescope” を指している矢印の元が “sailor” であるか，“saw” であるかに対応している．日本語の依存構造 (J1) では,「望遠鏡（で）」の主辞が「見る」になっており，英語の依存構造 (E1)b と類似した構造になっていることがわかる．

意味解析は，構文構造によって表現された意味内容を抽象化した形で表現した意味構造を解析する処理である．意味構造の表現には様々なレベルが考えられるが，本研究では構文構造との直接的な対応が考えやすい述語項構造を中心に考える．述語項構造とは，文の主要な要素である動詞や形容詞等の述語を中心として，その述語と関係する名詞で表現される概念（項）との関係を表現するものである．図 1.5 は，図 1.4 の英語の依存構造 (E1)b と日本語の依存構造 (J1) から抜き出した述語項構造の例である．PREDICATE (述語) “see” と「見る」それぞれに対して，AGENT (動作主)，THEME (対象) などの意味役割を持つ項が存在することを示している．述語項構造のレベルに抽象化すると，構文的な相違や言語的な違いをある程度吸収することができ意味内容の比較等をすることが可能になる．

1.3 日本語の多言語処理の課題と解決の方針

前節まで述べたことから，日本語の多言語処理を実現するためには (1) 多言語に適応可能な日本語の構文解析の枠組みの考案，(2) 多言語対応の構文解析を精度良く行うための解析手法の考案，(3) 多言語対応の構文解析を用いた多言語処理の有効性の検証を行う必要があると考える．以下ではそれぞれの課題と本論文における解決案について述べる．

(1) に関する課題は，多言語共通の構文解析の枠組みとして UD が提案されているが，UD は構文構造の表現方法として単語間の依存構造が使われてきた英語を中心とするインド・ヨーロッパ語族の言語圏から考案されており，日本語への適合性の検証が十分でない

[3]図では見やすさのため一部を省略して表記している．

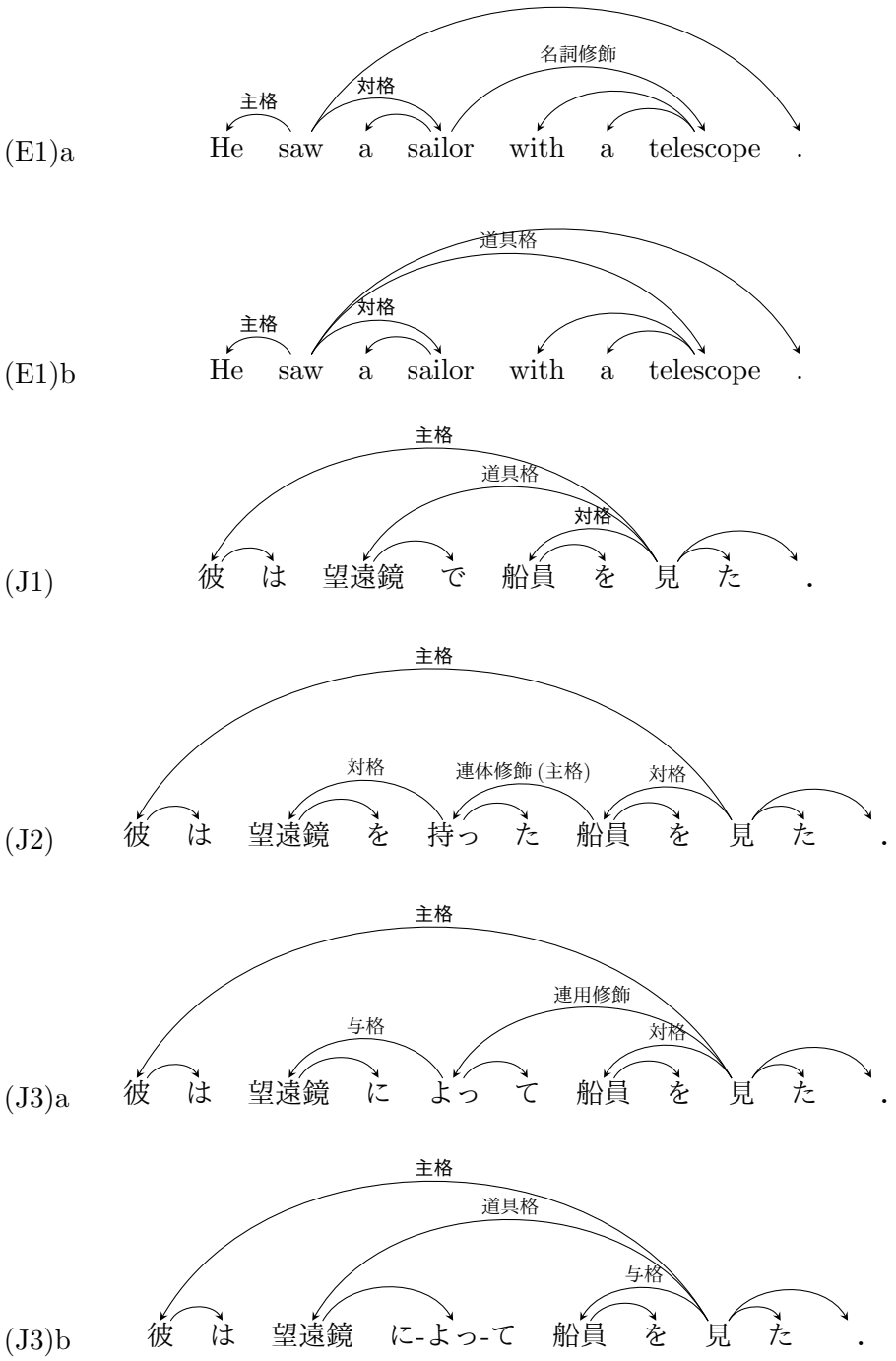


図 1.4: 単語依存構造による構文構造表示の例.

点である．前述のように日本語において構文構造の代表的な表記法は「文節」を単位とする依存構造であり，単語依存構造が用いられることは一般的ではない．そこで，日本語の単語依存構造を考えるにあたって，日本語の特徴を捉えつつ UD の枠組みに適合できる単語間の依存構造を考える必要がある．純粋な UD への日本語の適用については，UD の日本語仕様に携わる研究 [2, 3, 4] により仕様が策定されているが，日本語の特性を表現

PREDICATE:	see	PREDICATE:	見る
AGENT:	he	AGENT:	彼（は）
THEME:	sailor	THEME:	船員（を）
INSTRUMENT:	(with) a telescope	INSTRUMENT:	望遠鏡（で）

図 1.5: 単語依存構造から抽出した述語項構造.

するための単語依存構造は考えられていない．本研究では，UD と互換性を維持しつつ日本語に適合する単語依存構造を考える．これにより，日本語の自然言語処理として妥当かつ多言語処理にも対応可能な単語依存構造を実現することができると考える．

(2) に関する課題は，構文解析を行う際に依存構造の単位認定の曖昧性を考慮する必要がある点である．例えば，日本語の文 (J3) の構文構造の曖昧性として，図 1.4 の依存構造 (J3)a, (J3)b が考えられるが，その違いは依存構造の単位である単語認定を「に」「よっ」「て」の 3 語とするか,「によって」の 1 語とするかに依っている．通常の構文解析では，形態素解析により全ての単語認定を完了した後に構文構造を同定するが，この方法では単語認定の誤りが構文解析の誤りに伝播するため，単語認定の曖昧性を考慮した精度良い構文解析を行うことが困難である．本論文では，依存構造の単位である単語認定と構文構造の同定を同時に行う構文解析手法について提案する．

(3) に関する課題は，言語表現は多様性が高いため多言語のテキスト間の比較を行う際に構文解析の比較のみでは限界があるという点である．多言語テキスト間の比較に多言語共通の構文解析の枠組みを用いることは有用であるが，さらに構文構造の情報から意味構造に抽象化して比較することにより，表現に用いる構文の違いや言語による違いを吸収して比較することが可能になると考えられる．本論文では，多言語共通の構文構造から図 1.5 に示すような意味構造に変換することによって，同一言語のテキスト間または異なる言語のテキスト間の意味的な類似度を測定する方法を提案する．

1.4 論文の構成

本論文は，序論と結論と含めて全体を 5 章で構成している．第 1 章の序論に続いて，第 2 章では，多言語処理への対応および意味構造との統合を行いやすく，かつ日本語の特性に適応した単語依存構造に基づく構文解析の枠組みを提案し，その妥当性について検証した結果について述べる [5, 6]．提案する構文構造に基づいて大規模な言語コーパスを構築し実際の構文解析器にモデルを実装して検証した結果，単語依存構造に基づく構文解析は，従来の文節依存構造に基づく構文解析と同等以上の性能が実現可能でかつ依存構造に

付加された文法機能ラベルによって，より詳細な情報が獲得できることを述べる．本論文で提案する日本語の単語依存構造は，構文構造の多言語共通仕様である UD との互換性を考慮しつつ，日本語処理に必要な情報と構造を持つことを実現している．日本語の単語依存構造による構文解析の枠組みの提案から大規模データ構築による解析器適用の検証結果については，文献 [5, 7] に述べている．単語依存構造の一種である UD については，文献 [3, 7] で述べている．単語依存構造コーパス構築の基盤となった日本語の句構造コーパスの構築については，文献 [8] で述べている．

第 3 章では，形態素情報を保持する短い単語（短単位）と構文情報を保持する長い単語（長単位）からなる階層的単語構造に基づき，日本語の構文解析を行う方法について述べる [9, 10]．依存構造単位である長単位の同定と長単位間の依存構造の同定は独立の課題ではないため，それらを逐次的に行うと前段の処理の誤りが後段の処理に伝播する可能性がある．本章では，長単位の同定と依存構造の同定を同時に行うことにより誤りの伝播を防止しつつ，両者の整合性を高めることを可能にする構文解析方法について述べる．日本語の単語依存構造解析の構文解析アルゴリズムについては，文献 [9, 10] で述べている．

第 4 章では，単語依存構造による構文構造から変換された意味構造を比較することによって，テキストが表現する意味内容の類似性を測定する方法を提案する [11, 12]．対になったテキストの意味内容を比較したときに，両者が同一の内容を表しているのか，一方が他方を包含しているのか，あるいは両者の内容が矛盾しているのかを識別することは，機械翻訳の評価や対話処理など知的な情報処理にとって重要な要素である．同一の意味内容を表現する自然言語の表現は無数に存在すると考えられるので，テキストを比較する際には表層的な情報を直接比較するのではなく，一旦抽象的な表現に変換してから行うことが適切な方法であると思われる．また，提案する手法は抽象的な表現の比較を行うことによって，異なる言語間の比較を同一言語同士の比較と同様の方法で行うことができる．また，比較対象となる文それぞれの意味構造の対応付けを行うため，意味内容の類似している点，相違している点を提示することができる．意味グラフを用いたテキスト意味類似度測定方法について，単言語のテキスト間に関しては文献 [11]，多言語テキスト間への拡張に関しては文献 [12] で述べている．第 5 章で以上の成果についてまとめる．

第2章　日本語の文法機能タイプ付き単語依存構造解析

2.1　はじめに

自然言語における最も基本的な構造情報は文法的な構造を表す構文構造であるが，各言語の特性等によって慣習的に使用されている体系には違いがあり，多言語処理において構文構造を扱う場合には異なった体系の変換等が問題になる．日本語も例外ではなく文節という独自の単位に基づく構文構造が最もよく使用されており，多言語処理を行う際に障壁となっている．本章では，日本語を多言語処理の中で扱うにあたって，他の言語と共通的な枠組みに基づきかつ日本語の言語現象を表現可能な構文構造の表現方法について述べる．構文構造を表現する形式には，1.2 節で述べたように主に句構造による表現形式と依存構造による表現形式があるが，本論文では比較的各言語の文法的特性の影響を受けにくい依存構造による構文構造を扱う．依存構造にも様々な種類が存在するが，基本的な特徴を決定するのは依存構造の単位である．例えば，依存構造の単位を単語にした場合は単語と単語の関係を基本とした細かな構造が表現されるが，複数の単語が連結した単位（句や文節など）とした場合は大きな単位での依存構造となり大局的な情報が重視される構造となる．英語をはじめとする多くの言語では単語を単位とした依存構造が採用されているが，日本語では独自単位による依存構造が用いられてきたため，日本語と他の言語との対応付けを行うためには，日本語を単語単位の依存構造に合わせるかあるいは他の工夫を行う必要がある．本章では，日本語を多言語処理に適応させる基盤として，単語単位の依存構造解析に基づく日本語の構文解析の枠組みを提案し，既存の構文解析器を適用した際の解析性能を網羅的に評価する．

以下，2.2 節 で日本語独自の構文構造である文節依存構造の問題点と，その解決法として日本語の単語依存構造を設計する際の課題について述べる．2.3 節で本論文で提案する日本語の文法機能タイプ付き単語依存構造の設計について説明し，2.4 節で実際に行ったコーパスの構築について述べ，2.5 節では本章に関連する研究について述べる．2.6 節では依存構造の単位や構造の異なる単語依存構造データから構文解析モデルを構築し，それらの違いが構文解析の精度に与える影響や，文法機能タイプから得られる述語項構造情報の精度について評価実験を行った結果について述べる．2.7 節で単語依存構造において検討すべき課題について述べる．最後に 2.8 節で本章の成果についてまとめる．

2.2 日本語独自の構文構造の問題点と解決のための課題

2.2.1 文節依存構造の問題点

日本語の構文解析は，標準的に文節間の依存関係により構成される構造である文節依存構造（あるいは，文節係り受け構造）に基づいて行われてきた．特に，CaboCha [13] や KNP [14] に代表される文節依存構造に基づく解析器は高い解析精度を実現して広範に利用され，日本語の自然言語処理全般の発展に大きく寄与してきた．しかしながら，文節依存構造による構文構造の表現には，2 つの問題点があることが指摘されている [8, 15] ．一つは依存構造の単位が構文の構成素 (constituent)[1]と整合しないこと，もう一つは格関係[2]や連体修飾節の種別[3]などの統語情報（以下，文法機能情報と呼ぶ）を依存構造の中に付加することが困難な点である．これらの問題点は，述語項構造解析などの構文構造と密接な関係を持つ処理や，機械翻訳における事前並べ替え [16] や多言語間の質問応答など他の言語との対応付けが必要な処理で不都合を生じる要因となる．本論文では構文の構成素と整合する単位に基づき，文法機能情報を付加できることを特徴とする単語依存構造解析に基づく構文解析を提案する．以下では，文節依存構造解析の 2 つの問題点を述語項構造解析との関係を例に具体的に説明する．

一つ目の問題点である依存構造の単位と構文の構成素との不整合は，構文解析結果の部分構造（一つ以上の文節が結合した単位）が，名詞句や動詞句などの構文の構成素と必ずしも一致しないということである．例えば，次の文 (1) のように文節を単位として表現された文から述語項構造を抽出することを考える．

(1) $|_{b1}$ 彼 が $|_{b2}$ 飲ん だ $|_{b3}$ ワイン と $|_{b4}$ 酒 の $|_{b5}$ リスト
he NOM *drink* PAST *wine* CONJ *sake* GEN *list*

(2) [$_{\mathrm{NP}}$ [$_{\mathrm{NP}}$ 彼 が 飲ん だ [$_{\mathrm{NP}}$ ワイン と 酒]] の リスト]

文 (1) の文節依存構造には，4 つの依存構造（係り受け構造）が存在している．その依存構造とは，並列構造を含む依存構造 $b3$ – $b4$ と，並列構造を含まない依存構造 $b1$ – $b2$，$b2$ – $b4$ と $b4$ – $b5$ である．また，文 (1) には，文 (2) で表されるように「ワインと酒」「彼が飲んだワインと酒」「彼が飲んだワインと酒のリスト」の 3 つの名詞句 (NP) が階層的に含まれている．しかし，文 (1) の文節を結合してできる単位は最初の 2 つの名詞句のどちらとも一致しない．この結果として，文 (1) の文節依存構造から述語項構造を抽出しようとしたとき，述語「飲んだ」の項として並列構造を含む名詞句である「ワインと酒」を直

[1]本論文では，名詞句，動詞句などの「句」の単位を指す．
[2]「誰が」「何を」「何に」と動詞などで表される名詞と述語の関係．
[3]名詞を修飾する節と名詞との関係の種類を示す．例えば「昨日見た-夢」では,「夢」は「見る」の対象になっているが,「月に行った-夢」では,「月に行った」は「夢」の内容になっている，など関係の違いを表す．

接取り出すことができない．この不一致は，他の言語との対応付けを行うときにも同様の問題を生じる．例えば文 (1) および文 (2) と対訳関係にある文 (3) において，並列構造を含む名詞句 “wine and sake” に対応付けるべき名詞句「ワインと酒」を直接取り出すことができない．

(3) $[_{\mathrm{NP}}$ a list of $[_{\mathrm{NP}}$ $[_{\mathrm{NP}}$ wine and sake] he drank]]

もう一つの問題点である文節依存構造において文法機能情報を付加することが困難な点であるが，情報が付加できない結果として統語的に異なる構造を区別できないことが起こる．その典型的な例として，内の関係の連体修飾節（関係節）と外の関係の連体修飾節（内容節や補充節）の区別がある．文 (1) は主名詞句（被修飾名詞句）となる「ワインと酒」が述語「飲む」の対格の格関係を持つ関係節[4]を含み，文 (4) は主名詞句「理由と事情」が述語との格関係がない外の関係の連体修飾節（内容節）[5]を含んでいる．

(4) $|_{b1}$ 彼 が $|_{b2}$ 飲ん だ $|_{b3}$ 理由 と $|_{b4}$ 事情 の $|_{b5}$ 説明
he NOM drink PAST reason CONJ situation GEN explanation

述語項構造を抽出する観点では，文 (1) の名詞句「ワインと酒」は，述語「飲む」の項として抽出するが，文 (4) の名詞句「理由と事情」は項として抽出しない．このような連体修飾節の違いを区別するためには，依存構造に文法機能情報を付加してそれぞれの統語的な機能を表示することが考えられる．しかし，文節依存構造の場合主名詞句と文節の結合単位が一致しないため，文 (1)，文 (4) それぞれの文節 $b2$ と $b4$ の間の依存構造に文法機能情報を付加しても，述語と主名詞句の間の関係を適切に表示しているとは言い難い．

2.2.2 日本語の単語依存構造設計と課題

従来の日本語における文節依存構造の問題点を解決することを目的として，構文の構成素を適切に扱い文法機能情報を明示的に扱うことのできる単語単位の依存構造による構文解析を提案する．単語依存構造はあらかじめ文節のような固定したチャンクを依存構造の単位として設定するのではなく，全ての関係を単語単位が結合した構造として表現することにより，構文構造を柔軟に表現することを可能にする．構文の構成素との整合性を考慮するには，句構造による構文解析が有力な選択肢と考えられるが，他の言語との対応付けのしやすさや日本語の柔軟な語順への対応のしやすさ，および文節依存構造のアノテーションからの移行のしやすさの点から依存構造による構文解析を採用した．ただし，依存構造の設計は，構文の構成素に基づいた構造および文法機能情報を表している句のラベル

[4]「ワインや酒」-を-「飲む」という関係を持つ．
[5]「飲む」-という-「理由や事情」という関係を持つ．

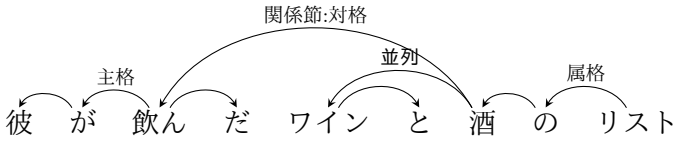


図 2.1: 単語単位の依存構造による構文構造の表示例.

(非終端記号) を持つことを特徴とする句構造を規範として，句構造の構造・情報を依存構造に変換する形で行った.

本論文で提案する単語依存構造[6]では，文 (1) に含まれる「ワインと酒」という並列構造は図 2.1 の例のように表現する[7]．すなわち,「ワイン」と「酒」という単語からなる依存構造に対して文法機能情報を表すラベル（以下，文法機能タイプ）「並列」を付加することにより，それぞれの単語を主辞とする構文要素からなる並列構造が存在することを示している．また,「ワインと酒」という名詞句は,「ワイン」「と」「酒」の 3 語から構成される依存構造と対応付けることができる．また，文 (1) と文 (4) の区別は，図 2.1 のように主名詞の主辞となる単語と連体修飾節の主辞となる述語の間の関係に「関係節」や「内容節」のような文法機能タイプを付加することで実現できる.

日本語の単語依存構造には他にも，多言語間で共通の構文構造表示を目指した UD [1, 17, 18] [8] の日本語仕様 [2, 3, 7] がある．UD の主目的は多言語間での仕様の共通化であるため，各言語の特徴的な言語現象に対するアノテーションは捨象される傾向があり， UD の粗い文法機能タイプでは例に挙げた関係節と内容節の区別を表示することができない. 本研究の主要な目的は，他の言語との対応付けが容易でかつ日本語において重要と考えられる文法機能情報を表示可能な単語依存構造を実現することである.

単語依存構造による構文解析は文節依存構造の課題を解決できる一方，構文解析器により自動解析を行う観点で見ると，解析結果として可能性のある依存構造の数が文節依存構造と比較して増大するため解析精度への影響が懸念される．単語依存構造の構造が文節依存構造に比べて複雑になる原因は，依存構造の単位の数が文節の数から単語の数に増加することと，依存構造の方向の制約が緩和されることである．依存構造の方向の制約とは，日本語の特性である主辞後置性（後方に位置する文節が前方に位置する文節の主辞になる性質）により，文節依存構造は単一方向の依存構造に限定されることである．単語依存構造ではその制約は当てはまらず，前方の単語が主辞になる場合と後方の単語が主辞になる場合の両方向の依存構造が考えられる.

また，日本語の単語依存構造としてどのような構造が適切であるか，すなわち依存構造

[6]後述するように 6 種類の依存構造の構成の仕方 (スキーマ) を提案する.

[7]本論文では，UD [1, 17] や Stanford typed dependencies [18] と同様に，主辞を起点として従属部に向かう方向の矢印により依存構造を表す.

[8]http://universaldependencies.github.io/docs/.

の主辞をどのように決定すればよいのかは自明ではない．森らは，単語間に依存構造を付与した大規模なコーパスを構築して，構文解析器の学習データとして適用した結果として，90%以上の精度が得られたことを報告している [19]．森らの依存構造は文節依存構造と同様に後方の語が主辞となる単一方向の依存構造から構成されるが，本論文で提案する単語依存構造は両方向の依存構造が含まれることにより構造の複雑さ増大するため解析精度への影響を検証する必要がある．また，依存構造に付与された文法機能タイプが解析器によりどの程度再現可能であるのかも確認する必要がある．

2.3 日本語文法機能タイプ付き依存構造の設計

本節では，構文の構成素と整合する部分構造から構成される，日本語の文法機能情報を表す文法機能タイプを持つ，という 2 点を満たす日本語の文法機能タイプ付き単語依存構造の設計について述べる．設計は，この 2 つの条件を満たしている句構造を規範として，木の構造と各句の非終端記号が持つ文法機能情報を単語依存構造へ変換するという方針に基づいて行った．句構造は田中らの句構造ツリーバンク [8] を用いた．

その方針の上で，(1) 依存構造を構成する単位を何にするか，(2) 日本語の構文構造に適切な依存構造をどのように定義するか，(3) どのような文法機能タイプ（依存関係タイプ）を定義するか，の 3 点を具体的に定める必要がある．本研究では，これらの点について以下のように対応した．

(1) 文法的機能を持つ単語単位（後述する長単位）を依存構造の単位とする

(2) 句構造木からの変換に基づいた 6 種類の依存構造スキーマを考える

(3) 句構造の非終端記号から変換した 35 種類の文法機能タイプを定める

本節の以降で (1) から (3) のそれぞれについて詳細を述べる．また，本節で定めた依存構造の単位，依存構造スキーマの解析精度に与える影響，および文法機能タイプの有効性を調べるために行った評価実験の結果については 2.6 節で述べる．

2.3.1 依存構造の単位

依存構造の構成単位を「単語」とする場合，その単語の定義を何にするかが全体設計に大きな影響を与える．本論文では明確に定められた基準により単語を認定する際の揺れが小さく斉一な単位として，現代日本語書き言葉均衡コーパス (以下，BCCWJ) [20, 21] で採用されている短単位 (Short Unit Word, SUW) を基礎として考える．短単位は，基準がわかりやすくアノテーション作業においても揺れが少ない，取り出した単位が文脈から

文	魚フライを食べたかもしれないペルシャ猫										
短単位	魚	フライ	を	食べ	た	か	も	しれ	ない	ペルシャ	猫
(SUW)	NN	NN	PCS	VB	AUX	P	PBD	VB	AUX	NNP	NN
長単位	魚フライ		を	食べ	た	かもしれない				ペルシャ猫	
(LUW)	NN		PCS	VB	AUX	AUX				NN	
文節	魚フライを			食べたかもしれない						ペルシャ猫	

図 2.2: 単語分割結果の例.

離れすぎない [22] という長所があり，単語についての詳細な情報を保持する解析処理の最小単位として適している．一方で，統語的な基本単位として依存構造を構成するには短過ぎる傾向がある．例えば,「かもしれない」のような機能語相当の働きを持つ複合辞は,短単位では「か」「も」「しれ」「ない」の 4 語に分割されるが，これらの間の依存関係を表現してもあまり意義のある関係は含まれない．そこで，同じく BCCWJ で採用されている長単位 (Long Unit Word, LUW) を依存構造を構成する単位として採用する．長単位は，一つの文節内を規則に従って自立語部分と付属語部分に分割することにより認定され，各文節は 1 語の自立語と 0 語以上の付属語から構成される．長単位では,「かもしれない」等の複合辞は助動詞相当の一語として扱われるため，重要度の低い依存関係を扱わなくてよいとともに構文構造を見通しよく表現することができる．

図 2.2 に，文を短単位，長単位に単語分割した例を示す．NN などの品詞シンボルは，表 2.1 に示しているものと同様である．短単位と長単位は階層的な関係になっているため，長単位の依存構造に基づく構文解析を行う場合にも，それぞれの長単位を構成する短単位の情報も解析器の素性として用いることができる．NN，VB などの品詞シンボルは後述する句構造ツリーバンクで定義された前終端記号を表しており，表 2.1 のように BCCWJ で定義された品詞と対応関係がある．この品詞シンボルは，2.4.1 節で述べる句構造から単語依存構造へ変換する規則が参照する．

2.3.2 依存構造スキーマ

日本語の文節依存構造では，各依存構造を構成する文節間で基本的に右側が主辞になる構造として定義されている（主辞後置型）．しかしながら，日本語において単語間の依存構造を考える場合には，主辞をどのように決定するか，すなわちどのような依存構造を構成するのが適切であるかは自明ではない．そこで本研究では,「構文解析結果の部分構造が構文の構成素 (constituent) と一致する」という要件を満たす句構造を出発点として，単語依存構造への変換方法を考えることにより，依存構造の設計を行った．

述語句（述語と後続する 0 語以上の助動詞や助詞で構成される句）とその項を構成素として含むような文に対して，二つのタイプの句構造を考える．一つは述語とその項を先

表 2.1: 句構造ツリーバンク [8] で前終端記号として用いられている品詞シンボル.

品詞 シンボル		BCCWJ の品詞	例
NN	General noun	名詞-普通名詞	橋，犬
NNP	Proper noun	名詞-固有名詞	日本，加藤
NPR	Pronoun	代名詞	それ，ここ
NV	Verbal noun	名詞-普通名詞-サ変可能	導入，議論
NADJ	Adjective noun	形状詞 / 名詞-普通名詞-形状詞可能	広大，高速
NADV	Adverbial noun	名詞-普通名詞-副詞可能	当面，今
NNF	Formal noun (general)	助詞-準体助詞	こと，の
NNFV	Formal noun (adverbial)	接尾辞-名詞的-副詞可能	(工事) 中，以上
NNFJ	Formal noun (adjective)	接尾辞-形容詞的	よう
PX	Prefix	接頭辞	準，旧
SX	Suffix	接尾辞	所，部
NUM	Numeral	名詞-数詞	123，二十三
CL	Classifier	名詞-普通名詞-助数詞可能	(5) 本，(10) 匹
VB	Verb	動詞-一般	走る，見る
ADJ	Adjective	形容詞-一般	早い，高い
ADNOM	Adnominal adjective	連体詞	その，こうした
ADV	Adverb	副詞-一般	ゆっくり，すっかり
PCS	Case particle	助詞-格助詞	が，を，に，と
PBD	Binding particle	助詞-係助詞	は，も
PADN	Adnominal particle	助詞-格助詞 (連体助詞「の」)	の
PCO	Parallel particle	助詞-格助詞 (並立助詞「と」)	と
PCJ	Conjunctive particle	助詞-接続助詞	て，が
PEND	Sentence-ending particle	助詞-終助詞	か，ね，よ
P	Particle (others)	助詞-副助詞	など，だけ
AUX	Auxiliary verb	助動詞	だ，ます
CONJ	Conjunction	接続詞; 副詞	ところで，しかし
PNC	Punctuation	補助記号-句点/読点	、,。
PAR	Parenthesis	補助記号-括弧	「,」
SYM	Symbol	補助記号	・
INTJ	Interjection	感動詞	ああ，はい
FIL	Filler	感動詞-フィラー	えー

に結合する構造，もう一つは述語と後続する助詞，助動詞などの機能語を先に結合する構造であり，本論文では，前者を述語項結合型（あるいは 1 型）, 後者を述語文節結合型（あるいは 2 型）と呼ぶ．図 2.3 は，同じ文を述語項結合型，述語文節結合型により構成した句構造の例である．四角で囲んだ VP は，述語を含む句で優先的に結合する部分木の非終端記号を表す．その他の非終端記号 PP-OBJ，IP-REL_sbj は，それぞれ対格の後置詞句，主格の空所を持つ関係節を表す．述語項結合型は以下の文 (5)a や対応する英語の文 (6)a のように HPSG 等で用いられる生成文法的な考えに基づく構造，述語文節結合型は文 (5)b や対応する英語の文 (6)b のような文節に類似した構造と捉えることもできる．

(5) a. [[[$_{\mathrm{VP}}$ 猫が 魚を 食べ] た] かもしれない]
S O V aux aux

b. [猫が [魚を [$_{\mathrm{VP}}$ 食べ た かもしれない]]]
S O V aux aux

(6) a. [The cat [may have [$_{\mathrm{VP}}$ eaten the fish]]] .
S aux aux V O

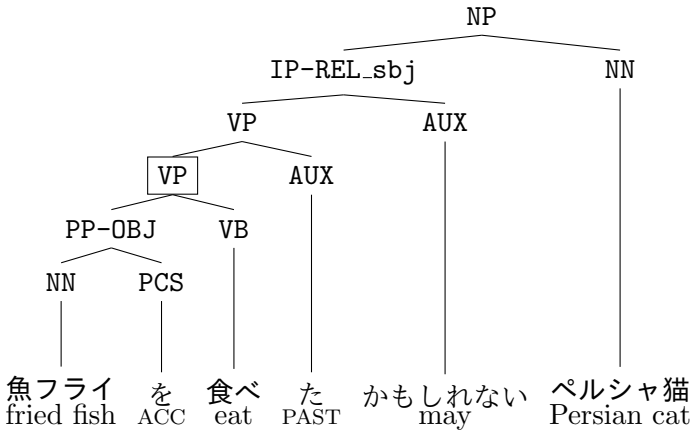
b. [The cat [[$_{\mathrm{VP}}$ may have eaten] the fish]] .
S aux aux V O

これらの句構造から単語間の依存構造へ変換することを考える．句構造が二分木で構成されている場合を考えると，各中間ノードの分岐ごとに左右どちらの要素が主辞になるかを順に決定することで依存構造を構築することができる．その際，日本語の文を構成する主要な要素である，名詞句+機能語（主に助詞）から成る後置詞句，述語+機能語（主に助詞，助動詞）から成る述語句それぞれにおいて，内容語と機能語のどちらを主辞と扱うかで 2 つのタイプに分ける[9]．ここでは便宜的に，後置詞句で機能語を主辞と扱うものを項機能語主辞 (AF-head)，内容語を主辞と扱うものを項内容語主辞 (AC-head)，述語句で機能語を主辞と扱うものを述語機能語主辞 (PF-head)，内容語を主辞と扱うものを述語内容語主辞 (PC-head) と呼ぶ．この主辞型の組合せが 4 通りあり，これらを 2 タイプの句構造の結合型への適用すると 8 通りになるが，述語内容語主辞の 2 つの主辞型については，句構造結合型の 1 型，2 型に適用した結果が同じ依存構造になるため，実際には 表 2.2 に示す 6 タイプの依存構造が構成されることになる．これらを依存構造スキーマと呼ぶ．

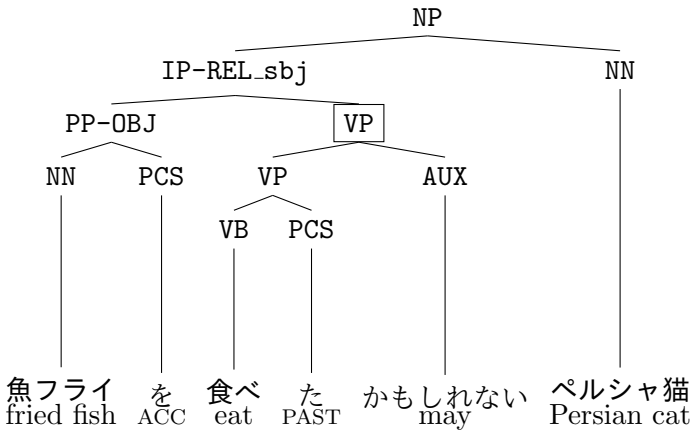
AF-head+PF-head は主辞後置の原則に則ったスキーマで，述語項結合型 (1 型) に適用したものを，主辞後置 1 型 (Head Final type 1, HF_1)，述語文節結合型 (2 型) に適用したものを，主辞後置 2 型 (Head Final type 2，HF_2) とする．

主辞後置型から述語句で内容語 (述語) を主辞に変えた AF-head+PC-head は，述語と項がより近い依存関係を持つようにしたもので，述語項結合型，文節結合型いずれに適用

[9]多くの場合，内容語を主辞とするものを意味主辞 (semantic head)，機能語を主辞とするものを統語主辞 (syntactic head) とみなすことができるが，ここでは両者を区別せずに扱う．



述語項結合型 (1 型)



述語文節結合型 (2 型)

図 2.3: 述語句の結合型の違いによる 2 種類の句構造の構成.

したものも同じ依存構造となる．これを述語内容語主辞型 (Predicate Content word Head type, PCH) とする．また，主辞後置型から後置詞句で内容語を主辞にした，AC-head+PF-head を述語項結合型，述語文節結合型それぞれに適用したものを，それぞれ 項内容語主辞 1 型 (Argument Content word Head type 1, ACH_1) ，項内容語主辞 2 型 (Argument Content word Head type 2, ACH_2) とする．

さらに，述語句，後置詞句の両方において内容語を主辞とした AC-head+PC-head は，述語と項の内容語同士が直接依存構造を持つようにしたもので，これを 内容語主辞型 (Content Word Head type, CWH) とする．この型は内容語間の関係を重視し，少なくとも一方が内容語となる単語間に対して依存構造を考える UD と類似した依存構造スキーマである．CWH と UD とは並列構造における主辞の扱い[10]や文法機能タイプなどが異なる．

[10]UD では並列要素のうち先頭の要素が主辞になる．

表 2.2: 句構造の結合型，主辞型の組合せと依存構造スキーマ．

主辞型 (述語句)	句構造結合型	主辞型 (後置詞句)	
		項機能語主辞 (AF-head)	項内容語主辞 (AC-head)
述語機能語主辞 (PF-head)	述語項結合 (1 型) 述語文節結合 (2 型)	主辞後置 1 型 (HF_1) 主辞後置 2 型 (HF_2)	項内容語主辞 1 型 (ACH_1) 項内容語主辞 2 型 (ACH_2)
述語内容語主辞 (PC-head)	述語項結合 (1 型) 述語文節結合 (2 型)	述語内容語主辞型 (PCH)	内容語主辞型 (CWH)

これら 6 通りの依存構造スキーマによる構文解析例を図 2.4 に示す．図中で枠で囲まれた文字列は句構造において先に結合する構造を示す．各スキーマによって，統語的関係にある語の間の距離が異なることから，解析器で再現する際の難易度も異なると考えられる．2.4 節でこれらの依存構造スキーマに基づいて構築したコーパスについて述べ，2.6 節で解析器への適用面から比較する．

2.3.3 文法機能タイプ

本研究では，SD を参考に 35 種類の文法機能タイプを定義した[11]．表 2.3 に，定義した文法機能タイプを示す．参考として，日本語 UD version 2 において定義が類似した文法機能タイプを併記している．ただし UD version 2 との対応は近似的なものであり，必ずしも同じ定義を持つとは限らない．例は，*dependency_type* (主辞, 従属部) の形で表記し，“-” は長単位の境界を表す．長単位列のうちどの長単位が主辞になるかは，依存構造スキーマによって異なる．SD との主な違いは，*nsubjpass*, *auxpass*, *agent* など受動態特有のタイプや *det* (determiner), *expl* (expletive), *xcomp* (open clausal complement) など，日本語では不要もしくは必要性の低いと考えられるタイプを採用していないこと，日本語の多様な連体修飾節に対応するため関係節に関するタイプ *rcmod* を細分化したことである．日本語の文法的特徴を反映させるため，日本語句構造ツリーバンクの非終端記号を文法機能タイプに変換することにより，元の句構造の情報が利用されるように定めた．以下に，カテゴリごとに定義した文法機能タイプについて説明する．

格関係のタイプ　述語と格関係を持つ項との関係に付与する文法機能タイプであり，必須項 (argument) タイプと，付加項 (adjunct) タイプに大別される．必須項タイプは，*nsubj*（主格），*dobj*（対格），*iobj*（与格） を，付加項タイプは，*lmod*（場所格），*tmod*（時間

[11]前置詞句や等位接続詞を含む構造に対して，SD で定義されている collapsed dependency type は採用していない．

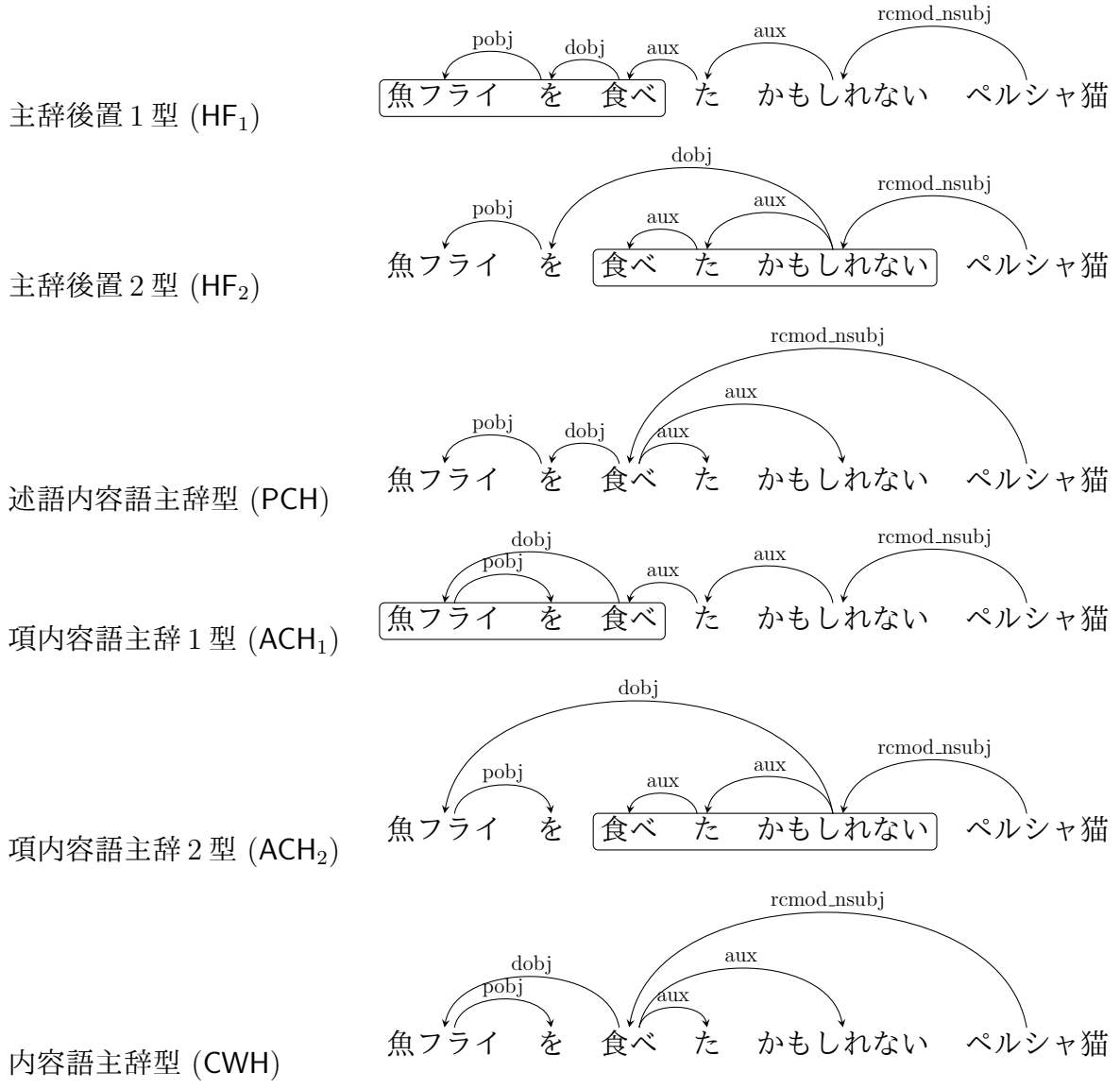


図 2.4: 各依存構造スキーマによる構文解析例.

格)，*arg*（その他の格）を定義した．アノテーションは，述語項構造情報に基づいて行うが，句構造ツリーバンク [8] の情報を利用する場合は，非終端記号に付加された文法機能ラベル (`-SBJ`（主格），`-TMP`（時間格）など）を変換して行う．

節のタイプ　日本語において節と句の境界は曖昧であるが，本論文の文法機能タイプは，句構造ツリーバンク [8] において定義されている節のタイプに基づいている．表 2.4 に句構造ツリーバンクで分類した節の種類と対応する文法機能タイプを示す．連体修飾節については，内の関係の連体修飾節（関係節）とそれ以外の外の関係の連体修飾節を区別するために，それぞれ，*rcmod*， *ncmod* という文法機能タイプを割り当てている．さらに関係節の空所 (gap) になっている格の種類を区別するため，*rcmod* を，*rcmod_nsubj*，*rcmod_dobj*，*rcmod_iobj*，*rcmod* の 4 つの文法機能タイプに細分化した．これにより，連

表 2.3: 文法機能タイプ.

カテゴリ	文法機能タイプ		例	UDv2
格関係	*nsubj*	主格	*nsubj* (食べる, 猫-が)	*nsubj*
(必須項)	*dobj*	対格	*dobj* (食べる, 魚-を)	*obj*
	iobj	与格	*iobj* (渡す, 生徒-に)	*iobj*
	post	属格	*post* (鈴, 猫-の)	*nmod*
格関係	*tmod*	時間格	*tmod* (食べる, 正午-に)	*obl*
(付加項)	*lmod*	場所格	*lmod* (食べる, テラス-で)	*obl*
	arg	その他の格	*arg* (食べる, スプーン-で)	*obl*
修飾句	*amod*	連体修飾 (形容詞)	*amod* (車, 赤い)	*advcl*
	nmod	連体修飾 (名詞)	*nmod* (三-メートル, 全長)	*nmod*
	advmod	連用修飾	*advmod* (走る, 早く)	*advmod*
	nummod	数詞	*nummod* (分, 50)	*nummod*
関係節	*rcmod_nsubj*	主格空所	*rcmod_nsubj* (猫, 魚-を-食べる)	*acl*
(内の関係)	*rcmod_dobj*	対格空所	*rcmod_dobj*(魚, 猫-が-食べる)	*acl*
	rcmod_iobj	与格空所	*rcmod_iobj* (問題, 直面する)	*acl*
	rcmod	その他	*rcmod* (椅子, 背-が-高い)	*acl*
内容節	*ncmod*		*ncmod* (臭い，焼い-た)	*acl*
・補充節				
（外の関係）				
引用節	*ccomp*		*ccomp* (認める, 話し-た-と)	*ccomp*
疑問節	*qcomp*		*qcomp* (わかる，する-か)	*qcomp*
副詞節	*advcl*		*advcl* (寝る, 疲れ-て)	*advcl*
並列句・節	*conj*		*conj* (酒, ワイン-と)	*conj*
同格	*appos*		*appos* (タマ, 白猫)	*appos*
複合語の関係	*nn*	複合名詞の関係	*nn* (本社, 大阪)	*compound*
	pmwe	複合辞等の関係	*pmwe* (いう, と)	*fixed*
	vmod	複合動詞の関係	*vmod* (くる, やっ-て)	*compound*
	vb	サ変名詞の関係	*vmod* (する, 解決)	*aux*
機能語の関係	*aux*	助動詞との関係	*aux* (食べ, た)	*aux*
	pobj	助詞との関係	*pobj* (が, 猫)	*case*
	cc	並列助詞との関係	*cc* (と, 猫)	*cc*
	cop	判定詞との関係	*cop* (だ, 猫)	*cop*
	mark	接続助詞との関係	*mark* (食べ, て)	*mark*
	pend	終助詞との関係	*pend* (か, する-の)	*aux*
その他の関係	*pnc*	句読点との関係	*pnc* (する, 。)	*punct*
	par	括弧との関係	*par* (重要, 」)	*par*
	root	文全体の主辞	*root* (-, 走る)	*root*
	dep	未定義の関係等	*dep* (,)	*dep*

体修飾節の主辞の述語と主名詞句の関係を区別し，文法機能タイプにより述語項構造を捉えることが可能になる．外の関係の連体修飾節には，内容節（月に行った-夢）と補充節（月に行った-結果）があるが，句構造ではこれらを区別をせず IP-ADN としており，依存構造でも文法機能タイプを *ncmod* とした．

語・句による修飾関係のタイプ　格関係以外の修飾関係（ただし節による修飾を除く）に付与する文法機能タイプで，*nmod*（名詞類による修飾関係），*amod*（形容詞類による修飾関係），*advmod*（副詞類による修飾関係）などがある．また，名詞句+格助詞「の」による連体修飾句の関係は *post* と定義した．

並列関係のタイプ　並列関係を持つ要素間に付与する文法機能タイプとして，*conj*（並列），*appos*（同格）を定義した．並列要素と並立助詞（「と」「や」など）との関係には，SD と同様に *cc* (coordination) を定義した．これは並立要素をマークしている関係を明確にすることで，並列構造を抽出しやすくするためである．

機能語関係のタイプ　機能語と内容語，機能語と機能語の間の依存関係に付与する文法機能タイプで，以下のような種類のタイプを定義している．*pobj*（助詞と内容語との関係），*aux*（助動詞と内容語との関係），*cop*（判定詞と名詞述語の関係）, *mark*（接続助詞と節の主辞との間の関係 ）などがある．

その他のタイプ　*pnc*（句読点との関係），*par*（括弧類との関係）など記号類を含む文法機能タイプや，*nn*（複合名詞の関係），*vmod*（複合動詞の関係）など複合語に関係する文法機能タイプがある．他に未定義の関係として *dep* を用意している．

様々な文法機能タイプの中でも，特に格関係のタイプと関係節のタイプを導入することの利点は，これらのタイプが付加された依存構造を辿ることにより，述語項構造が抽出できることである．例えば，図 2.4 の 主辞後置 1 型 (HF_1) の依存構造の場合には，述語と項となる内容語間に 2 つの path を辿ることができる．すなわち,「魚フライ (NN)←*pobj*←***dobj***←食べ (VB)」と「食べ (VB)←*aux*←*aux*←***rcmod_nsubj***← ペルシャ猫 (NN)」の path である．二つの文法機能タイプ *dobj* と *rcmod_nsubj* を識別することにより，述語「食べる」の直接目的語と主語としてそれぞれ「魚フライ」と「ペルシャ猫」を抽出することができる．

2.3.4　文法機能タイプ付き依存構造による構文解析例

提案した文法機能タイプ付き依存構造により，詳細化される構文情報について例を示す．本節で示す例は，述語内容語主辞スキーマ PCH によるものである．

表 2.4: 節の種類と句構造ツリーバンクで用いられている非終端記号，単語依存構造で導入した文法機能タイプ.

節の種類			非終端記号	文法機能タイプ	
主節			IP-MAT	-	
従属節	副詞節		IP-ADV	*advcl*	「帰ったら」+ 眠った
	連体修飾節	関係節 (内の関係)	IP-REL	*rcmod*	「昨日買った」+本を〜
		内容節 (外の関係)	IP-ADN	*ncmod*	「魚を焼いた」+臭いが〜
		補充節 (外の関係)	IP-ADN	*ncmod*	「家に帰る」+ 途中で
	補足節	補文 (名詞化)	CP-NNF	*ncmod*	「雨が降った」+ のは意外だった
		引用節	CP-THT	*ccomp*	「遅刻した」+ と認めた
		疑問節	CP-QUE	*qcomp*	「明日起きられるか」+わからない
並列節			IP-COORD	*conj*	「顔を洗って」+歯を磨いた

述語項構造　格関係の文法機能タイプにより，必須項と付加項との区別を含めて，述語と項の関係を明示することが可能になる．また，連体修飾節と主名詞の関係を明示することが可能になる．図 2.5 の一番上に示す例では，動詞「食べる」に対する格関係が，*nsubj*（主格），*dobj*（対格），*lmod*（場所格）により明示されている．次の二つの例では，動詞「食べる」と主名詞句との関係が，関係節であるか (*rcmod_dobj*)，内容節/補充節であるか (*ncmod*) が明示的に区別されている．これらの文法機能情報を構文解析結果から得られることにより，別に解析を行うことなく述語項構造の情報を取得できる．

並列構造　並列構造をスコープを含めて的確に表示することが可能になる．図 2.6 の上の例では，*cc*, *conj* のリンクにより，結ばれている名詞「猫」と「犬」が並列関係にあることが明示されている．さらにそれぞれの主辞から従属部へと辿ることにより，それぞれが「隣町に住んでいる猫」,「初めて見る犬」の名詞句から構成されていることも捉えることができる．また，図 2.6 の下は部分並列の例である（[23] の部分並列の例を一部改変）．例のように，複数の項の組を一つの述語が共有する場合でも，単純に全ての項の主辞を同一の述語（「かし」）とし，項が組になって部分並列の構造になっていること（「本を」と「太郎に」,「ノートを」と「三郎に」）は表示しない．このため，部分並列の場合，依存構造の情報のみから正しい述語項構造を取得することはできないが，これは，直接統語的に関係する語の間の依存構造を優先して構成する方針を採っているためである．

2.4　既存コーパスの変換による単語依存構造コーパスの構築

本節では，提案する文法機能タイプ付き単語依存構造コーパスの初期構築を行った際の方法について述べる．2.3 節で述べたように，本論文で提案する単語依存構造は句構造を

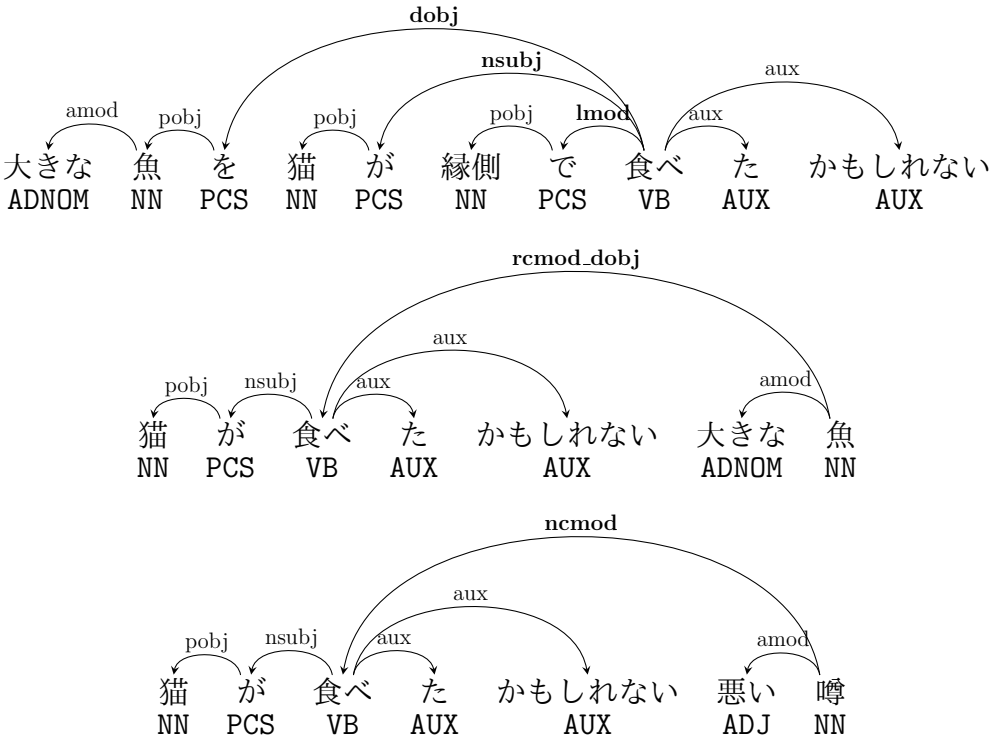


図 2.5: 文法機能タイプ付き依存構造 (PCH) による構文解析例（述語項構造，関係節）.

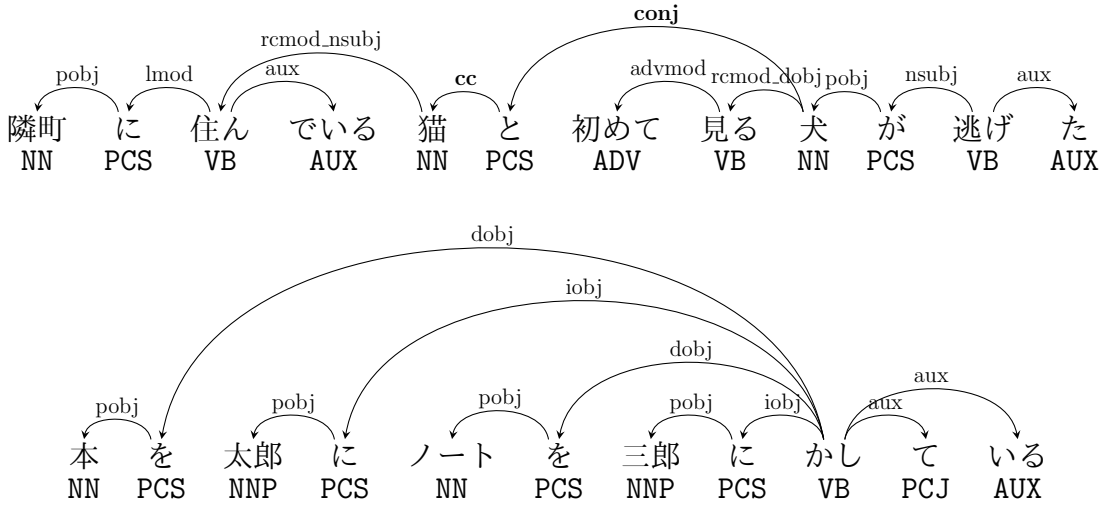


図 2.6: 文法機能タイプ付き依存構造 (PCH) による構文解析例（名詞句の並列，部分並列）.

規範として設計しているため，依存構造の連結した単位（部分構造）や文法機能タイプは，句構造の部分木の単位や非終端記号との親和性が高い．また，新たなアノテーションスキームに基づくコーパスを構築する際には，ゼロからアノテーションを行うより，既存のコーパスから一定の規則を定めて変換を行う方が，設計の変更への対応や一貫性を保持するのに有利であると考えられる．そこで，本研究では提案する単語依存構造に基づく

部分木の子の非終端記号 （左のノード：右のノード）	主辞となるノード (PCH)	主辞となるノード (CWH)
NN:PCS	right	left
NP:PCS	right	left
NN:PBD	right	left
NP:PBD	right	left
VB:AUX	left	left
VP:AUX	left	left

図 2.7: 述語内容語主辞型 PCH，内容語主辞型 CWH の主辞規則の例．

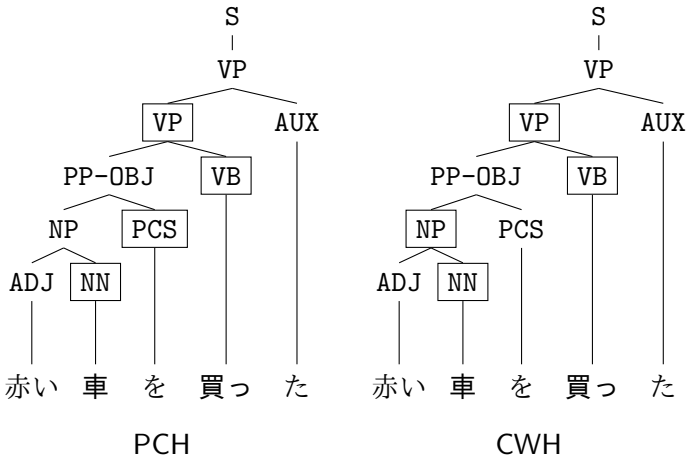


図 2.8: 変換元になる句構造の例．

コーパスの最初の構築を，既存の句構造コーパスの持つ構造と非終端記号の情報を変換する方法により行った．一般的に依存構造に比べて句構造木の方が情報量が多く，句構造木が存在する場合には句構造木から依存構造への自動変換で十分な質の依存構造が構築できると考えられる．句構造木から依存構造へは不可逆の変換であり，依存構造への変換によって情報量を落とすにもかかわらず依存構造を採用している理由は，依存構造は述語項構造等の意味解析と対応付けを行うのに十分な情報量を保持していると期待されるためおよび f，依存構造のアノテーションは句構造木に比べて難易度が低く将来依存構造のデータを直接構築する場合にも低いコストで行えるためである．また，解析器による自動解析を行う場合にも依存構造の方が句構造木の方が曖昧性が低く解析精度面でも有利である．本論文の単語依存構造で述語項構造情報がどの程度保持されるかについては，2.6.4 節 で評価を行う．以下では，実際に 2 種類の句構造コーパスを変換することにより構築した単語依存構造コーパスの統計量についても述べる．

2.4.1 句構造木から依存構造への変換

2.3.2 節，2.3.3 節で述べたように，依存構造スキーマ，文法機能タイプの設計は句構造木からの変換に基づいて行った．同様に文法機能タイプ付き単語依存構造コーパスの構築を，句構造ツリーバンク [8] から句構造木の構造と付加された文法機能ラベル（非終端記号）を自動変換することよって行うことを考える．前述したように句構造木は依存構造に比べて情報量が多いため，依存構造コーパスの初期構築は句構造木からの自動変換により十分な質で行えると考えられる．将来的には構築コストの高い句構造木を経由せず，平文のテキストから直接依存構造を低いコストで構築することができる．

この句構造木は完全な二分木で構成されており，非終端記号には，文法機能タイプへの変換に必要な格の情報，関係節の情報等が含まれる[12]．

句構造木から依存構造への変換は，二分木の各分岐に対して主辞決定規則（以下，主辞規則）を適用し，非終端記号の組合せにより左右どちらの子を主辞とするかを決定することで行う．この主辞規則は，依存構造スキーマに応じて異なる規則を用意する．図 2.7 に述語内容語主辞型 PCH, 内容語主辞型 CWH の各スキーマの主辞規則の例を示す．図 2.8 は，変換元となる句構造の例と，図 2.7 の主辞規則を適用した結果を示す．図 2.8 で，四角い枠で囲まれた非終端記号は，各分岐ごとに主辞規則を用いて決定された主辞を示している．

文法機能タイプは，句構造木の各分岐ごとに周辺のノードの非終端記号を参照する文法機能タイプ変換規則によって決定する．図 2.9 は，文法機能タイプ変換規則の例である．ある葉ノードの単語 D（前終端記号 P） と D の主辞との依存構造の文法機能タイプを決定する際には，D から根ノードの方向にたどり D が主辞とならない最初のノード C，C の左の子ノード L， C の姉妹ノード H を参照する．図中の “*” は，任意の非終端記号を表す．

図 2.10 は，図 2.8 の句構造に，図 2.7 の主辞規則と図 2.9 の文法機能タイプ変換規則を適用して変換した 2 タイプの依存構造 PCH, CWH の結果を示す．このように各依存構造スキーマ用の主辞規則と文法機能タイプ変換規則を使用することにより対応するスキーマに基づく依存構造コーパスを構築することができる．

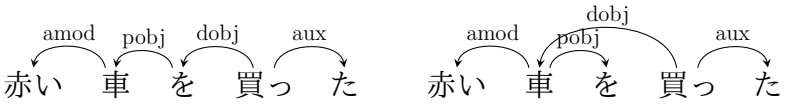
2.4.2 構築したコーパスの統計量

単語依存構造への変換する句構造ツリーバンクとして，京都大学テキストコーパス version 4.0 （以下，京大コーパス）[24] の 40,000 文のうち，文法機能ラベル付きの句構造木が構築されている 19,953 文（一般記事 9,953 文，社説記事 10,000 文），BCCWJ のコアデータのうち同じく句構造木が構築されている 18,400 文 を用いた．本論文ではこれらの

[12] 一部は Kaede Treebank として公開されている (`https://github.com/mynlp/kaede/`).

依存元の葉ノード	非終端記号			文法機能タイプ	
の前終端記号 P	C	L	H	（提案スキーマ）	（参考: UD）
PCS	PP-	NN	∗	*pobj*	*case*
PCS	PP-	NP	∗	*pobj*	*case*
NN	VP	PP-OBJ	∗	*dobj*	*obj*
PCJ	VP	VP	∗	*mark*	*mark*
AUX	VP	VP	∗	*aux*	*aux*
ADJ	NP	∗	∗	*amod*	*acl*

図 2.9: 文法機能タイプ変換規則の例.



述語内容主辞型 (PCH)　　内容語主辞型 (CWH)

図 2.10: 句構造から変換された依存構造の例.

句構造から単語依存構造に変換したコーパスを以下それぞれ，KyotoDep, BCCWJDep と呼ぶ．述語内容語型スキーマ (PCH) に基づく依存構造に変換したコーパスに含まれる短単位・長単位の数，各文法機能タイプの出現頻度の統計量を表 2.5 に示す．

付与された文法機能タイプの統計量から，各コーパス・サブコーパスの構文的な特徴を概観することができる．例えば，書き言葉で書かれたコーパス（KyotoDep や BCCWJDep の出版・書籍や新聞など）では，主格の関係にある *nsubj* は文数よりも多くあらわれる傾向があるが，話し言葉に近い BCCWJDep の知恵袋，ブログでは，文数よりも極端に下回っていることがわかる．また，内の関係の関係節の割合は，固めの書き言葉で書かれている白書，新聞で高く，次いで出版・書籍と雑誌，最も低い割合が，知恵袋，ブログという傾向が見られる．このように，文法機能タイプを様々なコーパスにアノテーションすることは，それぞれのコーパスに含まれている文の文体や構文的な特徴を分析することにも有効であることが期待される．

2.5　関連研究

日本語の構文構造に文法機能情報を付加する研究としては，語彙化文法に基づくものがある．Sag らにより提唱された主辞駆動型句構造文法 (Head-driven Phrase Structure Grammar, HPSG) [25] がその代表的なものである．HPSG の日本語文法を構築した研究には，理論的に精緻な日本語の文法 JPSG (Japanese Phrase Structure Grammar) を提案した郡司の研究 [26] ，話し言葉を対象に実用的な文法を構築した永田らの研究 [27]，広い言語現象をカバーし意味表現 MRS (Minimal Recursion Semantics) [28] と統合した文

表 2.5: 構築したコーパスの統計量（依存構造スキーマ PCH ）.

	KyotoDep	BCCWJDep					
			PB	PM	PN	OW	OC	OY
		合計	出版 · 書籍	雑誌	新聞	白書	知恵袋	ブログ
文	19,953	18,400	3,583	3,951	4,820	1,903	2,118	2,105
短単位	550,501	445,463	88,231	85,826	115,356	81,621	38,141	36,288
長単位	424,978	354,086	74,866	71,429	88,005	56,467	32,638	30,681
nsubj	25,179	17,943	4,098	3,922	4,935	2,029	1,597	1,362
dobj	17,770	12,859	2,726	2,448	3,567	2,449	876	793
iobj	12,450	8,917	1,997	1,812	2,286	1,557	631	634
tmod	4,122	2,015	185	312	910	377	97	134
lmod	2,197	1,493	330	336	543	89	87	108
rcmod_nsubj	5,206	3,616	777	663	4,935	781	156	192
rcmod_dobj	918	703	178	134	3,567	130	32	43
rcmod_iobj	510	587	99	89	2,286	223	28	26
ncmod	9,580	8,785	2,115	1,852	1,851	1,434	839	694
ccomp	2,964	1,962	422	337	709	93	268	133
advcl	15,222	14,701	3,708	3,130	3,107	1,826	1,583	1,347
conj	7,771	6,285	961	1,083	1,540	1,934	365	402
appos	907	729	91	145	261	169	21	42

法 JACY を構築した Siegel らの研究 [29] がある．そのうち永田らは構築した文法に基づく構文解析器を実現し，Bond らは上記の文法 JACY に基づいて大規模なツリーバンクを構築しツリーバンクから学習した解析モデルにより構文解析器を実現している [30].

HPSG の他には，日本語の語彙機能文法 (Lexical Function Grammar, LFG) [31] を構築した増市らの研究 [32]，日本語の組合せ範疇文法 (Combinatory Categorial Grammar, CCG) を提案した戸次の研究 [33], その文法に基づいたツリーバンクを既存の文節依存構造コーパスと述語項構造データから構築した植松らの研究 [34] などがある．

これらのアプローチでは，統語情報，意味情報，語用情報等を統合した精緻な情報を扱えることが大きな利点である．しかし，解析は基本的に素性構造とよばれる素性（属性）と値の組を複数持つ構造[13]を矛盾なく統合する単一化という操作に基づく方法で行われ，単一化は規則を満たす組み合わせが爆発的に生じることで高い計算コストを必要とするいう実用上の短所がある．また，文節依存構造に比較すると正解データを作成するために必要な情報を付加するコスト（アノテーションコスト）が高いため，大規模なデータが構築しにくく，また対象ドメインに対して新たにコーパスを用意するのも容易ではない．

[13]LFG, HPSG では，語彙，書き換え規則（文法規則）ともに素性構造で表現される．

語彙化文法に基づく方法に比較して計算コスト，アノテーションコスト両面で軽量で，かつ文法機能情報を活用できる解析の枠組みとして，文脈自由文法 (Context Free Grammar, CFG) によるもの，依存構造文法によるものがある．日本語の CFG を構築して解析器を実装，評価したものとしては，野呂らの研究 [35] があるが，彼らの研究では前節に挙げたような，連体修飾節の区別や述語項構造の情報は扱っていない．田中らは，非終端記号としてこれらの文法機能情報を埋め込んだ句構造木ツリーバンクを構築して確率自由文脈文法 (Probabilistic Context Free Grammar, PCFG) の構文解析モデルを評価した結果，文節依存構造解析相当の精度を維持しながら文法機能タイプを扱えることを報告している [8]．また，Butler らは，Penn 通時コーパスの規約に従いより詳細な句構造コーパスとして Keyaki Treebank を構築し [36]，PCFG による構文解析を行った結果についても報告している [37]．しかし，日本語の句構造木の構築は，従来の文節依存構造の構築よりも難易度が高いため，ドメイン適用のためのデータを構築する障壁となる可能性がある．

単語依存構造は，アノテーションを行う観点からすると同じ依存構造である文節依存構造に近く，句構造木の構築に比較して低いアノテーションコストで様々なドメインのデータが構築しやすいと考えられる．日本語の単語単位の依存構造解析については，森らが，現代日本語書き言葉均衡コーパス [20, 21] で定義されている短単位をベースとして[14]，大規模なコーパスを構築している [19] ．彼らのコーパスでは単語間の文法機能タイプのアノテーションはなく，文法機能情報を獲得するためには別のデータを準備することが必要になる．他に，内元らは，日本語の話し言葉を対象として，主に言い淀みやフィラー等の文節依存構造ではうまく表現が行えない現象に対して，単語間の依存構造を定義しているが [38]，詳細な文法機能を区別するための仕組みは考慮されていない．

文法機能タイプを持つ単語依存構造で構文構造を表現する方法は，英語等では広く使われており，中でも Stanford typed dependencies （以下，SD）[39] が代表的なアノテーションスキーマとなっている．日本語の文法機能タイプは，標準的な文節依存構造では扱われていないため，本研究では SD で用いられている文法機能タイプを参考にして新たに定めた．ただし，述語項構造情報を抽出するために必要になる情報など日本語の構文構造を表現するのに不足している情報があるため，日本語の文法的特徴が付与されている句構造木の非終端記号を変換するという方針により，必要な情報を表現できるように拡張した[15]．句構造は各部分木が構文の構成素と一致することからも，構造の変換元とすることに適している．本論文では，提案する単語依存構造に基づく最初のコーパスの構築を，既存コーパスの句構造木に付与された情報を単語依存構造に変換することで行い，変換された構文木を学習データとすることで解析器を実現した．

2.2.2 節で述べた多言語横断で共通の構文構造表示を目指すスキーマである UD は，SD

[14]動詞，形容詞等用言は，短単位を語基と活用語尾でさらに分割している．

[15]例えば,「内の関係」と「外の関係」の連体修飾節の区別を表現できるように文法機能タイプを細分化した．

を拡張したものであり日本語 UD の仕様も策定されコーパスの構築が行われている [2, 3, 7]. 前述したように UD では，言語特有の文法機能情報は UD の仕様の範囲外に位置づけられアノテーションされない傾向があるが，本研究の単語依存構造は日本語の文法機能情報を保持するように設計されている．また，本研究の単語依存構造は UD に変換可能なように UD に必要な情報を包含するように考慮している．例えば，文法機能タイプに関しては，2.3.3 節に述べたように UD をベースとして，より細かいタイプを採用しているため，タイプを統合することにより UD への対応付けが可能である．

2.6 構文解析器への適用評価

本論文で提案する単語依存構造は，依存関係を結ぶ組合せの数が，文節依存構造と比較して増加するだけでなく，従来の単一方向の依存構造からなる文節依存構造と比較して複雑な構造となっているため，構文解析器に適用した場合にどの程度の解析精度が得られるかを確認する必要がある．また，依存構造スキーマによっては，統語的に重要な関係を持つ語の間の距離（ノード間の弧の数）が長くなることによって，高次の特徴量を使用しないと十分な精度が得られない懸念がある．

本節では，提案する単語依存構造による構文解析モデルを実際の解析器で構築し，これらの要素がどの程度解析精度に影響するのかを評価する．評価の主眼は，それぞれの依存構造スキーマに基づく単語依存構造に対して解析モデルを最適化することではなく，既存の解析器の標準的な設定で構築したモデルを使用する際に，依存構造スキーマ，依存構造単位（短単位，長単位，文節）の違いが解析精度へどう影響するかを確認することである．実験には，単語依存構造解析で実績のある 2 種類の transition ベースの構文解析器を使用することで，標準的な条件での解析精度が得られることを想定している．また，本論文の依存構造解析から得られた述語項構造情報を，既存の述語項構造解析により得られた述語項構造情報と比較することにより，導入した文法機能タイプの効果を確認する．

各依存構造スキーマに基づいた依存構造コーパスを学習データとして構文解析モデルを構築し，解析精度について評価した．解析評価には，前節で述べたように句構造ツリーバンクから自動的に変換した依存構造コーパスと，既存の構文解析器を使用した．評価する観点として，次の 3 点に着目した．

(1) 依存構造スキーマの解析精度への影響

(2) 依存構造単位の解析精度への影響

(3) 文法機能タイプから得られる述語項構造情報の評価

(1) では，2.3.2 節 で述べた 6 種類の依存構造スキーマに基づいて構築した長単位による依存構造解析モデルの解析精度を比較した．長単位列の正解データを入力として使用

表 2.6: 実験条件.

評価項目	入力	解析方法	解析の前工程
依存構造スキーマ	長単位列	単語依存構造解析:LUW	–
依存構造単位	短単位列	単語依存構造解析:LUW	長単位チャンキング
		単語依存構造解析:SUW	–
		文節依存構造解析	文節チャンキング
述語項構造情報	平文	単語依存構造解析:LUW	形態素解析+長単位チャンキング
		述語構造解析	形態素解析+文節チャンキング +文節依存構造解析

し，形態素解析（短単位へ分割と品詞タグ付与）や長単位へのチャンキングの精度の影響を排除した．(2) では，短単位列の正解を入力として，依存構造の単位（短単位，長単位，文節）が解析精度に与える影響を調べた．評価のベースラインを揃えるため，短単位列から長単位や文節のチャンキングを行う精度の影響も精度評価に含めた．すなわち比較するのは，(a) 短単位列から直接短単位間の依存構造解析を行う場合，(b) 短単位列から長単位へのチャンキングを行った後，長単位間の依存構造解析を行う場合，(c) 短単位列から文節へのチャンキングを行った後，文節間の依存構造解析を行う場合，の 3 種類である．最後に (3) では，依存構造解析結果から文法機能タイプを利用して抽出した述語項構造解析の情報と，既存の述語項構造解析器の結果との比較を行った．比較する解析器の条件を揃えるため，平文を入力とし，依存構造解析を行う場合は，短単位への形態素解析，長単位へのチャンキング，長単位間の依存構造解析を順に行った．表 2.6 にこれらの実験条件をまとめた．表中で，解析方法の LUW は長単位ベースの解析，SUW は短単位ベースの解析を表す．

以下，2.6.1 節で共通の実験設定について述べた後，(1)-(3) に対応する評価実験についてそれぞれ 2.6.2 節- 2.6.4 節で述べ，最後に 2.6.5 節で評価結果をまとめる．

2.6.1 実験設定

評価実験には，句構造ツリーバンクから 2.4 節で述べた方法で依存構造に変換したものを使用した．KyotoDep は，全 19,953 文を学習データ 15,839 文，開発データ 2,114 文，評価データ 2,000 文に，BCCWJDep は，全 18,400 文を学習データ 14,772 文，開発データ 1,919 文，評価データ 1,798 文に分割し，それぞれのコーパスごとに構文解析モデルを構築した．表 2.7 に学習，開発，評価データの統計量を示す．

解析器の前処理が必要な場合，短単位への分割および品詞タグ付けは，MeCab [40] ，

表 2.7: 評価実験に使用したコーパスの統計量.

コーパス		文数	長単位数	短単位数
KyotoDep	学習データ	15,839	335,895	434,383
	開発データ	2,114	47,901	62,961
	評価データ	2,000	41,182	53,192
BCCWJDep	学習データ	14,772	285,217	356,242
	開発データ	1,919	34,739	46,192
	評価データ	1,798	36,097	43,220

長単位へのチャンキングは，Comainu [41] を使用して行った[16].

構文解析には，タイプ付き単語依存構造解析で実績のある transition ベースの構文解析器として，英語の SD において 80%以上のラベル正解率の実績を持つ MaltParser [42] および，主に UD の解析に使用されている UDPipe [43] の 2 種類の構文解析器を用いた[17]. MaltParser は，解析アルゴリズムに Stack アルゴリズム (projective) [44] を選択し，識別学習ライブラリに LIBLINEAR[18]を使用してモデル学習を行った．Stack アルゴリズム (projective) は transition ベースの構文解析の標準的なアルゴリズムで，スタックの最上位と 2 番目のノードの間に依存関係を順に構成することで構文解析を行う．6 種類の依存構造スキーマそれぞれに対して解析モデルを構築して評価を行った．解析モデルの素性には，表 2.8 に示す単語属性を組み合わせて用いた．長単位は複合名詞等の構成的な複合語を多く含み，長単位ベースの属性で構成した素性は疎になる傾向があるため，短単位ベースの属性を組み合わせた．短単位ベースの単語属性は，対象となる長単位の最左あるいは最右の短単位のものを用いた．例えば，長単位「魚フライ」に対して，短単位ベースの単語属性は，$S_L.l$ （最左の短単位の語彙素「魚」），$S_R.l$ （最右の短単位の語彙素「フライ」）等である．また，単語を汎化するため，日本語語彙大系 [45] の属性を用いて単語意味属性（一般名詞属性），固有名詞属性を付与した．単語意味属性は，過度の細分化を防ぐため 3 階層目までの属性を使用した．

用いた素性テンプレートを表 2.9 に示す．依存構造スキーマによって有効な素性は異なると考えられるが，各依存構造スキーマごとに表 2.9 の左側に記載しているようなカテゴリを当てはめて素性テンプレートを定義し，評価実験時には全依存構造スキーマの素性テンプレートの和集合を共通に使用した．

MaltParser は transition ベースの解析器であり，解析は先頭から順に決定的に行われる．素性テンプレートは，ある解析時点でのスタック S 中の部分木，キュー Q 中の単語

[16]MeCab は ver. 0.996 を UniDic 辞書 mecab-unidic ver. 2.1.2 とともに用い，Comainu は ver. 0.72 を用いた．

[17]MaltParser は ver. 1.8， UDPipe は ver. 1.2.0 を用いた．

[18]ver. 2.1 を用いた．

表 2.8: 構文解析器 (MaltParser) の素性テンプレートに用いた単語属性.

長単位 (LUW)		短単位 (SUW)	
$L.f$	書字形	$S_R.f$	書字形 (最右短単位)
$L.l$	語彙素	$S_L.f$	書字形 (最左短単位)
$L.p$	品詞	$S_R.l$	語彙素 (最右短単位)
$L.i_t$	活用型	$S_L.l$	語彙素 (最左短単位)
$L.i_f$	活用形	$S_R.p$	品詞 (最右短単位)
$L.t$	前終端記号	$S_R.t$	前終端記号 (最右短単位)
$L.c_g$	単語意味属性	$S_R.c_g$	単語意味属性 (最右短単位)
$L.c_p$	固有名詞属性	$S_R.c_p$	固有名詞属性 (最右短単位)
$L.d$	主辞との文法機能タイプ		
	(スタックの部分木中でのみ有効)		

に関する単語属性の組合せで定義している[19]．表中，スタック S 中の部分木の根の単語を最上部から，$\{s_0, s_1, \ldots s_n\}$，キュー Q 中の単語を先頭から，$\{q_0, q_1, \ldots q_m\}$ で表している．Stack アルゴリズムでは，スタックの最上位から 2 つの部分木 (根がそれぞれ s_0, s_1) から新しい弧を作る．スタック中の部分木については，$.h$（主辞），$.l_c$, $.l_d$ 等の指定子で対象の単語（長単位）を指定している．$.l_c$，$.r_c$ はそれぞれ部分木の根（主辞）の最左の子（従属部），最右の子，$.l_d$，$.r_d$ はそれぞれ部分木の中の根の最左の子孫，最右の子孫を表している．

UDPipe は，MaltParser と同じく transition ベースの解析器であるが，ニューラルネットワークベースの分類器を用いた予測を行い，素性を設定する必要はない．transition system は，既定値である projective な arc standard system を用いた．また，解析器自身で，単語分割，品詞タグ付けを行うことができるが，他と条件を揃えるため本論文では，UDPipe は依存構造解析のみを行い，短単位への分割，品詞タグ付けは，前述の MeCab を用いた．

2.6.2 依存構造スキーマの比較

長単位列の正解データを入力として与え，6 種類の依存構造スキーマに基づく解析モデルにより構文解析を行った．全体的な解析精度の評価には，依存構造解析で標準的に使用される Labeled Attachment Score (LAS) と Unlabeled Attachment Score (UAS) を用いた．LAS, UAS はそれぞれ，式 (2.1)，式 (2.2) で計算される．

$$\text{LAS} = \frac{\text{主辞 (係り元) および依存構造ラベルが正しい単語数}}{\text{解析結果中の全単語数}} \tag{2.1}$$

[19]表では単語属性の組合せの一例を示している．

表 2.9: 構文解析器 (MaltParser) に用いた素性テンプレート.

カテゴリ（想定する依存構造）	素性
単語列	
n-gram	$s_0.h.L.f \circ q_0.L.f$
	$s_0.h.L.f \circ q_0.L.f \circ q_1.L.f$
	$s_0.h.L.f \circ q_0.L.f \circ q_1.L.f \circ q_2.L.f$
主辞の列	$s_1.h.L.f \circ s_0.h.L.f$
	$s_2.h.L.f \circ s_1.h.L.f \circ s_0.h.L.f$
従属部の列	$s_1.l_c.L.f \circ s_0.l_c.L.f$
	$s_2.l_c.L.f \circ s_1.l_c.L.f \circ s_1.l_c.L.f$
述語項構造	
項従属部-述部主辞	$s_1.l_c.L.f \circ s_0.h.L.f$
	$s_1.l_c.S_R.f \circ s_0.h.S_R.f$
	$s_1.l_c.S_R.l \circ s_0.h.S_R.l$
	$s_2.l_c.S_R.l \circ s_1.l_c.S_R.l \circ s_0.h.S_R.l$
	$s_1.l_c.S_R.c_g \circ s_0.h.S_R.c_g$
	$s_2.l_c.S_R.c_g \circ s_1.l_c.S_R.c_g \circ s_0.h.S_R.c_g$
項主辞-述語主辞	$s_1.h.L.f \circ s_0.h.L.f$
	$s_1.h.S_R.f \circ s_0.h.S_R.f$
	$s_1.h.S_R.l \circ s_0.h.S_R.l$
	$s_1.h.S_R.c_g \circ s_0.h.S_R.c_g$
	$s_2.h.L.f \circ s_0.h.L.f$
	$s_2.h.S_R.f \circ s_0.h.S_R.f$
	$s_2.h.S_R.l \circ s_0.h.S_R.l$
	$s_2.h.S_R.c_g \circ s_0.h.S_R.c_g$
	$s_2.h.L.f \circ s_1.h.L.f \circ s_0.h.L.f$
	$s_2.h.S_R.f \circ s_1.S_R.L.f \circ s_0.h.S_R.f$
	$s_2.h.S_R.l \circ s_1.S_R.L.l \circ s_0.h.S_R.l$
	$s_2.h.S_R.c_g \circ s_1.S_R.L.c_g \circ s_0.h.S_R.c_g$
述部-名詞句主辞（連体修飾）	$s_1.h.S_R.f \circ s_0.h.S_R.f$
	$s_1.h.S_R.f \circ s_0.h.S_R.c_g$
	$s_1.l_c.S_R.f \circ s_1.h.S_R.f \circ s_0.h.S_R.f$
	$s_1.l_c.S_R.f \circ s_1.h.S_R.f \circ s_0.h.S_R.c_g$
述部-述部主辞（連用修飾）	$s_1.h.S_R.f \circ s_1.r_c.S_R.f \circ s_0.h.S_R.c_g$
	$s_1.h.S_R.c_g \circ s_1.r_c.S_R.f \circ s_0.h.S_R.f$
	$s_1.h.S_R.f \circ s_1.r_d.S_R.f \circ s_0.h.S_R.f$
	$s_1.h.S_R.c_g \circ s_1.r_d.S_R.f \circ s_0.h.S_R.f$
その他	
部分木の根の間の距離	$\mathrm{dist}(s_3.h.L.f, s_0.h.L.f)$
	$\mathrm{dist}(s_2.h.L.f, s_0.h.L.f)$
	$\mathrm{dist}(s_1.h.L.f, s_0.h.L.f)$

表 2.10: 全体解析精度（長単位）.

MaltParser

	KyotoDep		BCCWJDep	
Schema	UAS	LAS	UAS	LAS
HF_1	94.68	91.02	93.12	88.79
HF_2	**94.85**	**91.19**	**93.69**	**89.45**
PCH	94.44	91.13	93.19	89.29
ACH_1	92.65	89.13	91.12	86.73
ACH_2	93.27	89.55	91.84	87.35
CWH	92.83	89.46	91.84	87.66

UDPipe

	KyotoDep		BCCWJDep	
Schema	UAS	LAS	UAS	LAS
HF_1	**94.96**	90.90	93.46	88.84
HF_2	**94.96**	**91.01**	**93.94**	**89.63**
PCH	94.51	90.83	93.42	89.37
ACH_1	93.36	89.68	92.25	86.06
ACH_2	93.67	89.85	92.78	88.44
CWH	93.10	89.71	92.66	88.67

$$\text{UAS} = \frac{\text{主辞 (係り元) が正しい単語数}}{\text{解析結果中の全単語数}} \tag{2.2}$$

LAS は，解析結果の全単語のうち，依存構造の矢印の元の単語（主辞）と文法機能タイプ（文法機能ラベル）がともに正解となっている単語の割合を表している．UAS は，主辞が合っていれば文法機能タイプが誤っていても正解として扱った場合の単語の割合である．また，文法機能タイプ l 関する解析精度を評価するために，式 (2.5) で表される F1 スコア (F_l) を用いた．

$$P_l = \frac{\text{分母の内正解の依存構造}}{\text{解析結果の内ラベル } l \text{ を持つ依存構造}} \tag{2.3}$$

$$R_l = \frac{\text{分母の内解析結果に含まれる依存構造の数}}{\text{正解の内ラベル } l \text{ を持つ依存構造}} \tag{2.4}$$

$$F_l = 2P_lR_l/(P_l + R_l) \tag{2.5}$$

文法機能タイプごとの評価については，解析器の出力した文法機能タイプ l が正しい割合である適合率 P_l と正解データ中の文法機能タイプ l 対して解析器が出力できた割合である再現率 R_l の 2 つの観点が存在するため，両者の調和平均である F1 スコア F_l を採用している．

全体の精度として，UAS および LAS を表 2.10 に，文法機能タイプごとの F1 スコアを表 2.11，表 2.12 に示す．全体的な傾向として，後置詞句において助詞を主辞とするスキーマ (HF_1, HF_2, PCH) の精度が，内容語を主辞とするスキーマ (ACH_1, ACH_2, CWH) よりも高い結果となっている．特に，後置詞句，述語句ともに機能語を主辞とし，述語句を先に結合する HF_2 がいずれの場合も最も良い精度となっていることは，日本語の単語依

表 2.11: 文法機能タイプ別解析結果（長単位，KyotoDep）.

MaltParser

	F1 スコア					
文法機能タイプ	HF_1	HF_2	PCH	ACH_1	ACH_2	CWH
nsubj	83.37	**85.09**	84.76	77.76	79.50	78.31
dobj	**93.50**	91.54	93.46	90.56	90.25	92.13
iobj	**84.49**	82.55	84.22	80.06	77.47	80.13
tmod	54.35	52.65	52.52	54.51	**54.96**	53.81
lmod	48.78	**50.00**	49.79	45.27	44.90	49.17
rcmod_nsubj	67.46	69.29	**73.31**	66.54	69.82	73.19
rcmod_dobj	40.00	39.32	47.37	28.83	41.67	**50.00**
rcmod_iobj	42.11	41.03	**51.22**	42.67	38.46	50.63
ncmod	85.43	85.12	84.02	85.36	85.42	**85.93**
advcl	72.19	**72.43**	70.35	70.92	70.83	66.01
conj	72.98	**73.35**	71.41	65.35	68.01	66.30
appos	61.39	**63.92**	62.00	52.27	53.33	50.00

UDPipe

	F1 スコア					
文法機能タイプ	HF_1	HF_2	PCH	ACH_1	ACH_2	CWH
nsubj	84.64	**85.34**	84.32	81.99	83.48	82.85
dobj	93.17	92.02	**93.70**	90.94	90.64	92.59
iobj	83.05	82.47	**83.06**	81.05	80.31	81.94
tmod	54.15	54.58	**54.77**	51.37	50.18	53.45
lmod	43.40	**44.19**	38.71	41.41	42.98	42.68
rcmod_nsubj	66.41	67.92	67.66	68.09	64.39	**68.34**
rcmod_dobj	**46.67**	37.70	41.27	29.27	30.40	36.80
rcmod_iobj	**45.7**1	37.33	34.78	34.78	34.38	34.92
ncmod	**83.94**	83.63	80.72	84.04	82.14	82.93
advcl	70.74	**71.61**	69.65	69.43	71.05	67.65
conj	74.15	74.52	73.71	71.39	74.20	**75.97**
appos	**64.08**	62.96	67.31	57.14	58.00	55.45

表 2.12: 文法機能タイプ別解析結果（長単位，BCCWJDep）.

MaltParser

	F1 スコア					
文法機能タイプ	HF_1	HF_2	PCH	ACH_1	ACH_2	CWH
nsubj	76.93	**80.79**	78.70	68.53	73.44	71.66
dobj	**91.42**	91.21	90.35	89.81	88.87	89.71
iobj	**79.11**	76.06	76.93	73.93	73.27	75.15
tmod	28.46	33.73	29.17	**37.45**	34.72	36.63
lmod	**38.65**	35.53	37.62	37.23	32.43	37.56
rcmod_nsubj	58.00	57.82	61.12	58.63	61.34	**61.47**
rcmod_dobj	24.53	29.17	**42.99**	21.05	32.99	40.00
rcmod_iobj	42.25	37.84	**50.63**	30.30	28.99	36.63
ncmod	86.89	87.10	**87.38**	85.76	86.66	86.56
advcl	66.12	**70.51**	64.91	64.55	67.08	63.07
conj	68.45	68.51	**68.79**	60.32	64.84	63.24
appos	16.67	**20.00**	16.67	12.35	15.79	18.67

UDPipe

	F1 スコア					
文法機能タイプ	HF_1	HF_2	PCH	ACH_1	ACH_2	CWH
nsubj	78.62	**81.43**	79.89	75.91	78.21	78.74
dobj	90.48	**90.91**	90.00	86.46	89.77	89.85
iobj	76.21	75.74	**76.40**	75.29	74.90	76.06
tmod	27.38	30.96	32.43	**33.61**	31.08	26.55
lmod	23.64	**29.44**	29.13	22.83	28.57	26.18
rcmod_nsubj	57.30	**61.27**	57.63	58.04	55.46	59.31
rcmod_dobj	23.08	29.17	20.45	**32.56**	15.56	29.41
rcmod_iobj	**39.44**	33.33	30.56	30.00	26.83	20.59
ncmod	85.47	85.38	**86.16**	85.56	85.38	84.78
advcl	67.45	**69.77**	66.85	66.90	68.98	66.37
conj	65.79	66.29	65.48	65.31	**66.72**	66.13
appos	26.36	25.64	**36.07**	26.15	20.80	24.79

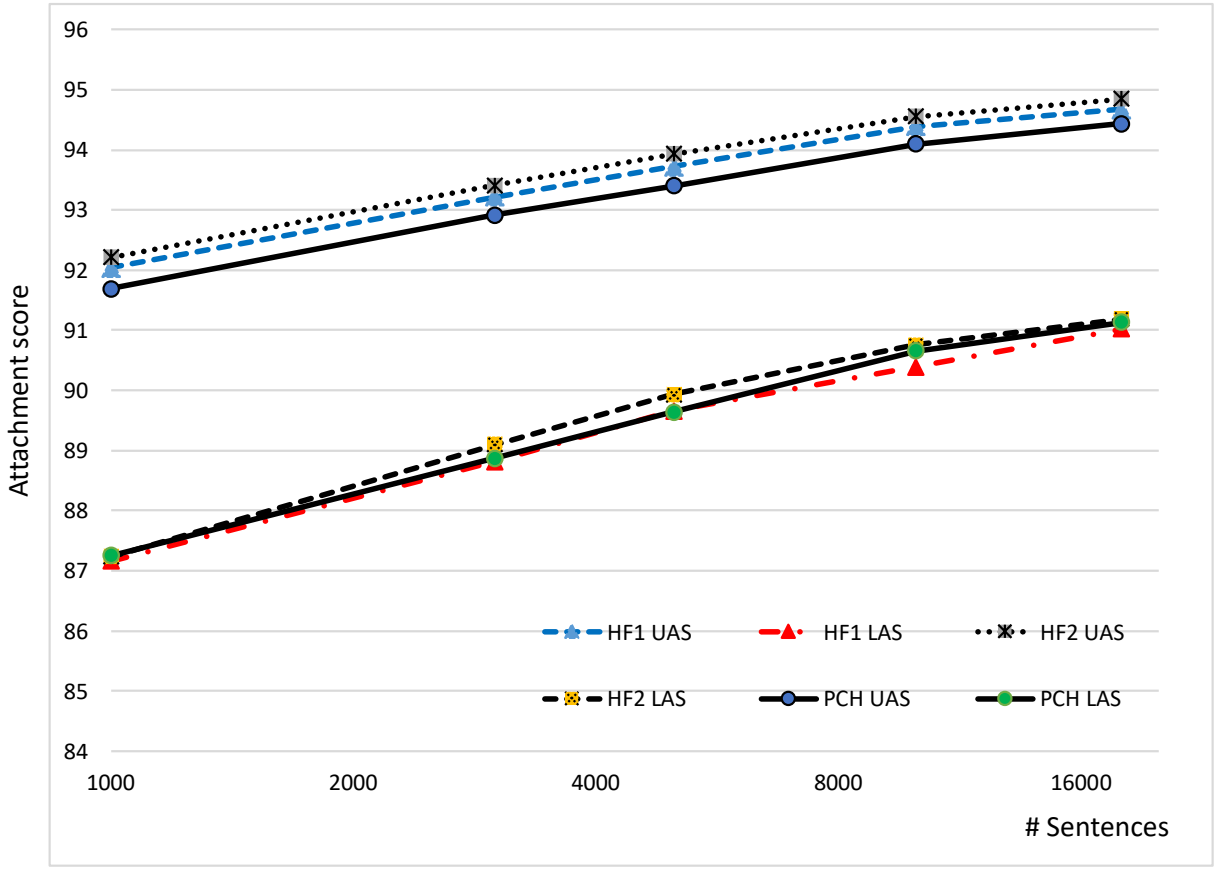


図 2.11: 学習コーパス量と解析精度の関係 (KyotoDep).

存構造解析を標準的な transition ベースの構文解析器への適用するにあたって，機能語主辞の構造が精度面で有利であることを示唆していると考えられる．

文法機能タイプ別の精度を見ると，コーパスで大きな割合を占める格関係の依存構造 (*nsubj* など) や連用修飾関係 (*advcl*)，並列構造 (*conj* など) において，HF_1, HF_2, PCH のスキーマによる精度が高い． これらの依存構造において，主辞を内容語とするよりも格助詞等の機能語とする方が有利であることを反映しており，3 つのスキーマの全体精度が高いことに寄与していると考えられる．

一方，連体修飾関係 (*rcmod*，*ncmod* など) の精度は，述語句で内容語を主辞とするスキーマ (PCH, CWH) が比較的高い精度を示していることがわかる．これは，連体修飾句の主辞となる用言と主名詞の内容語との間で直接依存関係を持つことが，依存関係先，および文法機能タイプの決定に寄与している結果と考えられる．

必須項の格関係 (*nsubj*, *dobj*，*iobj*) の精度は，付加項の格関係 (*tmod*, *lmod*) に比べて高い結果となっている．必須項の格関係については，典型的には名詞句に後置される格助詞「が」「を」「に」と動詞の組合せによって正しく推測可能な場合が多くの割合を占めるのに対して，時間格や場所格は，共起しやすい格助詞「に」「で」がそれ以外の広範な用法を持っており，時間格，場所格で用いられているのか他の用法で用いられているかの判別

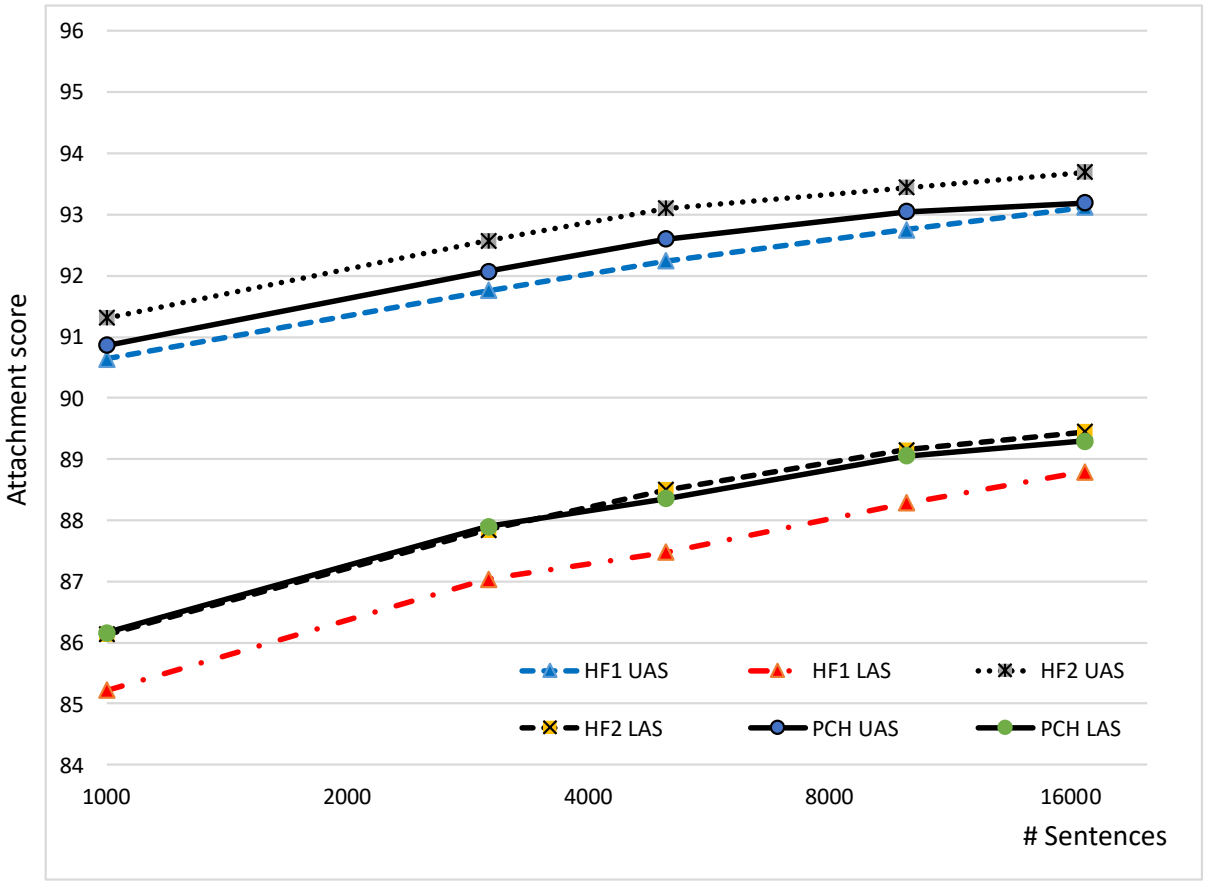


図 2.12: 学習コーパス量と解析精度の関係 (BCCWJDep).

が困難であることが多いためと考えられる．例えば，同じ「飛行機で」という句の場合，「飛行機で沖縄に行く」という文では,「行く」の動作の手段を表しているのに対し,「飛行機で昼間に眠る」では,「眠る」という動作の行われている場所を示しているが，このような違いを判別することは容易ではない．

学習コーパス量

学習コーパスの規模と解析精度との関係を見るため，KyotoDep，BCCWJDep それぞれについて，MaltParser の学習データとして使う文数を，1,000 文, 3000 文, 5,000 文, 10,000 文，全文に変えて解析モデル (HF_1, HF_2, PCH) を構築し，解析精度の変化を調べた結果を，図 2.11, 図 2.12 に示す．UAS は依存構造の形を同定する精度，LAS は文法機能タイプ（依存構造ラベル）の同定を含めた精度である．どのモデルも本論文の評価実験で用いた 1,000 文規模で UAS 91 以上の精度を実現し，10,000 文規模で UAS 93-94 に達しており，それ以上の文数では上昇幅がやや鈍化している．また，1,000-10,000 文の区間では，LAS の精度の上昇幅が大きくこの区間で文法機能タイプの学習が効果的に行われていることが分かる．すなわち，依存構造の形のみの精度 (UAS) は 1,000 文程度の学習データ

があればよいが，文法機能タイプの同定を精度よく行うためには，10,000 文規模の学習データを用いるのが一つ目安と考えられる．

2.6.3 依存構造単位の比較

依存構造の単位による解析精度の違いを調べるため，KyotoDep のデータに関して，長単位による依存構造の他に，短単位の依存構造，文節単位の依存構造について依存構造解析モデルを構築した．正解の短単位依存構造は，長単位依存構造のデータを元に各長単位を短単位に分割し，長単位内の隣接する短単位間に依存関係を結び，右側の短単位が左側の短単位の主辞になる構造とした．長単位間の依存構造は，依存関係元，依存関係先，それぞれの長単位の中で，一番右側に来る短単位，すなわち主辞になる短単位の間の依存構造に変換した．また，長単位内部の短単位間の文法機能タイプは *luw* に統一した．図 2.13 に，長単位依存構造から短単位依存構造へ変換した例を示す．

長単位と短単位

長単位の依存構造解析モデルと短単位の依存構造解析モデルを比較するため，短単位列の正解データを入力として，長単位依存構造解析モデル，短単位依存構造解析モデルそれぞれで解析した．長単位の依存構造解析は，入力の短単位列を Comainu でチャンキングした結果の長単位列を，長単位依存構造解析モデルで解析した．長単位の正解データを直接入力しないのは，短単位依存構造解析モデルと入力の条件を揃えることと，長単位チャンキングの全体解析精度への影響を加味するためである．

表 2.13 は，KyotoDep の短単位の形態素情報の正解データを入力として，長単位依存構造モデルで解析した結果と，短単位依存構造モデルで解析した結果である．比較のために，長単位解析モデルの結果も文法機能タイプ *luw* を用いて短単位の依存構造に変換して集計している．短単位の依存関係で集計すると，比較的簡単な長単位内部の依存関係 *luw* が含まれることにより，精度が高めに計測されるため，括弧内に依存関係ラベル *luw* を除いた精度を表示している．

表 2.13 の MaltParser の結果において，PCH, CWH を除いて短単位モデルの解析精度が長単位モデルより僅かに上回っているのは，長単位チャンキングの精度（長単位境界の F1 スコア 99.41，品詞の F1 スコア 98.80）が影響していると考えられる．一方，UDPipe では，全体に短単位モデルの結果よりも長単位モデルの結果の方が高く，長単位で依存構造を捉えることが有効に働いている．同じスキーマで比較する限り，UDPipe の長単位モデルでの解析精度は，MaltParser の長単位および短単位モデル，UDPipe の短単位モデルの結果をほぼ上回っていることから，ある程度のチャンキングの精度と，依存構造解析器の基本精度があれば，長単位モデルで解析する方が高い精度が得られると考えられる．

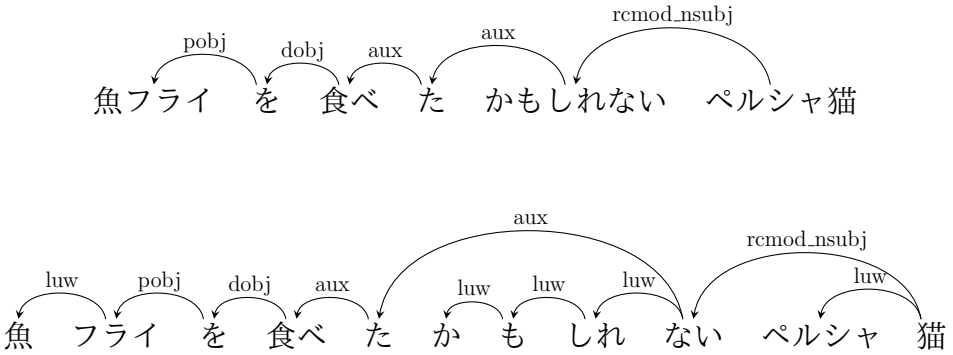


図 2.13: 長単位依存構造から短単位依存構造への変換の例 (PCH).

長単位・短単位と文節

文節単位の依存構造との比較を行うため，KyotoDep の 19,953 文について単語単位を短単位に置換したデータを人手で作成した．この形態素情報データのうち 15,893 文を CaboCha の学習データとして，文節チャンキングモデル，文節依存構造モデルをそれぞれ構築した．評価データを CaboCha で解析した結果は，CoNLL 形式による短単位間の依存構造として出力した [20].

条件を揃えるため，いずれの解析も短単位列データを入力として与え，短単位解析は短単位の依存構造解析，長単位解析は Comainu による長単位チャンキングと長単位の依存構造解析，文節解析は CaboCha による文節チャンキングと文節依存構造解析を行った．長単位・短単位依存構造と文節依存構造は，依存構造の単位が異なるため単純な精度の比較はできないが，構文解析から獲得できる重要な情報として，「格関係」「連体修飾関係」「連用修飾関係」「並列関係」の 4 つの文法機能タイプのカテゴリについて，正解の単語間の依存構造を出力できる割合（再現率）を調べた．

文法機能タイプのカテゴリ分けは，格関係は， *nsubj*, *dobj*, *iobj*, *tmod*, *lmod*, *arg*，連体修飾関係は，*rcmod*, *rcmod_nsubj*, *rcmod_dobj*, *rcmod_iobj*, *ncmod*，連用修飾関係は，*advcl*，並列関係は，*conj*, *appos* とした．CaboCha は文法機能タイプの出力を行わないので，いすれの解析結果についても文法機能タイプの正誤は無視した．結果を表 2.14 に示す．

スキーマにより違いはあるが，HF_1, HF_2, PCH については，ほぼ文節依存構造と同程度の精度を示している．特に連体修飾関係と並列関係は，文節依存構造よりも高い精度を示す結果となっている．一方，連用修飾関係は，文節依存構造が単語依存構造よりも高い精度を示している．依存関係にある単語間の距離による傾向を見るため，MaltParser の

[20]CaboCha は ver.0.68 を用いた．実際の解析は文節依存構造として行われるが，CaboCha の出力オプションで CoNLL 形式を指定することにより，文節内は隣接する左の単語から右の単語へ順に係り（右の単語が左の単語の主辞となり），文節間はそれぞれの文節の主辞が依存関係元と依存関係先の単語として出力される．

表 2.13: 構文解析結果:依存構造単位の比較（KyotoDep）.

MaltParser

	長単位		短単位	
Schema	UAS	LAS	UAS	LAS
HF_1	95.00 (93.70)	91.12 (88.84)	94.79 (93.42)	91.47 (89.32)
HF_2	**95.14 (93.91)**	**91.36 (89.16)**	**95.36 (94.20)**	**92.06 (90.08)**
PCH	94.38 (92.97)	91.25 (89.01)	94.32 (91.47)	91.40 (87.82)
ACH_1	92.78 (90.05)	89.15 (86.30)	93.18 (91.40)	90.00 (87.47)
ACH_2	93.45 (91.74)	89.01 (87.15)	93.90 (92.36)	90.59 (88.20)
CWH	93.13 (91.38)	89.92 (87.29)	90.55 (88.71)	87.12 (84.44)

UDPipe

	長単位		短単位	
Schema	UAS	LAS	UAS	LAS
HF_1	95.49 (94.32)	91.71 (89.60)	95.03 (93.69)	91.18 (89.09)
HF_2	**95.65 (94.56)**	**91.92 (89.88)**	**95.20 (94.07)**	**91.38 (89.40)**
PCH	94.77 (93.45)	91.59 (89.44)	94.04 (92.61)	91.00 (88.91)
ACH_1	93.81 (92.17)	90.62 (88.20)	93.25 (91.57)	89.92 (87.54)
ACH_2	94.04 (92.49)	90.76 (88.38)	93.89 (92.39)	90.66 (88.46)
CWH	93.83 (92.28)	90.78 (88.41)	93.48 (91.96)	90.42 (88.16)

HF_2（長単位）の結果と CaboCha の結果について，連体修飾と連用修飾の依存関係にある単語間の距離（短単位に換算）と再現率の関係を図 3.12 に示す．連体修飾関係の場合，単語依存構造の方が全般に高い精度を示している．連体修飾関係の依存構造の多くは 5 単位程度の短い距離であり，6-15 短単位の中程度の距離の事例は少ないため，推定するのが難しい依存関係と考えられる．単語依存構造は，連体修飾関係のこの中程度の距離において，再現率の低下を低く抑えられている．一方，連用修飾関係の場合，後置詞句と述語や，副詞節と主節など，絶対的な距離が長くなり得る関係が含まれるが，15 短単位程度までの距離では単語依存構造，文節依存構造で差がないのに対し，15 短単位を超える距離においては文節依存構造の方が精度を維持していることが分かる．これは，文節にチャンキングすることにより，依存構造を持つ単位（文節）の間の実質的な距離を縮めていることと，依存構造を構成する組合せの数を減らしてることが有利に働いていると考えられる．

表 2.14: 構文解析結果（文法機能タイプ種別再現率）.

解析器	依存構造スキーマ		格関係	連体修飾	連用修飾	並列
MaltParser	長単位	HF_1	86.59	**90.03**	77.02	71.43
		HF_2	87.35	**90.03**	78.24	**72.08**
		PCH	87.27	88.97	76.35	70.51
		ACH_1	81.99	89.80	73.21	58.41
		ACH_2	84.47	89.53	74.68	62.18
		CWH	84.30	89.59	74.22	60.34
	短単位	HF_1	86.27	89.11	73.56	70.02
		HF_2	**88.32**	88.29	78.98	69.91
		PCH	86.95	87.41	74.15	68.35
		ACH_1	82.82	89.10	71.54	55.06
		ACH_2	85.52	88.56	76.22	58.73
		CWH	84.63	87.26	71.14	58.08
UDPipe	長単位	HF_1	88.01	89.59	77.49	74.89
		HF_2	**88.95**	**89.97**	78.37	**75.22**
		PCH	88.14	87.30	75.89	73.95
		ACH_1	84.56	89.32	73.48	70.47
		ACH_2	85.94	88.39	74.08	73.06
		CWH	86.72	85.53	73.81	73.28
	短単位	HF_1	87.38	88.78	75.93	72.29
		HF_2	87.76	88.78	78.44	71.65
		PCH	87.45	85.74	73.48	72.23
		ACH_1	84.04	87.74	69.14	69.83
		ACH_2	86.17	88.39	73.55	70.37
		CWH	86.13	85.80	72.61	73.17
CaboCha	文節		88.28	88.35	**80.49**	70.83

2.6.4 述語項構造情報

2.1 節 で述べたように，述語項構造は構文構造と密接な関係があるため，構文構造に有効な情報を持たせれば，述語と項を直接抽出することが可能である．本節では，文法機能タイプ付き依存構造解析から抽出可能な述語項構造の情報の精度について評価を行う．述語項構造を抽出するために直接目印となる文法機能タイプは，必須項の格関係 (*nsubj*, *dobj*, *iobj*) と内の関係の関係節 (*rcmod_subj*, *rcmod_dobj*, *rcmod_iobj*) である．依存構造解析結果から，述語を起点としてこれらの文法機能タイプを辿ることによって，述語，格

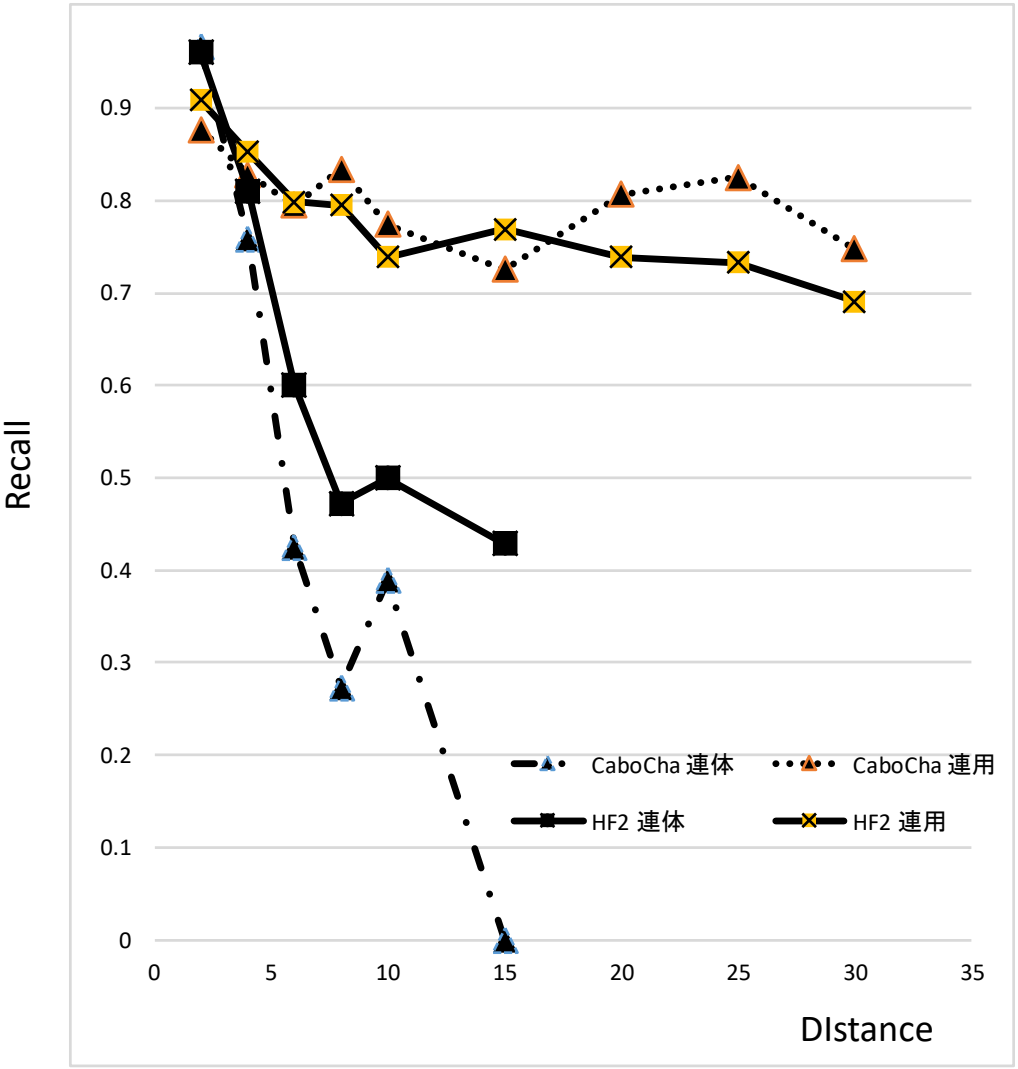


図 2.14: 依存関係にある単語間の距離と再現率.

関係，項 の 3 つ組 $(pred, rel, arg)$ を抽出する．述語 $pred$ は動詞（サ変名詞を含む）か形容詞（形状詞を含む），項 arg は項の主辞の名詞，格関係 rel は主格 $(nsubj)$, 対格 $(dobj)$, 与格 $(iobj)$ である．

正解データは，本論文で用いた KyotoDep のデータのうち，NAIST テキストコーパス ver.1.5 （以下，NAIST コーパス）[46] に収録されている述語項構造をこの 3 つ組に変換することによって作成した．基本的には，NAIST コーパス中の格関係 “ga”, “o” および “ni” をそれぞれ $nsubj$, $dobj$, $iobj$ に対応させた．ただし，能動態，使役態等の格交替が起きている場合でも，NAIST コーパス中の格関係は能動態のものに直して表示されているため，コーパス中の態での格関係になるように人手で修正した．また，抽出された 3 つ組のうち，ゼロ代名詞が含まれているものは除外し，コーパス中の依存構造で正解が抽出で

表 2.15: 述語項構造情報獲得結果（長単位）.

解析器	依存構造スキーマ	適合率	再現率	F1 スコア
MaltParser	HF_1	77.61	70.09	73.66
	HF_2	77.63	69.34	73.25
	PCH	**78.04**	71.13	**74.42**
	ACH_1	76.14	66.99	71.27
	ACH_2	76.55	67.33	71.64
	CWH	76.67	69.34	72.82
UDPipe	HF_1	76.58	71.33	73.86
	HF_2	77.39	70.19	73.86
	PCH	76.87	72.40	74.57
	ACH_1	77.79	70.82	74.14
	ACH_2	77.82	69.14	73.22
	CWH	**77.88**	71.67	**74.65**
SynCha		74.09	65.23	69.38
KNP		73.62	**73.85**	73.73

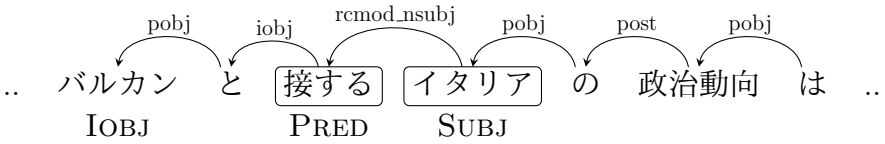
きる述語項の関係のみに絞った．最終的に，6,435 組を正解の述語項関係として抽出した．

本実験では，述語項構造解析器 SynCha [47] や項の情報を扱える構文解析器 KNP と条件を合わせて比較するため，平文を入力として MeCab で短単位への形態素解析を行い，Comainu での長単位チャンキングを経由して MaltParser または UDPipe で依存構造解析を行った（長単位依存構造解析）．SynCha は入力された平文に対して CaboCha で文節依存構造解析を行いその出力を用いて述語項構造解析を行い，KNP は入力された平文に対して文節依存構造解析を行い出力された格関係の情報を述語項構造情報に変換した[21]．

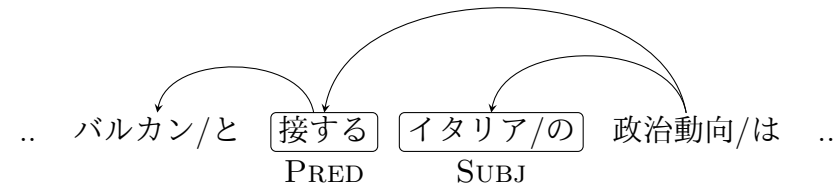
表 2.15 は，抽出された述語項の 3 つ組と正解データを比較した結果である．PCH, CWH は，構文解析精度の結果では HF_1, HF_2 に及ばなかったが，述語項構造情報の精度に関して，MaltParser では PCH, UDPipe では CWH が最も良い結果となっている．このことは，PCH, CWH が述語句内の内容語を主辞とすることにより，項になる要素と述語で直接の依存構造が構成される特徴によるものと考えられる．SynCha は，ゼロ代名詞の認定や照応解析も行っており直接的な比較はできないが，本論文の単語依存構造解析結果から抽出した述語項構造の精度は，構文解析（文節依存構造解析）と述語項構造解析を 2 段階で行っている SynCha の精度を十分に上回っており，依存構造解析時に同時に単語間の格関係を同定することが妥当であることを示していると考えられる．特に，PCH, CWH の結果は，再現率は KNP には及ばないが，適合率，F1 スコアでは同等以上の結果が得られている．

また，本論文の文法機能タイプ付き依存構造解析のように，述語項構造解析と構文解析

[21]Syncha は ver. 0.3.1，KNP は，ver. 4.1-beta を用いた．

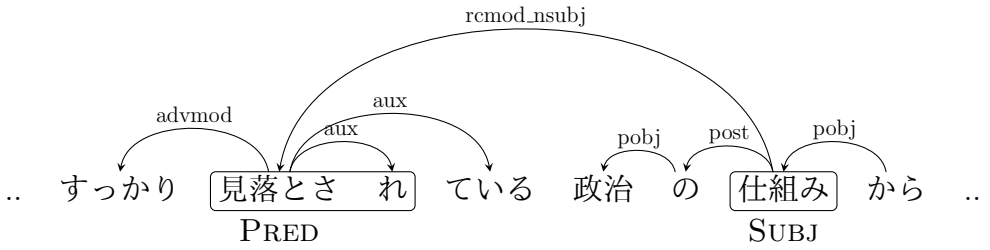


単語依存構造解析 (PCH) の結果.

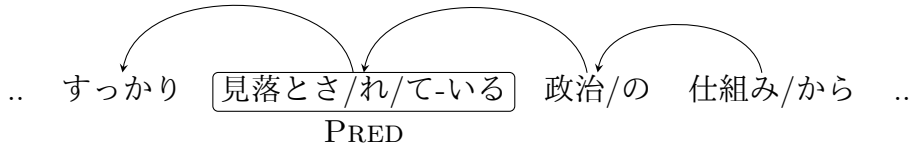


文節依存構造解析 + 述語項構造解析の結果.

図 2.15: 単語依存構造解析と述語項構造解析の結果の例（その 1）.



単語依存構造解析 (PCH) の結果.



文節依存構造解析 + 述語項構造解析の結果.

図 2.16: 単語依存構造解析と述語項構造解析の結果の例（その 2）.

と同時に行っている場合，構文解析と述語項構造解析を多段で行った場合に比べて両者の整合性を保持しやすい利点がある．典型的な例は,「庭から逃げた猫の足跡」のように主名詞を含む名詞句に「A の B」を持つような関係節を含む解析である．名詞 A が関係節の項に成る場合は，通常関係節の述語と名詞 A の間に依存関係があると考えられるが，単語依存構造解析の場合は，これらの関係を同時に扱うことができる．この例で，述語「逃げる」と統語的に関係のある語を「猫」とするのか,「足跡」とするのか判断が難しい場合であっても，述語「逃げる」の主格の項が「猫」になる可能性が高いと推定できれば，述語「逃げる」と名詞「猫」の間に *rcmod_nsubj* という文法機能タイプを持つ依存構造を

構成し，自動的に「逃げる」と「猫」の間に統語的関係があるという解析結果になる．

構文解析と述語項構造解析の多段処理の場合は，前段の構文解析での判断が難しく，誤った依存構造を構成してしまうと，後段の述語項構造解析でその誤りを伝播させる可能性が高まる．また，前段の構文解析結果と後段の述語項構造解析の結果で整合が取れる保証もない．

図 2.15 は，単語依存構造解析の結果，文節依存構造解析と述語構造解析の 2 段処理による結果の実際の例である．単語依存構造解析の結果では，述語「接する」と項「イタリア」が主格と述語の連体修飾関係 (*rcmod_nsubj*) で依存構造を構成しており構文解析結果と述語項構造解析結果が整合している．文節依存構造解析では，述語を含む文節「接する」と文節「政治動向は」で誤った依存構造を構成している一方で，述語項構造解析では正しい項「イタリア（の）」を抽出しており，両解析結果で整合が取れていない．また，図 2.16 の単語依存構造解析の結果では，述語「見落とされる」と項「仕組み」が主格と述語の連体修飾関係 (*rcmod_nsubj*) で依存構造を構成している．それに対し，文節依存構造解析では，述語を含む文節「見落とされている」と文節「政治の」で誤った依存構造を構成した結果，正しい項「仕組み」を抽出することに失敗している．実験で使用したテストセットに含まれるこのような構造を含む 75 例のうち，文法機能タイプ付き単語構造解析では 42 例で述語項構造と依存構造が正しく抽出できたのに対して，CaboCha，SynCha の多段処理で正しく抽出できたのものは，6 例に止まった．

2.6.5 評価のまとめ

2.6 節では，依存構造スキーマ，依存構造の単位，述語項構造情報の 3 点に着目して構文解析器への適用性を評価した．使用した構文解析器は，transition ベースの 2 種類に限定しており，素性設計を最適化しているわけではないため，確定的な結論は導き出せないが，評価実験によって得られた本論文の単語依存構造を解析器で使用する場合の傾向，特性について以下で述べる．

依存構造スキーマについては，後置詞句において機能語を主辞とする項機能語主辞型の HF_1, HF_2, PCH が構文解析の全体的な精度が高い傾向にある．特に，述語文節結合型と組み合わせた HF_2 が他の 2 つのスキーマより僅かに高い精度を出している．この結果は，格助詞等の機能語で後置詞の統語的役割を決め，述語は後続する機能語からなる述語句を単位として他の句と統語的関係を持つという日本語の特性を反映していると考えられる．

依存構造の単位を長単位にするか短単位にするかは，表 2.13 の MaltParser の結果のみからは判断が難しいが，UDPipe の長単位の結果が，ほぼ全てのスキーマにおいて，UDPipe 短単位の結果，MaltParser の長単位，短単位の結果を上回っていることから，解析器への適用面でも，構文構造を見通し良く表現できる長単位による依存構造を用いるのが良い

と考える．また，本実験の結果は長単位チャンキングの誤りを含んでいるので，チャンキングの精度向上によっても解析精度全体の向上が期待できる．

また，表 2.14 に示されるように，連用修飾関係以外の主要な文法機能タイプを持つ依存構造について，長単位ベースの項機能主辞型 HF_1, HF_2, PCH の単語依存構造解析の精度が文節依存構造解析の CaboCha の精度と同等以上の結果を出しており，日本語の構文解析として単語依存構造を用いることは実用性があると考える．

述語項構造情報の取得に関しては，全てのスキーマで 2 段階解析の SynCha の結果を上回っており，述語項構造情報を考慮しながら構文解析を行っている効果が表れていると考えている．述語項構造情報の取得に関しては，内容語主辞型の PCH, CWH が高い精度を出しており，格助詞等の手がかりがない関係節において，内容語同士の関係を捉えやすいスキーマの特性が反映されていると考えられる．

以上の結果から，構文解析の精度を重視する場合，長単位ベースで項機能語主辞型のスキーマ，特に HF_2 が第 1 候補として挙げられる．また，述語項構造情報を扱う場合は，PCH, CWH が有利だと考えられる．ただし，タスクに応じて複数の依存構造スキーマのコーパスや解析モデルを用意することは効率が悪いため，現実的には HF_2 など一つのスキーマをベースとして UD を含む他のスキーマへ変換可能な情報を付加した拡張スキーマを策定することが一つの有力な方法であると考える．

2.7 単語依存構造の課題

2.7.1 名詞句の内部構造の扱い

依存構造の単位として BCCWJDep の長単位を採用することで，機能語間の関係のように重要度の低い依存関係を無視できる反面，名詞句内の内部構造のような構造の表現能力には制約がある．例えば,「以降」のように短単位品詞が「名詞-普通名詞-副詞可能」である語や,「解消」のように「名詞-普通名詞-サ変可能」である語は，先行する名詞句と結合して一つの長単位を構成するため，前方から修飾する要素がある場合に長単位全体と依存関係を作ることになる．例えば,「昨年/の/事件以降」「項構造/の/曖昧性解消」という長単位に区切られるため「昨年の事件」「項構造の曖昧性」という単位の名詞句を切り出すことができない．この問題に関しては，全てを短単位の依存構造として扱う方法，あるいは，このような名詞句の構造を別の階層として扱う方法等が考えられるが，前者は解析精度への影響の面，後者は文節依存構造と同様に階層的構造の処理を別に考慮する複雑さが生じるという面がある．また，依存構造解析の単位として，長単位と短単位の中間的な単位を定義するという選択肢も考えられる．ただし，一貫性を確保できる明確な定義など，さらなる検討が必要である．

2.7.2 アスペクトおよびムードの扱い

本論文の依存構造では，複雑になることを避けるため，アスペクト，ムードを含む助動詞，補助動詞等を一括して扱い，項と述語のまとまりに対して後に結合するか（1 型），述語と先に結合するか（2 型）の 2 種類に分類した．しかしながら，寺村の分類 [48] にあるように，アスペクトを表す補助動詞を述語と結合して「コト」（叙述内容）の構造を一旦構成し，テンス，推量や説明等の概言のムードと呼んでいる助動詞，助詞類を外側に結合するという文の構成を，依存構造や文法機能タイプに反映させることは，文全体の意味解析処理に利用する上で有用であると考えられる．これを実現するためには，アスペクト，ムードの分類とともに，文法機能タイプ，依存構造への反映方法が検討事項となる．

2.7.3 Universal Dependencies との関係

UD は，依存構造スキーマ CWH と類似した構造を持っているが，本論文の評価結果からは，項機能語主辞型の HF_1, HF_2, PCH に比べて，精度面で低くなる傾向が見られている．また，本論文で文法機能タイプとして導入した連体修飾節の区別は UD では存在しないなど UD の方が情報量が少ない傾向にあるため，言語リソースを構築する際にはスーパーセットとして本論文で述べたようなスキーマで行い，UD に変換するという方法も有力であると考える[22]．構文解析器で UD の結果を得ようとする際には，直接 UD の構造を出力するのではなく，構造の近い PCH で解析結果を得て UD に変換することも選択肢になると考えられる．UD の主要な文法機能タイプは，連体修飾や付加項の細分類を除いてほぼ同等であるので，文法機能タイプの変換に関しては可能であると考えられる．構造的な変換に関しては，全面的に内容語が主辞になる構造や，先頭の要素が主辞になる並列構造への等への対応を検討する必要がある．

2.8 まとめ

本章では，日本語を多言語処理の中で扱うにあたって，他の言語と共通的な枠組みに基づきかつ日本語の特徴的な言語現象を表現可能な構文構造の表現方法として，文法機能タイプ付き単語依存構造解析を提案した．提案方法は，依存構造の単位が構文の構成素と整合し文法機能情報が利用可能になる点が従来の文節依存構造による構文解析に対して有利な点である．単語依存構造の適切な構成の仕方は自明でないため，6 タイプの依存構造スキーマを提案し，それぞれについて依存構造モデルを構築し構文解析器での解析精度について評価した．評価実験の結果，全般的に後置詞句においては機能語を主辞とするスキーマが精度的に有利であるという傾向があり，項機能語主辞型の単語依存構造解析は，

[22] UD の言語依存仕様としてサブタイプを定義する方法も考えられる．

文節依存構造解析と同等の精度が実現可能であることが確かめられた．また，依存構造の単位の違いによる比較では，ほぼすべてのスキーマにおいて長単位ベースの解析精度が短単位ベースの解析結果を上回っており，解析器への適用面および構文構造を見通しよく表現可能な点から長単位による依存構造を用いるのが適切と考えられる．さらに，単語依存構造解析の文法機能タイプにより得られる述語項構造情報は，構文解析と多段で行う述語項構造解析に匹敵する精度であることが確認された．このように，日本語において述語構造解析との統合が自然に行える単語依存構造解析が高い精度で行えることを示したことが本章の最大の貢献である．この構文解析とそれに基づく意味解析を利用した多言語対応処理は，第 4 章で述べる．

日本語に関する解析の詳細度を上げ意味解析との親和性をより高めるためには，名詞句の構造の洗練化やアスペクト，ムード等の文構造の反映方法が今後の検討事項であると考えている．また，多言語横断の構文解析の枠組みを目指した UD の対応についても，解析精度や情報量の面で有利である本論文で扱った依存構造スキーマを介して解析，変換を行うことは有力な方法の一つであると考える．

第3章　日本語の長単位チャンキングと単語依存構造の同時解析

3.1　はじめに

第2章で述べたように，日本語の構文解析は文節を単位とした依存構造で行うことが通常であるが一つの言語に閉じた処理ではなく様々な言語を対象とした多言語処理を行う場合には，日本語に関しても単語依存構造解析を行うことが有力な方法だと考える．一方で，日本語や中国語のように正書法として単語を空白で区切って記述しない言語に単語依存構造による構文解析を導入する場合，依存構造の単位として単語をいかに定義するかという課題がある．単語依存構造解析においては，単語単位の一貫性の不備が直接的に構文構造の不整合を生じさせるため単位となる単語の一貫性が重要になる．2.3.1 節 で述べたように，本論文では単語の基礎的な単位として短単位を採用している．短単位は単位認定の揺れが小さく形態素解析の単位として適しているが，構文構造を構成する単位としては短すぎる傾向があるため，本論文では単語依存構造解析の単位として1語以上の短単位が結合し文法的な機能を持つ長単位を採用した．すなわち，短単位と長単位の階層構造を持つ単語の定義に基づき，形態素の情報は短単位として保持し，その短単位1語以上が結合した長単位を単位とした依存構造解析を用いる．

単語に分かち書きのされていない日本語のテキストを入力として長単位を単位とした依存構造解析を行うためには，短単位を認定する（短単位に分割する）ことと，長単位を認定する（短単位列を結合して長単位へ変換する）ことが必要になる．ここで，最小単位となる単位列を結合して構成した新たな単位をチャンクと呼び，チャンクを構成していく行程をチャンキングと呼ぶ．日本語の伝統的な依存構造解析におけるチャンクは文節であるが，本論文の依存構造解析のチャンクは長単位である．

短単位列を認定した以降の最も基本的な解析方法は，長単位へのチャンキング（以下，長単位チャンキング）と長単位間の依存構造解析を逐次的に行うことであるが，長単位チャンキング時に長単位間で構成される依存構造の情報を参照できないことや，長単位チャンキングの誤りが次のステップである長単位間の依存構造構成時に悪影響を与える等の問題が生じる．

本章ではこの問題を解決するため，長単位チャンキングと長単位間依存構造解析を同時に行う方法を提案する．この方法では短単位と長単位からなる階層的単語構造を，長単位

内部の構造が短単位間の依存構造から構成されている構造として捉える．長単位間の依存構造は，それぞれの長単位を構成する中で代表となる短単位（主辞）の間の依存構造として表す．依存構造解析は，計算コストが軽く高い精度が見込める transition ベースの方法を採用する．すなわち長単位の依存構造を，長単位内部の依存構造（長単位を構成する短単位間の依存構造）と長単位外部の依存構造から構成されていると考え，入力された短単位列からこの両者の依存構造を同時に組み上げる．この際，長単位には長単位独自の品詞を付与する必要があるので，長単位を認定する際にこの長単位品詞も同時に同定する．

以下，3.2 節で短単位と長単位から構成される階層的単語依存構造について述べ，3.3 節で階層的単語依存構造の解析方法のうち，従来法である長単位チャンキングと単語依存構造解析を順次行う逐次解析処理による方法について述べ，3.4 節でこの両方の解析を同時に行う提案方法について述べる．3.5 節で関連研究について概観し，3.6 節で階層的単語構造を対象とした提案手法と単一の層から成る単語依存構造（短単位のみによる依存構造）の比較評価を行った結果について述べる．最後に 3.7 節で本章の成果についてまとめる．

3.2 階層的単語依存構造

日本語における「単語」を明確に定義することは簡単ではないが，2.3.1 節で述べたように構文構造を表す依存構造の単位としての単語は文法的な機能を持つ単位になっていることが望ましい．この要件を満たすために，本論文では単語依存構造の単位として長単位を採用している．長単位は 1 語以上の短単位の結合で構成され短単位よりも遥かに多様性が高く，長単位を安定して認定することは短単位に比較すると難しい．これに対して短単位は保持する文法的な機能は希薄であるが，テキストを分割して認定する単位としては曖昧性が低く安定性が高い．すなわち，単語に分かち書きされていない平文のテキストを形態素解析器により分割する単位として考えたときに，短単位は曖昧性を生じにくく一貫性の高い安定した結果を得ることができる．以上のようにテキストを構成している要素には，基盤として安定している分割の最小単位である短単位と，構文構造上で文法的な機能を担う長単位のそれぞれの特徴を持つ単語が階層的に存在していると考えることができる．そこで本論文の構文解析では，各文に含まれる形態素情報を短単位それぞれに持たせ，構文構造を長単位による単語依存構造により持たせる．このことにより，複合名詞や複合辞[1]を一つの構成単位（長単位）として扱うことができ，適切な文法的機能を持つ単位による依存構造を構成することが可能になる．短単位と長単位の関係は，日本語以外の言語においても，単語と複単語表現 (例えば，“*in spite of*” などの群前置詞) の関係として存在する．長単位のうち一つの短単位から構成されるものが約 80%を占め，複単語表現等複数の短単位から構成される長単位は全体の約 20%程度である．

[1]本論文では，複数の短単位が結合して機能語的な働きを担う長単位を指す．

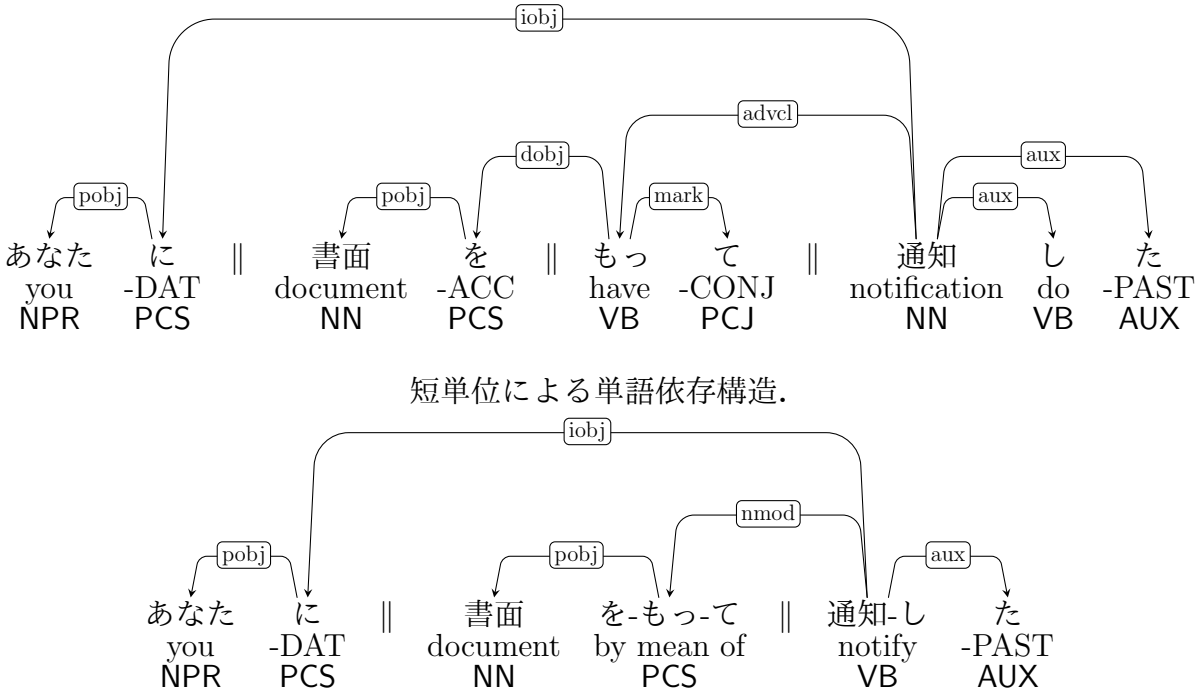


図 3.1: 短単位・長単位による単語依存構造の表現.

図 3.1 は同じ文を構文解析した結果である．上段が短単位による依存構造，下段が長単位による依存構造であり，異なる単位による依存構造を示している[2]．上段の短単位による依存構造では，複合辞の「を-もっ-て」を構成する動詞「もつ」は本動詞として扱われており，述語である動詞「もつ」と名詞「書面」を項とする述語項関係が表現されている．しかし，この「もつ」は本来の動詞としての意味は希薄であり，この短単位による依存構造では名詞「書面」と意味的な関係を持っている「通知する」との関係を隠蔽してしまうという欠点がある．図の下段の長単位による依存構造では，複合辞「を-もっ-て」を 1 語の長単位と捉えており，名詞「書面」と動詞「通知する」の関係を明確に表示することができる．このように構文構造を長単位の依存構造として扱うと，述語項構造の関係を捉えやすくなる．図 3.2 は図 3.1 の文から抽出した述語項構造を表しており，PREDICATE (述語)「通知する」に対して，意味役割 RECIPIENT (受領者) を持つ項が「あなた」，MANNER (方法) を持つ項が「書面」という関係になっている．これは長単位による依存構造において述語「通知する」が格助詞の長単位「に」を介して「あなた」と関係していること，および格助詞相当の長単位「を-もっ-て」を介して「書面」と関係していることと直接対応付けが行える．また，長単位にチャンキングする際には短単位列

[2]図中の依存構造スキーマは UD と類似しているが，主辞の選択方法が異なる．例えば，図のスキーマでは代名詞「あなた」に対して格助詞「に」を主辞として選択しているが，UD では代名詞「あなた」を主辞として選択する．

$$\left[\begin{array}{ll} \text{PREDICATE:} & \text{通知-する} \\ & \quad \text{notify} \\ \text{AGENT:} & \emptyset \\ \text{RECIPIENT:} & \text{あなた (に)} \\ & \quad \text{you-DAT} \\ \text{TOPIC:} & \emptyset \\ \text{MANNER:} & \text{書面 (を-もっ-て)} \\ & \quad \text{in writing} \end{array}\right]$$

図 3.2: 単語依存構造から抽出した述語項構造.

から長単位になるチャンクを決定するだけでなく，長単位に新たに付加する品詞（長単位品詞）を決定する必要がある．この例では，長単位「を-もっ-て」は格助詞 1 語相当の機能を担っており，長単位品詞として格助詞のシンボル PCS を付加されている．長単位チャンキングの際には，この長単位品詞を同定することも重要な要素である．

短単位依存構造から長単位依存構造への変換は曖昧性があるため，表層的に同一の文字列であっても構造を一意に決定することができない．例えば,「彼はその本をもっている」の文中の「をもって」の短単位列は格助詞相当の機能は持たないため，1 語の長単位とみなすのではなく，本動詞「もつ」を含む「を」「もっ」「て」の長単位 3 語とみなすのが適切である．このような曖昧性の解消は構文構造と密接に関係して行われるため，長単位を先に確定してから長単位間の依存構造を構築するよりも，長単位の同定と依存構造の構築を同時に行い決定しやすい方から確定する方法が解析精度面で有利だと考えられる．

3.3 従来手法：逐次処理による解析

3.3.1 長単位による依存構造解析に必要な処理

単語に分かち書きされていない平文のテキストを入力として長単位による依存構造を出力とする解析は，通常以下のような処理を逐次的に行う．

1. テキストを分割して短単位列に変換する（形態素解析）
2. 短単位列を結合して長単位列に変換する（長単位チャンキング）
3. 長単位列から長単位による依存構造を構成する（依存構造解析）

本章では，1 の形態素解析により平文のテキストから変換された短単位列を入力とする解析処理を扱う．図 3.3 は 2 の長単位チャンキングを行った結果の例である．上段は形態素

短単位	昨日	予備	調査	結果	に	つい	て	報告	し	た
(SUW)	NN	NN	NN	NN	PCS	VB	PCJ	NN	VB	AUX
	B	B	I	Ia	B	I	I	B	Ia	Ba
					↓					
長単位	昨日	予備調査結果			について			報告し		た
(LUW)	ADV	NN			PCS			VB		AUX

図 3.3: 長単位チャンキングの例

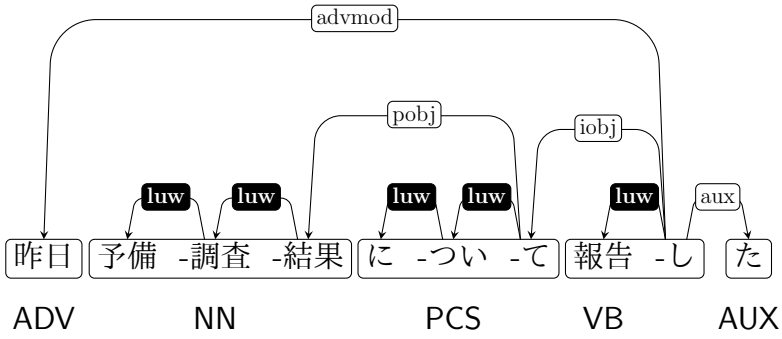


図 3.4: 長単位による依存構造の解析結果例.

解析によりテキストから変換された短単位列の例を示している．各単語表記の下には品詞シンボルにより各短単位の品詞（短単位品詞）を示している．下段は長単位チャンキングにより 1 語以上の短単位を結合して長単位を構成した結果の長単位列である．各長単位についても品詞シンボルにより各長単位の品詞（長単位品詞）が示されているが，長単位を構成している短単位品詞とは必ずしも一致するとは限らない．例えば，短単位「昨日」の品詞は NN（名詞） であるが，長単位「昨日」の品詞は，ADV（副詞）となっている．これは短単位品詞は単語ごとに代表的な品詞が付加されているのに対して，長単位品詞は文脈による文法機能を反映しているためである．このように長単位チャンキングの処理では，短単位を結合して長単位を構成するだけでなく長単位品詞も同定する必要がある．

長単位チャンキングにより短単位列を長単位列に変換した後，3 の依存構造解析により長単位列の間の依存構造を構築する．図 3.4 は依存構造解析により得られた長単位による依存構造の例である．枠で囲まれた単語列それぞれが長単位を表しており，長単位間に文法機能タイプ付きの弧を張ることにより長単位による依存構造を表している．後述するように，黒地に白抜きの文法機能タイプ *luw* で表されている弧は長単位を構成する短単位間の依存関係を示している．

以下，3.3.2 節で長単位チャンキング，3.3.3 節で依存構造解析のそれぞれ代表的な手法を述べ，3.3.4 節で逐次的処理による解析の問題点を述べる．

3.3.2 系列ラベリングによる長単位チャンキング

前節で述べたように長単位チャンキングの処理では，短単位列を入力として短単位を結合して長単位を構成することと，長単位それぞれの品詞を同定する必要がある．このように，入力された単位列のチャンクへの結合とチャンクに対するラベルの付加という処理は，系列ラベリングによる固有表現抽出の処理と類似している [49, 50]．小澤らは，短単位列を入力として長単位に結合するチャンキングモデルと，結合した長単位に対して長単位品詞を推定するカテゴリ推定モデルを構築して，2 段階で解析する手法を提案している [41]．単語列から人名や場所名等を表すチャンクを抽出する固有表現抽出では，チャンクの始まりである単語に B タグを，チャンクの先頭以外の単語に I タグを，チャンク以外の単語に O タグを系列ラベリングによって付与することによりチャンキングを行う方法がある．長単位チャンキングでもこれと同様に，入力された短単位列について長単位の先頭になる短単位に B タグを，長単位の先頭以外になる短単位に I タグを付与する系列ラベリングを行うことにより実現することができる．なお，長単位チャンキングでは全ての短単位が必ずいずれかのチャンク（長単位）に属するので，チャンク以外を示す O タグは使用しない．

小澤らの手法の概要は以下の通りである．

1. **長単位チャンクの認定：** 入力された短単位列について，長単位の先頭の短単位 B タグ，長単位の先頭以外の短単位 I タグを同定する系列ラベリングを行う．ただし，単独で長単位を構成しかつ長単位品詞が短単位品詞[3]と一致する短単位には Ba タグを，複数の短単位から構成される長単位の末尾かつ長単位品詞が短単位と一致する短単位には Ia タグを付与する．

2. **長単位品詞の推定：** 前の手順で Ba タグあるいは Ia タグのついた短単位を持つ長単位は，その短単位の品詞を長単位品詞とする．Ba タグも Ia タグも含まない長単位に対しては，前後 1 長単位の文脈情報から構築された長単位品詞推定モデルを用いて，対象の長単位の品詞を推定する．ただし，「助詞」「助動詞」は別に設けた複合辞テーブルと表層形が一致しない限り候補の品詞に含めない．

図 3.3 の上段の短単位列の品詞シンボルの下の列には，小澤らの方法により長単位チャンキングのためのタグを付与した結果を示している．図の例では， B タグ（あるいは Ba タグ）を先頭とする 5 つの長単位「昨日」,「予備-調査-結果」,「に-つい-て」,「報告-し」,「た」が抽出される．長単位品詞の同定は短単位に付与されたタグによって異なる方法で決定する．Ba タグ あるいは Ia タグを含む長単位の品詞は，元の短単位品詞から自動的に決定される．すなわち Ba タグを付与した短単位は，その短単位 1 語で長単位を構成し

[3]元の手法では活用型，活用形を考慮するが本論文では品詞シンボルのみを対象とする．

表 3.1: 依存構造解析で定義した action.

action	説明
`Shift`	キュー q_0 から長単位 1 語を取り出しスタック σ の最上位 σ_0 に入れる
$\texttt{Reduce}_L(dep)$	スタック σ の上位 σ_0, σ_1 の長単位の間で依存構造を構成する (σ_0 が主辞となる左向きの弧を結ぶ．文法機能タイプ dep)
$\texttt{Reduce}_R(dep)$	スタック σ の上位 σ_0, σ_1 の長単位の間で依存構造を構成する (σ_1 が主辞となる右向きの弧を結ぶ．文法機能タイプ dep)

表 3.2: 依存構造解析における action と transition.

action	transition	条件
`Shift`	$(\sigma, Q\|x_k, A) \Rightarrow (\sigma\|x_k, Q, A)$	$\|Q\| \geq 1$
$\texttt{Reduce}_L(r)$	$(\sigma\|x_k\|x_m, Q, A) \Rightarrow (\sigma\|x_k, Q, A \cup \{r(x_k, x_m)\})$	$\|\sigma\| \geq 2$
$\texttt{Reduce}_R(r)$	$(\sigma\|x_k\|x_m, Q, A) \Rightarrow (\sigma\|x_m, Q, A \cup \{r(x_m, x_k)\})$	$\|\sigma\| \geq 2$

x_k　長単位

$r(x, y)$　単語依存構造（主辞: x, 依存構造タイプ: r）

初期状態：$([\texttt{ROOT}], [x_0, ..., x_n], \emptyset)$

最終状態：$([\texttt{ROOT}], [], A_f)$

短単位品詞がそのまま長単位品詞に引き継がれる．図 3.3 の例では，Ba タグを付与した短単位「た」の短単位品詞 AUX（助動詞）は，そのまま長単位「た」の品詞に引き継がれる．また，Ia タグを付与した短単位「調査」の短単位品詞 NN（名詞）は，それを含む長単位「予備-調査-結果」の品詞に引き継がれる．Ba タグ，Ia タグどちらも含まない長単位の場合は短単位品詞から自動的に決定できないため，単語の前後の文脈から構築した品詞推定モデルを用いて決定する．図の例では，長単位「昨日」と「に-つい-て」それぞれを構成する短単位には Ba タグ，Ia タグを持つものが存在しないため，元の短単位とは別の品詞を長単位に付与する必要がある．この 2 つの長単位に対しては，手順 2 の長単位品詞推定の手順によってそれぞれ長単位品詞 ADV，PCS が付与されている．

3.3.3 Transition ベースの手法による依存構造解析

本節では，入力された単語列（ここでは長単位列）から依存構造を構築する解析方法の中で代表的な transition ベースの手法 [42, 51, 52] について述べる．Transition ベースの手法では，入力となる単語列から 1 語ずつ入力していき，最新の入力語とその時点での構築している依存構造との間に依存関係を結ぶかどうか（結ぶ場合は文法機能ラベルも決定する）を順次判断して依存構造を構成することを繰り返し構文解析を行う．この判断は正解の依存構造データから機械学習により構築した解析モデルにより行う．この方法では，入力単語列（長単位列）を格納し 1 語ずつ順次供給するキューと，部分的な依存構造を構

Algorithm 1: LUW-based dependency parsing

```
01 function luwdep_parse():
# スタック，キューの初期化
02   σ = [ROOT] # スタック先頭に ROOT を入れる
03   Q = [(入力文の長単位列)]
# メインループ
04   call Shift()
05   while TRUE:
06     (act,dep) = call getNextAction(feat)
07     if (act == Shift):
08       call Shift()
09     else if (act == Reduce_L):
10       call Reduce_L(dep)
11     else if (act == Reduce_R):
12       call Reduce_R(dep)
13     if (|σ| == 1 and |Q| == 0 ):
14       exit
15   end while

# q_0 の長単位をスタック σ に push する
16 function Shift():
17   push (σ, q_0)
18   pop(Q)

# σ_0 から σ_1 へ左向きの長単位の弧 (文法機能ラベル dep) を張る
19 function Reduce_L(dep):
20   LeftArc(σ_0, σ_1, dep)
21   remove(σ, σ_1)

# σ_1 から σ_0 へ右向きの長単位の弧 (文法機能ラベル dep) を張る
22 function Reduce_R(dep):
23   RightArc(σ_1, σ_0, dep)
24   remove(σ, σ_0)
```

図 3.5: 長単位の依存構造解析のアルゴリズム．

成していくスタックを用意する．1 ステップごとに，一つの長単位をキューからスタックに移す shift 操作，あるいはスタックの最上位の部分木と 2 番目の部分木の間で依存関係を構築する reduce 操作を行い，キューが空の状態でかつスタックの最上位に依存構造が完成した状態（最終状態）になるまで繰り返すことが基本となる．

長単位による依存構造は，依存関係にある 2 つの長単位 x_i, x_j とその間の文法機能ラベル (依存関係ラベル) r の組 $r(x_i, x_j)$ の集合 A で表現できる．ここで x_i は主辞となる長単位である．A の依存構造は根ノードを一つ持つ木構造になっている．キューを Q，スタックを σ とすると，依存構造解析器 (構文解析器) の内部状態は 3 つ組 triple (σ, Q, A) として表される．初期状態ではキュー Q は入力文全体の長単位列 $[x_0, ..., x_n]$ 全てを含ん

Step	Action	Stack				Queue	
		σ_3	σ_2	σ_1	σ_0	q_0	q_1
0	–				ROOT	昨日	予備-調査-...
1	Shift			ROOT	昨日	予備-調査-...	に-つい-て
2	Shift		ROOT	昨日	予備-調査-結果	に-つい-て	報告-し
3	Shift	ROOT	昨日	予備-調査-結果	に-つい-て	報告-し	た
4	Reduce$_L$($pobj$)		ROOT	昨日	に-つい-て	報告-し	た
				予備-調査-結果 $\overset{pobj}{\swarrow}$			
5	Shift	ROOT	昨日	に-つい-て	報告-し	た	–
			予備-調査-結果 $\overset{pobj}{\swarrow}$				
6	Shift	昨日	に-つい-て	報告-し	た	–	–
		予備-調査-結果 $\overset{pobj}{\swarrow}$					
7	Reduce$_R$(aux)	ROOT	昨日	に-つい-て	報告-し	–	–
			予備-調査-結果 $\overset{pobj}{\swarrow}$		$\overset{aux}{\searrow}$ た		
8	Reduce$_L$($iobj$)		ROOT	昨日	報告-し	–	–
				に-つい-て $\overset{iobj}{\swarrow}$ $\overset{aux}{\searrow}$ た			
			予備-調査-結果 $\overset{pobj}{\swarrow}$				
9	Reduce$_L$($advmod$)			ROOT	報告-し	–	–
				昨日 $\overset{advmod}{\swarrow}$ に-つい-て $\overset{iobj}{\swarrow}$ $\overset{aux}{\searrow}$ た		
				予備-調査-結果 $\overset{pobj}{\swarrow}$			
10	Reduce$_R$($root$)				ROOT	–	–
					$\overset{root}{\searrow}$ 報告-し		
				昨日 $\overset{advmod}{\swarrow}$ に-つい-て $\overset{iobj}{\swarrow}$ $\overset{aux}{\searrow}$ た			
				予備-調査-結果 $\overset{pobj}{\swarrow}$			

$\overset{dep}{\swarrow}$, $\overset{dep}{\searrow}$: 長単位間の弧 (文法機能ラベル dep).

図 3.6: 長単位による依存構造解析の動作例.

だ状態であり，スタック σ は最終的な依存構造木の根ノード ROOT のみが格納された状態である．つまり解析前の初期状態は $([\text{ROOT}], [x_0, ..., x_n], \emptyset)$ であり，解析結果の依存構造の集合を A_f とすると最終状態は $([\text{ROOT}], [], A_f)$ と表すことができる[4]．構文解析は，初期状態から表 3.1 の 3 種類の操作 (action) を適用することにより最終状態に遷移させることである．表 3.2 は 3 種類の action を適用したときにスタックやキューの状態がどう遷移するか (transition) を表しており，右端の欄が各 action を適用可能な必要条件を示している．$|Q|$，$|\sigma|$ という記法はキューやスタックの要素数を表している．例えば，$|\sigma| = 0$ はスタック σ が空の状態であるという条件を示している．

図 3.5 に依存構造解析のアルゴリズムを示す．図ではキューを $Q = [q_0, q_1, ...]$，スタックを $\sigma = [\sigma_0, \sigma_1, ...]$ で表しており，q_n には長単位が，σ_m には依存構造木の部分木が入る．初期値としてスタックには ROOT という依存構造木の根を表すノードを格納し，キューには入力文の長単位列を格納する（2-3 行目）．表 3.1 に示すスタック，キューに対する 3 種類の操作 (action) はアルゴリズムの中の同名の関数 (function) と対応している．メインループで

[4] この記法では，スタックは依存構造木の根ノードの長単位のみを示す (ROOT を含む)

action を繰り返しながらスタック σ 内に依存構造を構成する．関数 `getNextAction`($feat$) は，特徴量 $feat$ を引数として与えると解析モデルを用いて次の action を返す関数である．解析モデルはあらかじめ依存構造の正解データから機械学習により構築する．最終的にキューが空になり ($|Q| = 0$) かつスタック内の要素が最上位の 1 要素のみになる ($|\sigma| = 1$) ときに終了する．このとき，スタックの最上位 (σ_0) に解析結果である `ROOT` を根ノードとする依存構造木が格納されている．

図 3.6 に，図 3.3 の長単位列を入力とした場合の依存構造解析の動作例を示す．入力された長単位列 (「昨日」,「予備調査結果」,「について」,「報告し」,「た」) がキュー Q の先頭からスタック σ に順に供給されスタック内に依存構造を構成していき，10 ステップ目を実行した後 σ_0 に解析結果である依存構造木 (図 3.4 と同等) が格納される．

3.3.4　逐次処理による解析の問題点

以上に説明したような長単位チャンキングと依存構造解析を逐次的に行う方法は処理が単純に行えることが利点であるが，長単位チャンキングの曖昧性解消と依存構造解析の曖昧性解消の処理の順序を固定しているため依存構造を先に決定した方が容易である場合にも，必ず先に前段の長単位チャンキングの曖昧性を解消しなければならないという点が短所である．すなわち前段の長単位チャンキングで解析を誤った場合，その影響が直接後段の依存構造解析に及ぶ可能性がある．例えば類似した 2 文「彼について長く考える」,「彼について長く走る」において，最初の文の「について」は格助詞相当の機能を持つ 1 語の長単位であり，後の文の「について」は動詞を含む「に」「つい」「て」という 3 語の長単位である．この構造の曖昧性を解消するには全体の構文構造を考慮する必要があるが，逐次処理では必ず先に「について」の単位が 1 語の長単位であるのか，それとも 3 語の長単位であるのかを決定しなければならず，この固定された処理順序が解析精度に影響する恐れがある．そこで本論文では，長単位チャンキングと長単位間の依存構造解析を同時に行い曖昧性を解消しやすい箇所から解析を行う方法を提案する．

3.4　長単位チャンキングと依存構造の同時解析

本節では，提案手法である長単位チャンキングと依存構造解析の同時解析について説明する．同時解析では，入力された「短単位列を長単位に結合する操作」と「長単位間の依存構造を構築する操作」を適切な順序で繰り返すことにより長単位の依存構造を作り上げる．この 2 種類の操作を柔軟に行うため，長単位内部の短単位間にフラットな依存構造が存在するとみなして，長単位による依存構造全体が短単位による依存構造で表現された形で依存構造解析を行う．ここで，フラットな依存構造とはそれぞれの短単位について

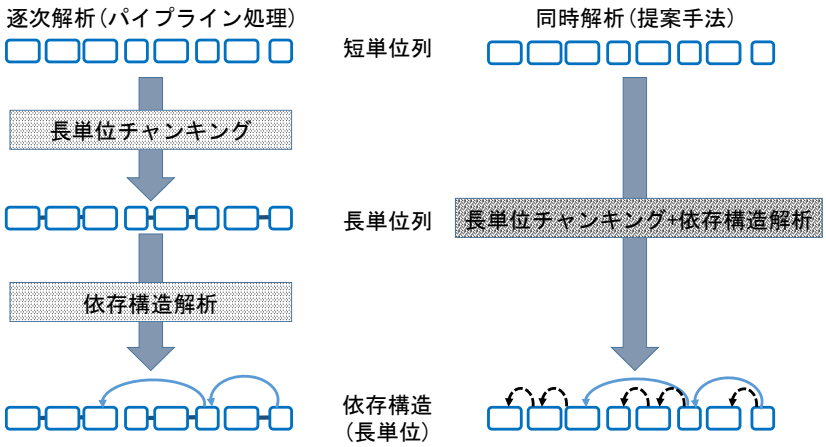


図 3.7: 長単位依存構造の解析手法.

表 3.3: 提案手法（同時解析手法）で定義した action.

action	説明
長単位チャンキングに関する action	
ShLUW($lpos$)	キュー q_0 から短単位 1 語を取り出しスタック σ_S の最上位 σ_{S0} に入れる (長単位先頭を構成する短単位, 長単位品詞 $lpos$)
ShSUW	キュー q_0 から短単位 1 語を取り出しスタック σ_S の最上位 σ_{S0} に入れる (長単位先頭以外を構成する短単位)
ReSUW$_L$	スタック σ_S の上位 σ_{S0}, σ_{S1} の短単位の間で長単位内部の依存構造を構成する (σ_{S0} が主辞となる左向きの弧を結ぶ．内部文法機能タイプ dep)
ReSUW$_R$	スタック σ_S の上位 σ_{S0}, σ_{S1} の短単位の間で長単位内部の依存構造を構成する (σ_S が主辞となる右向きの弧を結ぶ．内部文法機能タイプ dep)
PopLUW	スタック σ_S の最上位 σ_{S0} の部分木を長単位として確定させる (スタック σ_S から σ_{S0} を取り出し，スタック σ_L に σ_{L0} として入れる)
長単位間依存構造解析に関する action	
ReLUW$_L$(dep)	スタック σ_L の上位 σ_{L0}, σ_{L1} の長単位の間で依存構造を構成する (σ_{L0} が主辞となる左向きの弧を結ぶ．文法機能タイプ dep)
ReLUW$_R$(dep)	スタック σ_L の上位 σ_{L0}, σ_{L1} の長単位の間で依存構造を構成する (σ_{L1} が主辞となる右向きの弧を結ぶ．文法機能タイプ dep)

右側に隣接する短単位を主辞とするような構造を指す．図 3.4 は入力された短単位列から同時解析によって直接長単位間の依存構造を構築した結果を示している．この構造には，各長単位を構成する短単位列が luw のラベルの付加された依存構造として含まれている．複数の短単位から構成される助詞句「を-もっ-て」やサ変動詞「通知-する」が 1 語の長単位として認定され，長単位内部の依存構造は特別な依存構造タイプ luw によるフラットな構造で表されている．

表 3.4: 提案手法（同時解析手法）における action と transition.

action	LUW transition		条件
ShLUW(p)	$(\sigma_S, \sigma_L, \beta \vert x_k, A, L)$	$\Rightarrow(\sigma_S \vert p\langle x_k\rangle, \sigma_L, \beta, A, L)$	$\vert\sigma_S\vert = 0$
ReLUW$_L$(r)	$(\sigma_S, \sigma_L \vert p\langle x_k\rangle \vert q\langle x_m\rangle, \beta, A, L)$	$\Rightarrow(\sigma_S, \sigma_L \vert p\langle x_k\rangle, \beta, A \cup \{r(p\langle x_k\rangle, q\langle x_m\rangle\}, L)$	$\vert\sigma_L\vert \geq 2$
ReLUW$_R$(r)	$(\sigma_S, \sigma_L \vert p\langle x_k\rangle \vert q\langle x_m\rangle, \beta, A, L)$	$\Rightarrow(\sigma_S, \sigma_L \vert q\langle x_m\rangle, \beta, A \cup \{r(q\langle x_m\rangle, p\langle x_k\rangle\}, L)$	$\vert\sigma_L\vert \geq 2$
PopLUW	$(\sigma_S \vert p\langle x_k\rangle, \sigma_L, \beta, A, L)$	$\Rightarrow\{\sigma_S, \sigma_L \vert p\langle x_k\rangle, \beta, A, L \cup \{p\langle x\rangle\})$	$\vert\sigma_S\vert = 1$
action	SUW transition		条件
ShSUW	$(\sigma_S \vert p\langle x_k\rangle, \sigma_L, \beta \vert x_m, A, L)$	$\Rightarrow(\sigma_S \vert p\langle x_k\rangle \vert x_m, \sigma_L, \beta, A, L)$	$\vert\sigma_S\vert \geq 1$
ReSUW$_L$	$(\sigma_S \vert p\langle x_k\rangle \vert x_m, \sigma_L, \beta, A, L)$	$\Rightarrow(\sigma_S \vert p\langle x_k\rangle, \sigma_L, \beta, A, L \cup \{\ell(x_k, x_m)\})$	$\vert\sigma_S\vert \geq 2$
ReSUW$_R$	$(\sigma_S \vert p\langle x_k\rangle \vert x_m, \sigma_L, \beta, A, L)$	$\Rightarrow(\sigma_S \vert p\langle x_m\rangle, \sigma_L, \beta, A, L \cup \{\ell(x_m, x_k)\}$	$\vert\sigma_S\vert \geq 2$

x_k 短単位
$p\langle x\rangle$ 長単位の品詞 (主辞: x)
$r(x, y)$ 単語依存構造 (主辞: x, 依存構造タイプ: r)
$\ell(x, y)$ 長単位内部の依存構造
初期状態： $([\texttt{ROOT}], [], [x_0, ..., x_n], \emptyset, \emptyset)$
最終状態： $([\texttt{ROOT}], [], [], A_f, L_f)$

それぞれの操作は 3.3.3 節で述べた transition ベースの依存構造解析を拡張した方法により進める．同時解析による方法では，図 3.7 の右側のフローで示すように長単位列ではなく短単位列を入力として 1 段階の処理で長単位による依存構造を構築する．長単位チャンキングと依存構造解析の 2 種類の操作を行うために，shift 操作，reduce 操作ともに複数の種類を用意する．本論文の方法では，キュー Q から入力される短単位列を 2 種類のスタック σ_S, σ_L を使用して長単位列と依存構造木を別々に構成して出力する．すなわち，スタック σ_S を 短単位間の依存構造（長単位内部の依存構造）を構成するために，スタック σ_L を 長単位間の依存構造を構築するために使用する．本論文のアルゴリズムは，長単位列と長単位による依存構造をそれぞれ長単位の内部依存構造 L と長単位間の依存構造 A として出力する．構文解析器の内部状態は 5 つ組 quintuple $(\sigma_S, \sigma_L, Q, A, L)$ として表される．ここで Q は初期状態では入力文全体の短単位列 $[x_0, ..., x_n]$ 全てを含んでいる．

表 3.3 は提案手法である同時解析アルゴリズムで使用する action のリストを，図 3.4 は action を適用することによりスタックやキューの状態がどう遷移するか (transition) を示している．それぞれの action を適用可能な必要条件は最も右側の列に示されている．図 3.8 はこれらを用いた同時解析のアルゴリズムである．前述したように，長単位チャンキングでは長単位を認定するだけでなくそれぞれに対して長単位品詞を付与する必要があるが，本論文の方法の特徴は shift 操作 (ShLUW) を行う際に長単位を構成する最も左の短単位に新しい品詞を一時的に割り当てることである．この方法は，Pop 操作 (PopLUW) の際に品詞を決定する方法よりも，良い精度が得られることが実験的に確かめられている．

Algorithm 2: Joint parsing

```
01 function joint_parse():
# スタック，キューの初期化
02   σ_L = [ROOT] # 先頭 σ_L0 に ROOT を入れる
03   σ_S = []
04   Q = [(入力文の短単位列)]
# メインループ
05   call ShLUW(pos)
06   while TRUE:
07     (act,dep,pos) = call getNextAction(feat)
08     if (act == processSUW):
09       call processSUW()
10       continue
11     if (act == processLUW):
12       call processLUW()
13   end while

# 長単位に関する処理
14 function processLUW():
15   call PopLUW()
16   while TRUE:
17     (act,dep,pos) = call getNextAction(feat)
18     if (act == ReduceLUW_L):
19       call ReLUW_L(dep)
20       if (σ_L0 == ROOT and q_0 == NULL):
           exit
21     else (act == ReduceLUW_R):
22       call ReLUW_R(dep,pos)
23       if (σ_L0 == ROOT and q_0 == NULL):
           exit
24     else (act == ShiftLUW):
25       call ShLUW(pos)
26       return
27   end while

# 短単位に関する処理
28 function processSUW():
29   call ShSUW()
30   (act,dep,pos) = call getNextAction(feat)
31   if (act == ReduceLUW_L):
32     call ReSUW_L()
33   else (act == ReduceLUW_R):
34     call ReSUW_R()
35   return

# q_0 の短単位を品詞 pos の長単位の
# 先頭としてスタック σ_S に push する
36 function ShLUW(pos):
37   push (σ_S, q_0)
38   pop(Q)
39   SetPOS(σ_S0, pos)

# q_0 の短単位を短単位として
# スタック σ_S に push する
40 function ShSUW():
41   push (σ_S, q_0)
42   pop(Q)

# σ_S0 を長単位として確定させて
# スタック σ_L に push する
#（長単位の品詞は shLUW の引数 pos）
43 function PopLUW():
44   push(σ_L, σ_S0)
45   pop(σ_S)
46   return

# 左向きの短単位の弧
47 function ReSUW_L():
48   LeftSArc(σ_S0, σ_S1)
49   remove(σ_S, σ_S1)

# 右向きの短単位の弧
50 function ReSUW_R():
51   RightSArc(σ_S1, σ_S0)
52   remove(σ_S, σ_S0)

# 左向きの長単位の弧
# (文法機能ラベル dep)
53 function ReLUW_L(dep):
54   LeftLArc(σ_L0, σ_L1, dep)
55   remove(σ_L, σ_L1)

# 右向きの長単位の弧
# (文法機能ラベル dep)
56 function ReLUW_R(dep):
57   RightLArc(σ_L1, σ_L0, dep)
58   remove(σ_L, σ_L0)
```

図 3.8: 長単位チャンキングと依存構造の同時解析アルゴリズム．

図 3.9，図 3.10 に「昨日予備調査結果について報告した」という文の短単位列を入力したときの，同時解析の動作例を示す．キュー Q からは短単位が順次スタック σ_S に供給され，短単位間の依存構造（長単位の内部構造）を構成する．PopLUW の action によりスタック σ_S 内の短単位から構成される部分構造が長単位として同定されるとスタック σ_L に移される．スタック σ_L 上では，長単位「昨日」「予備-調査-結果」「に-つい-て」「報告-し」「た」が，それぞれの品詞 ADV(副詞)，NN(名詞)，PCS(格助詞) とともに順次同定されている．スタック σ_L 上では，これらの長単位間の依存構造により ROOT を根ノードとする依存構造木が構成されている．

3.5 関連研究

従来の多くの日本語の依存構造による構文解析は，依存構造の単位となるチャンクを同定して，そのチャンク間の依存構造を求めるという 2 段階の処理により行われてきた．依存構造の単位となるチャンクは通常文節であり文節の認定に関しては曖昧性が小さいことから，単語列に現れた品詞列のパタン等で文節へのチャンキングを行った後に，文節間の依存構造を求めることが一般的である [13, 14]．なお，日本語版 UD では依存構造の単位として短単位を採用しているためチャンキングは不要であり，UDPipe [43] などの UD の構文解析器では形態素解析結果の短単位列から直接依存構造を構築する．

日本語の短単位と長単位の階層的単語構造において長単位チャンキングを行う解析方法については前述した小澤らの研究 [41] がある．小澤らは固有表現抽出で用いられる系列ラベリングの手法を用いて，短単位の列から長単位のチャンクを同定するとともに長単位の品詞を求める手法を提案している．短単位，長単位情報をアノテーションしたコーパスを学習データとして条件付き確立場 (Conditional Random Field, CRF) を用いて構築した解析モデルによりチャンキングの曖昧性を解消する．ただし，長単位チャンキングと構文解析とは独立に考えらえれているため，長単位の曖昧性と構文構造の曖昧性とを同時に解消することができない．

Transition ベースで複数の異なる階層の言語解析を同時に行う手法としては，中国語における単語分割と構文解析を行った Zhang らの研究 [53, 54] や，複単語表現解析と構文解析を行った Constant らの研究 [55] がある．これらの手法および我々の手法では，異なる階層の解析ごとに異なる操作を定義し各操作の適用方法を学習データからモデル化するという点で共通している．我々の方法の特徴の一つは，短単位と長単位という階層ごとに短単位から長単位を構成する操作，長単位から長単位間の依存構造を構成する操作を定義し，後者の操作のパラメータとして長単位品詞を含めている．長単位依存構造を構成する際に同時に長単位品詞を決定することにより，逐次的処理では品詞決定の後に依存構造決定を行わなければならない順序的制約を解消することができる．

Step	Action	Stack				Queue	
		σ_{L3}	σ_{L2}	σ_{L1}	σ_{L0}	q_0	q_1
		[σ_{S3}	σ_{S2}	σ_{S1}	σ_{S0}]		
0	–				ROOT	昨日	予備
		[			-]		
1	ShLUW(ADV)				ROOT	予備	調査
		[			昨日]		
2	PopLUW			ROOT	昨日/ADV	予備	調査
		[			–]		
3	ShLUW(NN)			ROOT	昨日/ADV	調査	結果
		[			予備]		
4	ShSUW			ROOT	昨日/ADV	結果	に
		[		予備	調査]		
5	ReSUW$_L$			ROOT	昨日/ADV	結果	に
		[			予備←調査]		
6	ShSUW			ROOT	昨日/ADV	に	つい
		[		予備←調査	結果]		
7	ReSUW$_L$			ROOT	昨日/ADV	に	つい
		[			予備←調査←結果]		
8	PopLUW		ROOT	昨日/ADV	予備←調査←結果/NN	に	つい
		[			–]		
9	ShLUW(PCS)		ROOT	昨日/ADV	予備←調査←結果/NN	つい	て
		[			に]		
10	ShSUW		ROOT	昨日/ADV	予備←調査←結果/NN	て	報告
		[		に	つい]		
11	ReSUW$_L$		ROOT	昨日/ADV	予備←調査←結果/NN	て	報告
		[			に←つい]		
12	ShSUW		ROOT	昨日/ADV	予備←調査←結果/NN	報告	し
		[		に←つい	て]		
13	ReSUW$_L$		ROOT	昨日/ADV	予備←調査←結果/NN	報告	し
		[			に←つい←て]		
14	PopLUW	ROOT	昨日/ADV	予備←調査←結果/NN	に←つい←て/PCS	報告	し
		[			–]		
15	ReLUW$_L(pobj)$		ROOT	昨日/ADV	に←つい←て/PCS	報告	し
					予備←調査←結果/NN $\overset{pobj}{\swarrow}$		
		[			–]		
16	ShLUW(VB)		ROOT	昨日/ADV	に←つい←て/PCS	し	た
					予備←調査←結果/NN $\overset{pobj}{\swarrow}$		
		[			報告]		

“/POS”: 確定した長単位品詞 POS.

$\overset{dep}{\swarrow}$, $\overset{dep}{\searrow}$: 長単位間の弧 (文法機能ラベル dep).

図 3.9: 長単位チャンキングと依存構造解析の同時解析の動作例.

Step	Action	Stack				Queue	
		σ_{L3}	σ_{L2}	σ_{L1}	σ_{L0}	q_0	q_1
		[σ_{S3}	σ_{S2}	σ_{S1}	σ_{S0}]		
17	ShSUW		ROOT	昨日/ADV	に↼つい↼て/PCS 予備↼調査↼結果/NN (pobj) 報告 し]	た	–
18	ReSUW$_L$		ROOT	昨日/ADV	に↼つい↼て/PCS 予備↼調査↼結果/NN (pobj) [報告↼し]	た	–
19	PopLUW	ROOT	昨日/ADV	に↼つい↼て/PCS 予備↼調査↼結果/NN (pobj)	報告↼し/VB [–]	た	–
20	ReLUW$_L$($iobj$)		ROOT	昨日/ADV	報告↼し/VB に↼つい↼て/PCS (iobj) 予備↼調査↼結果/NN (pobj) [–]	た	–
22	ShLUW(AUX)		ROOT	昨日/ADV	報告↼し/VB に↼つい↼て/PCS (iobj) 予備↼調査↼結果/NN (pobj) [た]	–	–
23	PopLUW	ROOT	昨日/ADV	報告↼し/VB に↼つい↼て/PCS (iobj) 予備↼調査↼結果/NN (pobj)	た/AUX [–]	–	–
24	ReLUW$_L$($advmod$)			ROOT	報告↼し/VB 昨日/ADV (advmod) に↼つい↼て/PCS (iobj) (aux) た 予備↼調査↼結果/NN (pobj) [–]	–	–
25	ReLUW$_L$($root$)				ROOT 報告↼し/VB (root) 昨日/ADV (advmod) に↼つい↼て/PCS (iobj) (aux) た 予備↼調査↼結果/NN (pobj) [–]	– –	– –

図 3.10: 長単位チャンキングと依存構造解析の同時解析の動作例（図 3.9 の続き）.

表 3.5: コーパス統計量.

データセット		文数	長単位数	短単位数
KyotoDep	訓練データ	17,953	383,796	497,344
	評価データ	2,000	41,182	53,192

3.6 評価実験

本節では，提案手法である長単位チャンキングと長単位間の依存構造解析の同時解析による構文解析の有効性を検証する評価実験について述べる．同時解析手法の利点は，柔軟な順序で長単位と依存構造の曖昧性を解消することにより複合辞を含む長単位をより的確に同定することが可能になり，文の構造の中心的な役割を担う述語項構造を精度良く捉えるとともに全体解析精度の向上が図られる点である．提案手法であるチャンキング，依存構造解析を同時に解析する手法の他に，チャンキングと依存構造解析を逐次的に解析する方法それぞれにより評価文 2,000 文を解析した構文解析結果を比較することにより評価を行う．逐次解析においてチャンキングと組み合わせる構文解析器は 3 種類を選択し，それぞれの組み合わせで解析精度の評価を行う．また長単位チャンキングを行う有効性を検証するため，短単位を入力として従来の transition ベースの依存構造解析により直接短単位による依存構造を構築する方法とも比較する．短単位による依存構造解析では，長単位内部の短単位間の依存構造を *luw* という文法機能タイプを持つ通常の短単位による依存構造とみなして解析を行う．

評価指標は，2.6.2 節 と同様に式 (2.1) で計算される LAS，式 (2.2) で計算される UAS の 2 種類の attachment score を用いて解析結果の依存構造の精度を評価する．依存構造の単位の異なる長単位による依存構造と短単位による依存構造を比較するために，長単位依存構造は，各長単位を疑似的な文法機能タイプ *luw* で結合された短単位列に分割することにより短単位依存構造と単位を合わせてから比較を行った．ただし，精度を算出する際，文法機能タイプ *luw* は通常の依存構造とは区別し，長単位内部の短単位間の依存構造 *luw* を無視して長単位間の依存構造のみを計算する条件のスコア (w/o luw deps) と，全ての依存構造を同等に扱い計算する条件のスコア (all deps) の 2 種類のスコアを算出する．また，依存構造タイプの種類別の評価は，式 (2.5) で計算される F1 スコア (F_l) を用いる．

3.6.1 実験の設定

本実験では，解析モデル構築のための訓練データと手法の評価データとして 2.4 節で述べた日本語の単語依存構造コーパス KyotoDep を用いた．KyotoDep は，京都大学テキストコーパス [24] から選択された約 2 万文の新聞記事からなる長単位に基づく依存構造コーパスである．表 3.5 に訓練データ，評価データとして使用した文数，長単位数，短単位数を示す．

実験の同時解析手法には，動的計画法に基づく transition ベースの構文解析器 [51, 52] を用い 3.4 節で述べたように長単位チャンキングの操作を加える拡張をしている．本論文ではこの構文解析器を SR joint と呼ぶ．SR joint は各状態から訓練データから学習した

長単位	
$\sigma_{L0}.h_{Lw} \circ \sigma_{L1}.h_{Lw}$	$\sigma_{L0}.h_{Lt} \circ \sigma_{L1}.h_{Lw}$
$\sigma_{L0}.h_{Lt} \circ \sigma_{L1}.h_{Lt} \circ \sigma_{L0}.l_{Lt}$	$\sigma_{L0}.h_{Lt} \circ \sigma_{L1}.h_{Lt} \circ \sigma_{L0}.r_{Lt}$
$\sigma_{L0}.h_{Lw} \circ \sigma_{L1}.h_{Lw} \circ \sigma_{L2}.h_{Lw}$	$\sigma_{L0}.h_{Lt} \circ \sigma_{L1}.h_{Lt} \circ \sigma_{L2}.h_{Lt}$
$\sigma_{L0}.h_{Lt} \circ \sigma_{L1}.h_{Lt} \circ q_0.t$	$\sigma_{L0}.h_{Lw} \circ \sigma_{L1}.h_{Lt} \circ q_0.t$
$\sigma_{L0}.h_{Lt} \circ \sigma_{L1}.h_{Lw} \circ q_0.t$	$\sigma_{L0}.h_{Lw} \circ \sigma_{L1}.h_{Lw} \circ q_0.w$
複合辞	
$q_0.f_{comp} \circ \sigma_{S0}.h_{St}$	$q_0.f_{comp} \circ \sigma_{S0}.h_{St} \circ q_0.t$
$q_0.f_{comp} \circ \sigma_{S0}.h_{St} \circ q_0.t \circ q_1.t$	$q_0.f_{comp} \circ \sigma_{S0}.h_{Sw}$
$q_0.f_{comp} \circ \sigma_{S0}.h_{Sw} \circ q_0.w$	$q_0.f_{comp} \circ \sigma_{S0}.h_{Sw} \circ q_0.w \circ q_1.w$

図 3.11: 追加した特徴量テンプレート.

表 3.6: 実験に用いた解析器と組み合わせ.

依存構造の単位	解析の組み合わせ名	長単位チャンキング	依存構造解析
長単位	SR joint	SR joint	SR joint
	Coma + SR single	Coma	SR single
	Coma + Malt	Coma	Malt Parser
	Coma + MST	Coma	MST Parser
短単位	SR single	—	SR single
	Malt	—	Malt Parser
	MST	—	MST Parser

解析モデルにより次の操作（action）を決定し，図 3.8 のアルゴリズムに従って処理を行う．解析モデルで使用する特徴量は元々の構文解析器で用いたものに加えて，長単位チャンキングの操作に関して複合名詞や複合辞（機能語）についての特徴量を加えている．追加した特徴量を 図 3.11 に示す．スタック σ_L, σ_S の各要素にある部分木に関して修飾子の $.h, .l, .t$ はそれぞれ主辞，最左の子，最右の子を，添字の $_{Lw}, _{Lt}, _{Sw}, _{St}$ は，それぞれ長単位の単語出現形，品詞，短単位の単語出現形，品詞を表している．キュー Q の先頭要素 q_0 に関して修飾子 $.w, .t$ は，それぞれ短単位の単語出現形，品詞を表している．さらに各短単位について，複合辞（複数の短単位から構成される機能語）の先頭になる可能性の有無を特徴量として加えている．図 3.11 の $q_0.f_{comp}$ は，短単位 q_0 の単語出現形を先頭とする複合辞が存在するかどうかを複合語辞書との照合を行い，存在する場合 1，存在しない場合 0 とするフラグである．この特徴量により辞書に記載されるような複合辞になりやすい語句に重みをつけると同時に，複合辞の候補になり得ない語句を排除し易くすることができる．また，SR joint はビーム探索によって探索範囲を絞って解析を行うが，本論文では開発データによる実験によりビーム幅を 12 と定めた．

表 3.6 に本実験で用いた解析器と使用した組み合わせを示す．いずれの場合も入力は短

表 3.7: 構文解析結果の比較.

KyotoDep		all deps		w/o luw deps	
手法		UAS	LAS	UAS	LAS
LUW-based	SR joint	95.0	91.4	93.7	89.3
	Coma + SR single	94.9	91.3	93.5	88.9
	Coma + Malt	94.7	91.4	93.3	89.0
	Coma + MST	94.9	91.3	93.5	88.9
SUW-based	SR single	93.6	89.6	92.3	87.5
	Malt	92.9	89.2	90.9	86.7
	MST	93.5	89.4	91.8	86.9

単位列である.

逐次処理では，長単位チャンキングの解析器として Comainu (Coma) [41] を用いた．Comainu は依存構造解析とは独立に処理を行い，3 種類の依存構造解析器と組み合わせて使用した．Comainu による長単位チャンキングを行う組み合わせの手法は長単位チャンキングおよび依存構造解析の 2 段階の逐次処理を行い，残りの手法は依存構造解析 1 段階のみの処理で完了する．依存構造解析器の一つは，SR joint から長単位チャンキングに関する操作を取り除いたものであり，この解析器を SR single と呼ぶ．残りの 2 つの依存構造解析器は，SR single と同じ transition ベースの解析器である MaltParser [42] とグラフベースの MST Parser [56] を用いた．長単位チャンキングおよび長単位依存構造解析のモデルは，KyotoDep の訓練データにより構築した.

また，入力された短単位列から長単位チャンキング相当の処理を行う手法との比較のために，短単位列から直接短単位間の依存構造解析を行う解析モデルを構築した．この解析モデルは，長単位による依存構造中の各長単位を短単位間の依存構造に変換した構造を学習データとして構築した.

3.6.2 結果

表 3.6 の解析器の組み合わせにより KyotoDep の 評価文を解析した結果を表 3.7 に示す．同じ手法の中では，全依存構造を対象に計算したスコア (all deps) が長単位間の依存構造に限定したスコア (w/o luw deps) よりも高くなる傾向が出ている．これは，図 3.4 の文法機能タイプ *luw* で表されているような長単位内部の依存構造は，その他の文法機能タイプで表されている長単位間の依存構造よりも高い精度で推定できていることを示している．つまり，長単位チャンキングの方が依存構造の同定よりも高い精度で行えることを示している.

全体として長単位による依存構造解析の結果は短単位による依存構造解析の結果を上

表 3.8: 依存関係タイプのカテゴリ別の結果 (F1 スコア).

手法	述語項	連体	連用	並列
	Pred Args	Adnom	Adverb	Coord
LUW-based				
SR joint	76.6	68.5	65.4	66.5
Coma + SR single	75.9	65.9	65.3	65.9
Coma + Malt	75.3	68.2	64.6	65.7
Coma + MST	75.5	65.8	63.4	65.8
SUW-based				
SR single	74.2	63.8	60.9	63.5
Malt	73.2	63.5	58.4	59.7
MST	73.2	62.2	58.6	63.9

回っている．これは，2.6.3 節 の評価結果で見られたように，長単位の依存構造モデルで構文構造を捉えることが短単位の依存構造モデルで構文構造を捉えるよりも解析精度に有利であることと一致している．一方で長単位による依存構造解析同士を比較すると，同時解析の SR joint の結果が逐次処理の手法 (Coma+*) の結果を上回ってはいるが大きな差は生じていない．ただし，同時解析手法と逐次処理手法のスコアの差は，長単位間の依存構造のみを考慮したスコア (w/o luw deps) の方が，全ての依存構造を考慮したスコア (all deps) よりも若干大きくなっている．このことから，同時解析 SR joint の効果は長単位チャンキングの精度向上よりも，依存構造解析の精度向上に得られていると見ることができる．つまり，長単位の内部構造が誤りが改善される割合よりも，述語項構造等の内容語間のより重要な関係の誤りが改善される割合が高いと考えられる．

また，表 3.8 は各依存構造タイプのカテゴリ別の F1 スコアを表している．この結果から，同時解析手法と逐次解析処理の主要な依存構造タイプに関する結果の差は表 3.7 の全体の結果の差よりも大きいことが分かる．特に，主格 *nsubj*，対格 *dobj*，与格 *iobj* を含む述語項構造に関わるカテゴリ (Pred Args) のスコアが差が大きく，同時解析 SR joint は述語項構造や連用修飾 (Adverb) のカテゴリにおいて，逐次解析処理よりも 0.7 ポイント以上の向上を見せている．全ての依存構造を含んだ結果では小さな差であったことを考えると，主要な依存構造について同時解析の効果がより表れていると見ることができる．

表 3.9 は代表的な複合辞を含んだ依存構造のスコア LAS と UAS を表している．依存構造解析方法の複合辞の扱い方の違いが複合辞を含む依存構造の精度に反映されており，特に格助詞は述語項構造の依存構造に，接続助詞は連用修飾節の依存構造タイプにおけるスコアの差に影響を及ぼしている．すなわち，同時解析手法の SR join が逐次解析処理の SR single 単独よりも複合辞に関連する依存構造について高い精度が得られており，このことが述語項構造等の特定の依存構造タイプの精度向上に寄与していると考えられる．

表 3.9: 複合辞を含む依存構造のスコア.

複合辞	出現頻度	SR joint		SR single	
		UAS	LAS	UAS	LAS
格助詞					
に-つい-て *about*	19	89	58	84	47
と-いう *(a bird,) called (swallow)*	138	94	58	88	88
接続助詞					
と-し-て *by way of (explanation)*	83	84	74	81	74
に-よる-と *according to*	21	91	71	81	67
に-よっ-て *by*	12	92	83	75	67
に-とっ-て *(easy) to (me)*	17	82	35	82	35

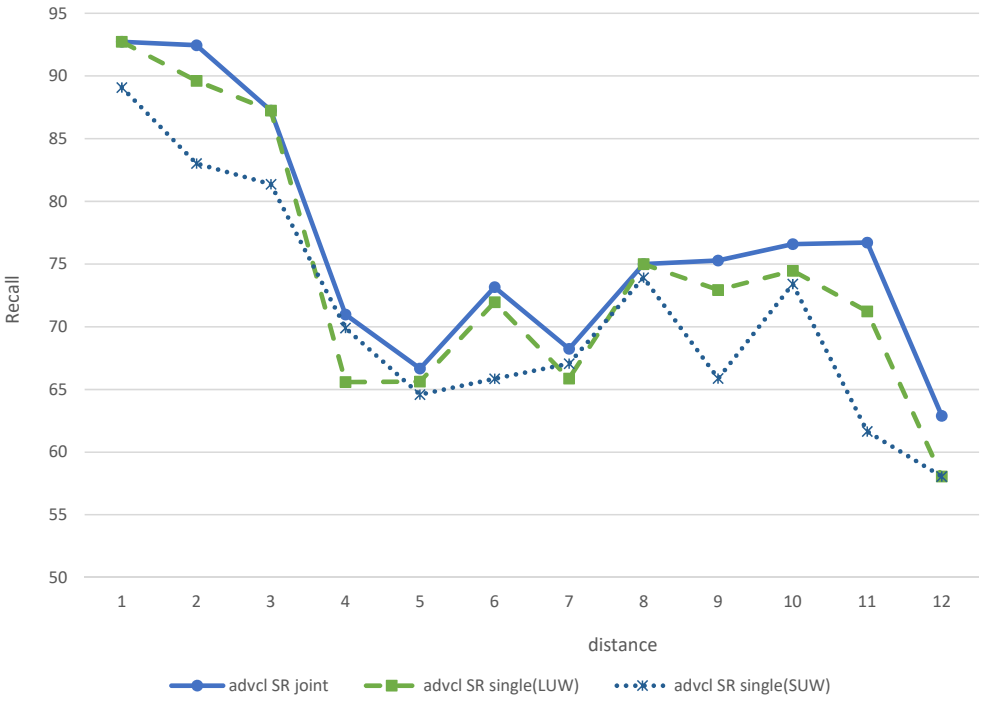


図 3.12: 連用修飾節依存関係 (*advcl*) にある単語間の距離と再現率.

また，同時解析手法は連用修飾 (advervial) と述語項構造を含む長い距離間の依存構造に関するスコアも，逐次解析処理および短単位による依存構造で解析する方法に比較して高い値を保持できることがわかる．図 3.12 は連用修飾節の関係 *advcl* を持つ依存構造について，依存構造を持つ単語間の距離とスコアの関係を表したものである．依存関係にある単語の距離が長くなるほど正しく依存構造を同定することが難しくなるが，同時解析手法 SR joint の方が，逐次解析処理の SR single (長単位による依存構造および短単位による依存構造) に比較してスコアの低下を抑えられていることが分かる．長単位チャンキングを行うことにより短単位のみで依存構造を決定するよりも長距離依存構造に頑健になり，

同時解析を行うことにより長単位チャンキングを含む全体の精度が維持できる効果が得られていると考えられる．

3.7　まとめ

本章では，日本語に単語依存構造解析を適用する際に，形態素情報を保持し安定性のある短単位と文法的機能を保持する長単位からなる階層的単語構造を採用し，長単位チャンキングを導入することにより日本語の特性を考慮した解析ができることを示した．階層的単語構造をベースとする単語依存構造解析を行う手法として，長単位チャンキングと長単位依存構造解析を同時に行う方法を提案した．長単位チャンキングと依存構造解析の同時解析手法は，逐次解析処理でチャンキングと依存構造解析を順次行う方法に比べて，複合辞等の機能語を同定する精度を向上させるとともに，格関係や連用修飾節等，重要性が高くかつ構文解析の難易度が高い依存構造のカテゴリについて良い精度を実現可能であることを示した．

第2章における長単位依存構造解析の評価では入力を長単位列としたが，実際の処理中で解析を行う際には多くの場合入力は平文あるいは形態素解析後の短単位列となることが想定される．短単位列を入力とする場合依存構造の単位であるチャンク（長単位）を同定することが必要になるが，文節以外の曖昧性の高い単位のチャンキングを行い，依存構造解析を行った場合の有効性は不明であった．本章で提案した解析手法は短単位列を入力として長単位依存構造を出力するものであり，評価結果において全体的な解析精度が従来法の精度以上を上回っていることは，第2章で提案した新たな日本語の単語依存構造の枠組みの中で構文解析が実用的な精度で実現できることを示している．さらに本章で提案した手法では，チャンキングを依存構造解析と同時に行うことにより，文全体の解析精度および述語項構造情報の取得に必要となる依存構造の解析精度を高水準で得ることができる．このことは，次章で扱う多言語間テキスト意味類似度測定のように意味解析および多言語処理を扱う処理にそれらとの親和性の高い単語構文解析を高精度で提供することに貢献している．

第4章　意味グラフに基づくテキスト意味類似度測定

4.1　はじめに

第 2 章，第 3 章ではテキストの構文構造を同定する構文解析の問題について扱ったが，本章では構文構造より抽象度の高い意味構造を扱いテキストが表す意味的な内容を比較する問題を対象とする．機械翻訳や自動要約の評価や剽窃検出など自然言語処理の主要なタスクにおいて，テキスト間の意味的な類似性 (Semantic Texutual Similarity, STS) を測定することの重要性が高まっている．評価型ワークショップ SemEval 2017 [57] の STS タスクに参加した多くのシステムは，辞書知識，単語集合 (bag of words)，単語分散表現 (word embeddings) など，表層的あるいは抽象化した語彙的知識に基づいて類似度を測定する手法を採っている．しかしながら，これらの手法は文同士を比較した際に，意味内容として類似している点と異なっている点を明示的に説明することができないという問題がある．例えば,「犬が猫に追われた」と「犬に猫が追われた」という 2 文の比較を考えると，文字列としては 2 文字異なるのみで単語の集合としては同一である．構造的な情報を考慮しない手法では，このように表層的な類似性が高い表現の間の意味内容の違いを検出することは難しく，比較対象の文の構造を考慮した類似点，相違点は扱われてこなかった．また，機械翻訳の評価においてもテキストの類似性を測定する技術が用いられているが，基本的には評価対象テキストと正解の翻訳テキスト（参照訳）とを比較し，表層的な文字列の類似性が高いものに高い評価値を与えるものである．最も基本的かつ代表的な自動評価尺度である BLEU [58] は，システムの出力文と正解の翻訳文の間で単語 n-gram が一致する割合に基づいて算出される．BLEU は，人間の主観評価との相関が高いとされているが，先程の例のように僅かな文字が異なるだけで文意が全く異なるような場合には対応できない．

このような表層的な情報のみに依らずテキストの表す意味内容を比較するために，テキストを意味解析した結果の意味構造を用いる方法がある．有向グラフにより表現された意味構造である意味グラフの代表的なものとして Abstract Meaning Representation (AMR) [59] や Scene Graph [60] がある．また，これらの意味グラフ同士の比較方法として，AMR を対象とする Smatch [61]，Scene Graph を対象とする Semantic Propositional Image Caption Evaluation (SPICE) [62] がそれぞれ提案されている．これらの方法は，意味グ

ラフ間の重なりを計測することで意味構造を考慮した類似度スコアを算出する．意味構造を用いることの利点は表層に現れない情報を扱えるようになることであり，テキストで表現されている概念間の関係を捉えた比較が可能になる．例えば前述の例では,「追う」という動作の「動作主」や「対象」がそれぞれ何であるかという内容の違いを区別することができる．一方で，意味解析は構文解析に比較して解析の難易度が高く，類似度の算出が意味解析の精度に依存していると多様な文体のテキストに対して頑健に結果を算出することが難しい．

テキストの表す意味内容の比較に意味構造を用いるもう一つの利点は，構文構造に比べて対象とする言語への依存度が低いため異なる言語間のテキストの比較に適用しやすい点である．SemEval 2017 のシェアードタスクでも見られるように，異なる言語のテキストを比較対象にした言語横断のテキスト意味類似度 (Crosslingual STS, CL STS) の測定を機械翻訳の評価や言語横断情報検索に適用することも期待されている．しかし，AMR や SPICE は主に英語を中心として考えられてきた意味表現であり，日本語を含む他の言語に対応させるためには入力文から意味表現への変換方法を言語個別に考えなければならない[1]．

本章では，個々の言語への依存度の低い意味グラフの比較に基づいて，日本語を含む多言語に適用可能なテキストの意味類似度を測定する方法について提案する．幅広い言語への対応を可能とするために，多言語共通の構文解析の枠組みである UD と UD に基づいて構築可能な意味グラフ UDepLambda [63] を用いて，同一言語のテキストの比較と同様の方法で多言語間のテキスト比較が可能な方法を提案する．

以下，4.2 節で多言語対応のテキストの意味類似度測定方法の基本的な方針を述べ，4.3 節でいくつかの意味グラフおよび複数の意味グラフ間の比較方法について概観し，4.4 節で多言語に対応した意味グラフ UDepLambda を用いたテキスト意味類似度の測定方法について述べる．4.5 節で関連研究について述べた後，4.6 節で STS のベンチマークデータおよび機械翻訳の評価データを使って，人間が評価した類似度との比較を行うことにより提案手法が算出した類似度の妥当性の評価を行う．最後に 4.7 節で本章の成果についてまとめる．

4.2 構造情報を用いた言語横断テキスト意味類似測定の方針

本論文で提案する方法では，それぞれの意味表現間の対応付けを行うことを基本として 2 文間の類似度を測定する．比較する言語への依存度を低減するため，意味表現は多言語への対応のしやすさが考慮されている UDepLambda を用いる．UDepLambda は構文解析アノテーションの多言語共通仕様である UD を基盤としており，多くの言語への拡張が

[1]AMR については，ドイツ語，スペイン語，中国語など英語以外の言語でのアノテーションが存在する．

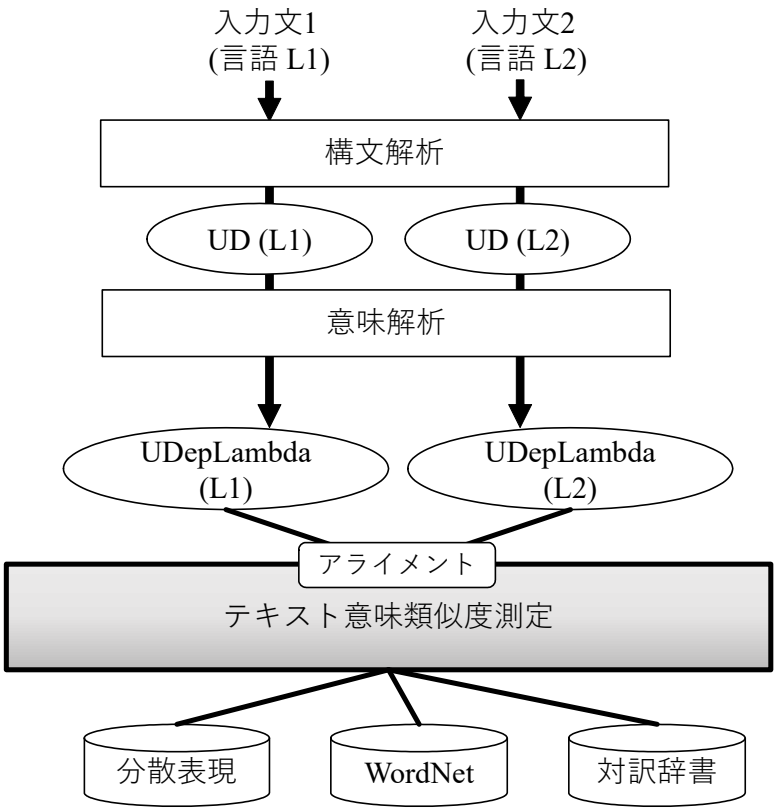


図 4.1: 提案する言語横断テキスト意味類似度測定方法の概略図.

比較的容易である.

図 4.1 に提案するテキスト意味類似度測定方法の概略図を示す．提案手法では対象となる文を単に文字列として比較するのではなく，言語共通の枠組みである構文解析 (UD) および意味解析 (UDepLambda) を行い意味表現に抽象化して比較する．言語間で共通の意味表現の対応付け（アライメント）を行うことにより，多様な言語表現の差異を吸収して類似度測定を行うことができると考えられる．中間的な表現である意味表現は言語共通の構文解析の枠組みから変換可能なため多くの言語に対応可能であり，異なる言語で書かれた文を比較する場合であっても，同一言語で書かれた文同士を比較する場合と同様の方法で測定することができるという利点がある．また，比較する文の間で類似点と相違点を意味表現間のアライメントの結果として提示することが可能である．ただし，意味表現のアライメントのみでは 2 文の類似の度合いを定量的な値として測定することが難しいため，出現単語の重複度合いや n-gram の重複度合いなど従来の類似度測定方法で用いられる情報で補完して類似度スコアを算出する．すなわち，本論文の方法は意味グラフのアライメント結果と様々な特徴量から算出される類似度スコアを得ることができると想定している.

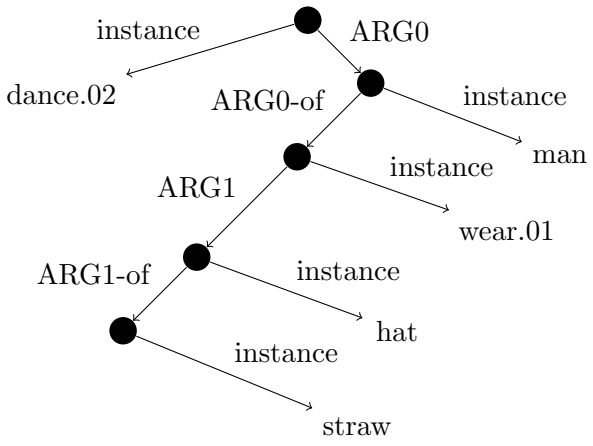


図 4.2: 有向グラフによる AMR の意味表現の表示例.

Frame:	wear.01	Frame:	wear.06
Description:	simple dressing	Description:	erode, diminish something
ARG0:	*wearer* (Agent)	ARG0:	*cause of erosion* (Agent)
ARG1:	*clothing* (Theme)	ARG1:	*thing eroded, worn* (Theme)
		ARG2:	*source location* (Source)

図 4.3: PropBank の概念フレームの例.

4.3 意味グラフと比較方法

本節では，代表的な意味グラフによる意味表現とその比較手法について概観する．

4.3.1 AMR

AMR による意味表現では，対象文から意味を持った概念を抽出し概念間の関係をグラフ構造で表現する．このグラフは，単一の根ノードを持ちエッジと葉ノードにラベルを持つ有向グラフである．AMR には，PropBank [64] で定義された概念フレームと意味役割の情報と，同一文内での共参照情報，固有表現とそのタイプ，モダリティ，否定，疑問，量化子の情報が含まれる．葉ノードはラベル付きの概念を表し，中間ノードはエンティティ，イベント，属性，状態を表す変項に対応する．エッジはノード間の関係に対応し，各エッジには PropBank で定義されている意味役割ラベルが付加される．

図 4.2 は，“A man wearing a straw hat is dancing.” という文の AMR を意味グラフにより表示した例である．意味グラフにおいて各黒丸は変項を表す中間ノードであり，instance のラベル付きエッジの先に記述されている “man” や “wear.01” が葉ノードに付加されている概念のラベルである．概念のラベルのうち，“dance.02” や “wear.01” のようにアルファベットにピリオドと数字を付加したものが概念フレームを表しており，PropBank で定義されている述語の持つ概念を表す．概念フレームに対応する概念からは “ARG0”,

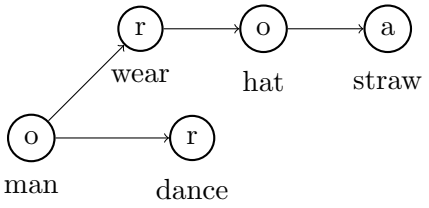


図 4.4: Scene Graph の例.

“ARG1” のようなラベル付きのエッジが伸びており，ラベルが意味役割，エッジの先が項となる概念を示している．図の例では，概念フレーム “wear.01” に対して “ARG1” のラベル付きのエッジが “hat” を示す概念を指しており，“wear.01” という「服等を着る」という概念フレームの「対象」(ARG1) が “hat” という概念であることを示している．図 4.3 は，PropBank で定義されている “wear” の概念フレームの例である．同じ表記の単語に対して語義により異なる複数の概念フレームが定義される可能性があり，”wear” の場合 “wear.01” の他に “wear.06” という「すり減らす，使い古す」に対応するフレーム等がある．図 4.2 の例の AMR 中の一つのノードが “wear.01” のフレームと対応している．

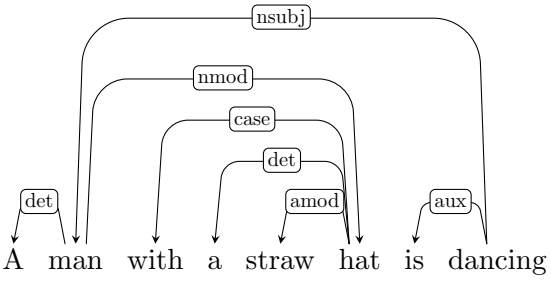
2 つの AMR を比較する方法として，Smatch [61] が提案されている．この方法では，AMR における意味グラフを $(relation, variable, concept)$ または $(relation, variable_1, variable_2)$ の 2 種類の triple に分解して扱う．AMR の意味グラフを比較するとき，Smatch は比較対象となる意味グラフそれぞれから抽出された triple の集合を比較して，適合率，再現率，F1 スコアを計測する．Smatch によるスコアは，比較する一対の意味グラフに対して含まれる変項の 1 対 1 の対応を取った時の最大の F1 スコアとして定義される．このとき，2 変項間の対応の強さは無視され，個々の変項間の対応スコアは 1 または 0 である．

4.3.2 Scene Graph

Scene Graph は，主に画像のキャプションが表現している意味を表すために考案された有向グラフであり，エンティティ（オブジェクト），属性，関係それぞれを object タイプ，attribute タイプ，relation タイプのノードとして表現する[2]．AMR における意味グラフでは関係はノード間のエッジで表現されるが，Scene Graph では関係もノードで表現されておりノード間のエッジにはラベルが付加されない．図 4.4 の例は，前節と同じ文 “ A man wearing a straw hat is dancing.” の Scene Graph による表現である．“man” と “hat” は object タイプ (記号 “o” で表記) のノードである．“hat” と “man” のノードは relation タイプ (記号 “r” で表記) の “wear” というノードで接続されている．また，“hat” の object タイプのノードには attribute タイプ (記号 “a” で表記) の “straw” というノード が接続されている．

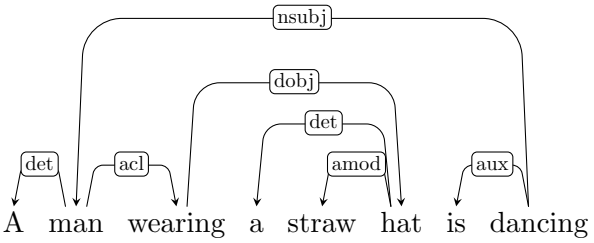
[2]Stanford coreNLP[60] <https://stanfordnlp.github.io/CoreNLP/> の出力として得ることができる．

文 A



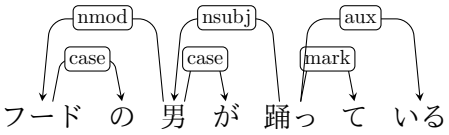
$\lambda x.\exists yz\ \mathrm{dance}(x_e) \wedge \mathrm{man}(y_a) \wedge \mathrm{hat}(z_a)$
$\wedge \mathrm{straw}(v_a)$
$\wedge \mathrm{arg}_1(x_e, y_a) \wedge \mathrm{nmod.with}(y_a, z_a)$

文 B



$\lambda x.\exists yzw\ \mathrm{dance}(x_e) \wedge \mathrm{man}(y_a) \wedge \mathrm{hat}(z_a)$
$\wedge \mathrm{wear}(w_e) \wedge \mathrm{straw}(v_a)$
$\wedge \mathrm{arg}_1(w_e, y_a) \wedge \mathrm{arg}_2(w_e, z_a) \wedge \mathrm{arg}_1(x_e, y_a)$

文 C



$\lambda x.\exists yz$ 踊る $(x_e) \wedge$男 $(y_a) \wedge$フード (z_a)
$\wedge \mathrm{arg}_1(x_e, y_a) \wedge \mathrm{nmod.no}(y_a, z_a)$

図 4.5: 対象となる英文（文 A）と類似した英日 2 文 (文 B, C) の UD による構文構造と UDepLambda による意味構造の例.

2 つの Scene Graph の比較による画像のキャプションの評価方法として，SPICE [62] が提案されている．SPICE は，ノードとノード間の連結の仕方の組み合わせを 3 種類の tuple として捉えて比較する．その 3 種類の tuple とは，(object), (object, attribute), (relation, object_1, object_2) である．類似度は，比較対象である意味表現 Scene Graph から tuple 間の重複度合いを F1 スコアとして計算し類似度スコアとする．エンティティ間の関係 (relation) は，subject-object, color, size などのいくつかのタイプに分類される．

4.3.3 UDepLambda

UDepLambda は， 多言語共通の構文構造の枠組みである UD に基づいて定義されている論理式形式の意味表現であり，変項[3]をノード，変項間の関係をエッジに対応させることで有向グラフとして扱うことができる．UDepLambda は多言語対応可能な意味表現であ

[3]本論文では，1 階述語論理の個体変項 (individual variable) を指し事物 (object) が入る．

るが，その比較方法については提案されていない．図 4.5 は，対象となる英語 1 文と類似した英語の文，日本語の文 1 文ずつそれぞれについて，UD による構文構造と UDepLambda による意味構造を表示した例である．UDepLambda は，依存構造で表された UD の構文構造を論理式に変換することにより構築されており，原理的には UD 形式で構文解析が可能な言語であれば共通の意味表現を獲得可能である[4]．すなわち，UD が対応している（UD の仕様が定義されている）多くの言語に対して適用可能であることが，UD に基づく意味表現を用いる利点である．ただし，Reddy らにより英語の他にドイツ語とスペイン語の解析モデルは構築されているが，インド・ヨーロッパ語族以外の言語的特性の大きく異なる言語の解析モデルは構築されておらず，日本語への UDepLambda 適用の有効性は検証する必要がある．

4.4 多言語対応意味グラフを用いたテキスト意味類似度測定

4.4.1 意味解析モデルの日本語対応

提案手法は意味グラフとして UDepLambda を用いるが，4.3.3 節で述べたように意味解析モデルは英語，ドイツ語等の限られた言語についてのみ構築されており，日本語のモデルは構築されていない．本論文では，日本語の UD の構文木（依存構造）から lambda 式に変換する規則を追加することにより日本語のモデルを構築し，UDepLambda に基づく STS の日本語への有効性を検証する．本論文で構築した日本語モデルは UD からの基本的な変換のみに対応したものであるが，UD の依存構造タイプを変換することにより述語項構造のような基本的な関係は捉えられるため，多言語対応の実現可能性の検証を十分行えると考える．また，英語，ドイツ語等とは語順の大きく異なる日本語における有効性の検証は，新しい言語を追加して多言語対応を拡大する実現可能性を検証する上でも大きな要素になると考える．

4.4.2 意味グラフベースの指標と単語・単語列ベースの指標

本論文で提案する方法は，比較する 2 文をそれぞれ意味解析して得られた意味グラフ間のアライメントを行い，アライメントされた度合いを基本としてテキストの類似度を測定する．前述のように日本語を含めた幅広い言語のテキストに適用可能とするため，多言語共通の構文解析の枠組み UD を基盤とした UDepLambda による意味グラフを採用する．UDepLambda は同一の意味を表現するのに用いられる構文の多様性を吸収して，意

[4]文が表している意味内容をより厳密に捉えようとすれば，個別の言語毎に対応すべき現象は存在するが，UD の持つ文法機能ラベルは各言語で表現される述語項構造の関係を区別することができるため，その文法ラベルを直接変換した意味表現により主要な意味内容を捉えられると考えられる．

表 4.1: 文の類似度測定に使用した類似度の指標の組み合わせ．

類似度の指標	出力		対象言語	
	アライメント結果	スコア	単言語	多言語
意味グラフベースの指標				
アライメント	✓	σ_{align}	✓	✓
重複度合い		σ_{ov}	✓	✓
単語・単語列ベースの指標				
単語表層：重複度合い		ϕ_1, ϕ_2	✓	✓
synset：重複度合い		ϕ_3	✓	✓
n-gram：重複度合い (n = 1,2,3)		ϕ_4, ϕ_5, ϕ_6	✓	

味内容の比較をするために有用である．例えば，文 “a man wears a hat” の構文木では，“wear” と “hat” の間に依存構造タイプ *dobj* (述語－目的格) の関係を持つが，文 “the hat that a man wears” の構文木では “wear” と “hat” の間に依存構造タイプ *acl* (連体修飾節) の関係を持ち両者は一致しない．しかし，UDepLambda ではどちらの文でも 2 つの概念間は同じ関係 arg_2 (動作－対象) を持ち，類似度を計算する上で重要な共通点を捉えることができる．UDepLambda を用いる別の利点としては，構文木に直接現れない関係が補完されることによって，より適切な比較が可能になることである．例えば，文 “the children want to swim” の意味解析結果からは，構文的に直接依存関係のある “children” と “want” の間の tuple $(arg_1, \text{want}, \text{children})$ だけでなく，“want” を介した関係も tuple $(arg_1, \text{swim}, \text{children})$ として抽出される．

提案手法では意味グラフの対応付けを行う際に，アライメントの対応度合いによって算出されるアライメントスコアが最も高くなるアライメント（最適なアライメント）を求める．アライメントスコアは，後述するように比較対象の意味グラフを tuple に変換した後，対応付けられた tuple の間のマッチングスコアに基づいて算出する．最適なアライメントは，ランダムに定めた初期アライメントから始めて最もアライメントスコアが高くなる意味グラフのエッジとノードの組み合わせを山登り法を用いて探索して求める．山登り法で探索を行うため局所的にアライメントスコアが高いアライメントが解になる場合があり得るが，複数の初期値を試行することにより実際の文のアライメントで局所最適解に陥るリスクを軽減できると考えられることと，計算量的に有利な点からこの方法を採用する．また，文全体の類似度スコアを算出するためにアライメントスコア以外に表 4.1 に示すような意味グラフの重複度合いによる指標のスコアや単語・単語列ベースの指標のスコアを用いる．これらのスコアを組み合わせて文間の類似度測定モデルを構築し，2 文全体を比較した類似度スコアを算出する．これらの指標の多くは単言語，多言語どちらの場合にも適用可能であるが，n-gram のような表層的情報は異なる言語間で比較してもあまり意味がないことから多言語には適用を行わない．

以下では，4.4.3 節で UDepLambda による意味表現から tuple への変換について，4.4.4

節で 2 つの tuple 間のマッチングスコアについて述べる．また，4.4.5 節で意味グラフに基づくスコアの算出方法について，4.4.6 節で単語・単語列に基づくスコアの算出方法について述べ，4.4.7 節でこれらのスコアを用いた文全体の類似度スコアの算出方法について述べる．

4.4.3 UDepLambda から tuple への変換

提案方法では，2 つの意味グラフ間のアライメントや重なり度合いに基づくスコアを計算するために，最初に UDepLambda の意味表現を 3 項の tuple の集合へ変換する．図 4.5 の例にあるように，UDepLambda は predicate(argument) の形で表記される述語 1 つと項 1 つの要素と predicate(argument1, argument2) の形で表記される述語 1 つと項 2 つの要素からなっている．前者の要素は predicate がインスタンス（entity または event)，argument が変項と対応する形式で用いられる．この要素は変項とインスタンスを結びつけることを表しており，これを tuple (**inst**, 変項, インスタンス) に変換する．**inst** は，変項とインスタンスを結びづける関係を表すラベルである．後者の要素は predicate が関係，argument1, argument2 がそれぞれ変項 1，変項 2 に対応する形式である．この要素は，変項 1 と変項 2 の間の関係を表しており，これを tuple (関係, 変項 1, 変項 2) に変換する．

図 4.6 は，図 4.5 の 3 文の UDepLambda による意味表現を tuple に変換した例を示している．変項の存在している元の文が区別できるように，各変項の右肩に元の文を表す記号 A, B ,C を付加している．上に述べたように tuple には，変項とインスタンスの関係を表す tuple と，変項間の関係を表す tuple の 2 種類がある．例えば，tuple $(\mathbf{inst}, y^A, \mathrm{man})$, $(\mathbf{inst}, z^A, \mathrm{hat})$ は，それぞれ変項 y^A に “man” が，変項 z^A に “hat” が対応することを示している．また，変項の間の関係を表している tuple ($nmod.with$, y^A, z^A) にこれらのインスタンスを代入すると，tuple ($nmod.with$, man, hat) が得られる．

UD および UDepLambda はともに内容語を主辞とする依存構造を原則とするため，述語項構造のように内容語が直接関係している構造が，前置詞や助詞等の機能語が介在し内容語が直接関係しない構造と異なるものとして扱われる可能性がある．例えば， 図 4.5 の上から 文 A，文 B それぞれに含まれる “man” と “hat” は，2 文において同じ意味関係を持っている．文 A からは tuple $(nmod.with, \mathrm{man}, \mathrm{hat})$ のみが抽出され，文 B からは 2 つの tuple $(arg_1, \mathrm{wear}, \mathrm{man})$ と tuple $(arg_2, \mathrm{wear}, \mathrm{hat})$ が抽出されるが，“man” と “hat” の関係を直接比較することができない．この違いを補完するために，本論文では同じ主辞を共有する 2 つの tuple について，新しい tuple を補完することを行う．例えば上の例では，文 B において “wear” を共通の主辞とする 2 つの tuple を合成した新しい tuple $(rel*, \mathrm{man}, \mathrm{hat})$ を追加して，文 A との比較を可能にする．ここで，$rel*$ は合成した tuple

文 A	文 B	文 C
(**inst**, x^A, dance)	(**inst**, x^B, dance)	(**inst**, x^C, 踊る)
(**inst**, y^A, man)	(**inst**, y^B, man)	(**inst**, y^C, 男)
(**inst**, z^A, hat)	(**inst**, z^B, hat)	(**inst**, z^C, フード)
	(**inst**, w^B, wear)	
(**inst**, v^A, straw)	(**inst**, v^B, straw)	
	(arg_1, w^B, y^B)	
	(arg_2, w^B, z^B)	
(arg_1, x^A, y^A)	(arg_1, x^B, y^B)	(arg_1, x^C, y^C)
($nmod.with$, y^A, z^A)	($rel*$, y^B, z^B)	($nmod.no$, y^C, z^C)

図 4.6: 図 4.5 の UDepLambda から変換した tuple の例.

に追加した関係であり，他の任意の関係と対応づけることができる．

4.4.4 2 つの tuple 間のマッチングスコア

本論文の提案方法では Smatch による AMR の比較の場合と異なり，意味グラフの対応付け時に変項（ノード）同士の対応付けを行い，変項間の対応度合いを表すマッチングスコアを足し合わせたアライメントスコアを算出する．変項間のマッチングスコアは 0 から 1 の実数値である．この手順により，変項で示されている概念間の類似度を捉えて意味グラフ間の類似度に反映させることができる．例えば，“he runs a street” という文に対して，同じ述語 “run” が現れる 2 文 “Tom ran track” と “he runs a hospital” それぞれとの類似度を比較するとき，“run” の対象 (ARG1) となる “track” あるいは “hospital” それぞれと “street” との概念の類似度を意味グラフのアライメントスコアに反映させることができる．

2 つの意味表現をアライメントする際に，実際には前節で述べたように変換した tuple 同士を比較し，2 つの tuple 間の一致度合いを表す値としてマッチングスコアを用いる．$t_{Ak} = (r_{Ak}, v_{Ap}, v_{Aq}) \in T_A$ と $t_{Bl} = (r_{Bl}, v_{Br}, v_{Bs}) \in T_B$ の tuple 間のマッチングスコア $\sigma_T(\cdot,\cdot)$ は，式 (4.1) で計算する．

$$\sigma_T(t_{Ak}, t_{Bl}) = \begin{cases} sim(I(v_{Ap}), I(v_{Br})) & \text{if } (r_{Ak} = r_{Bl} = \textbf{inst}) \\ sim(I(v_{Aq}), I(v_{Bs})) \cdot sim(I(v_{Ap}), I(v_{Br})) & \\ & \text{if } (r_{Ak} = r_{Bl} \text{ or} \\ & \ r_{Ak} = rel* \text{ or} \\ & \ r_{Bl} = rel*) \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

ここで，$I(\cdot)$ は変項からインスタンスへの写像である．すなわち，$(\mathbf{inst}, v_{Ai}, e) \in T_A$ のとき $I(v_{Ai}) = e$ である．インスタンス間の類似度 $sim(\cdot,\cdot)$ は，2 つのインスタンスが同一文字列の場合は 1 とする．異なる言語のテキスト間を比較する場合は，各インスタンスの文字列は基本的には一致しないため，対訳辞書を用いて対訳関係のある 2 語（例えば，船と ship）について最大の類似度 1 を与える．同一の言語間，異なる言語間いずれにおいても，比較するインスタンスの文字列が一致せず，対訳辞書で対訳関係が認定できない場合は，オントロジや単語分散表現を用いて各インスタンスを抽象化して概念間の類似度を用いる．オントロジとは，単語で表された表現を概念に抽象化して扱うために用いられる言語資源で，概念の定義や概念間の関係が記述されており概念間の対応付けを行うのに適している．単語分散表現は，各単語を類似した文脈で用いられるものが近傍に配置されるようにベクトル空間に写像した表現であり，これも概念間の対応付けに用いることができる．

本論文では，インスタンス e_1 と e_2 の間の類似度の算出方法として，WordNet [65] を用いたオントロジに基づく類似度 $sim_{wn}(e_1, e_2)$ と word2vec [66] や GloVe [67] 等の単語分散表現を使った単語ベクトルに基づく類似度 $sim_{vec}(e_1, e_2)$ の 2 種類を考える．WordNet は，synset と呼ばれる同義語の集合を上位－下位関係等の関係を表すリンクで接続したオントロジである．WordNet に基づく類似度 $sim_{wn}(e_1, e_2)$ は，インスタンス e_1，e_2 の属するそれぞれの synset c_1, c_2 を用いて式 (4.2) で計算する [68].

$$sim_{wn}(e_1, e_2) = \frac{2 \cdot \mathrm{depth}(\mathrm{lso}(c_1, c_2))}{\mathrm{pathlen}(c_1, c_2) + 2 \cdot \mathrm{depth}(\mathrm{lso}(c_1, c_2))} \tag{4.2}$$

ここで，$\mathrm{lso}(c_1, c_2)$ は synset c_1, c_2 の上位－下位関係の共通の親のうち最も深い synset であり，$\mathrm{pathlen}(c_1, c_2)$ は WordNet 上で c_1 から c_2 へ辿る最短経路の長さ，$\mathrm{depth}(\cdot)$ は synset の WordNet 上での深さを表す．

また，単語分散表現を使った単語ベクトルに基づく類似度 $sim_{vec}(e_1, e_2)$ は，インスタンス e_1, e_2 の単語ベクトルをそれぞれ $\boldsymbol{v_{e1}}$, $\boldsymbol{v_{e2}}$ とすると，式 (4.3) で表されるコサイン類似度 $sim_{cos}(e_1, e_2)$ を用いて式 (4.4) で表される．

$$sim_{cos}(e_1, e_2) = cos\frac{\boldsymbol{v_{e1}} \cdot \boldsymbol{v_{e2}}}{|\boldsymbol{v_{e1}}||\boldsymbol{v_{e2}}|} \tag{4.3}$$

$$sim_{vec}(e_1, e_2) = \begin{cases} sim_{cos}(e_1, e_2) & (sim_{cos}(e_1, e_2) \geq 0) \\ 0 & (sim_{cos}(e_1, e_2) < 0) \end{cases} \tag{4.4}$$

2 つのインスタンス間の類似度 $sim_{vec}(e_1, e_2)$ は非負の値とし，コサイン類似度が 0 以下になる場合は $sim_{vec}(e_1, e_2) = 0$ とする．多言語間のマッチングスコアに使う単語類似度については 4.4.8 節 で述べる．また，単語分散表現として何を用いるのが適当であるかは 4.6.1 節で評価する．

4.4.5 意味グラフに基づくスコア

対象とする文の意味表現である UDepLambda をそれぞれ tuple 集合に変換して，Smatch や SPICE と類似した方法でこれらの集合を比較する．本手法の類似度計算では，次の 2 種類の tuple の比較方法を組み合わせる．一つは Smatch で行われているような tuple 間のアライメントスコア，もう一つは SPICE で行われているような tuple 間の重複度合いを表す F1 スコアである．以下で 2 つのスコアの算出方法について述べる．

意味グラフのアライメントスコア

アライメントのアルゴリズム　2 つの意味グラフの間でアライメントスコアが最大となるようなアライメントを求め，その最大のアライメントスコアを用いる．2 つの意味グラフ間のアライメントをとるアルゴリズムを図 4.7 に示す．本節の説明では，比較する 2 文を S_A, S_B，それぞれの文を意味解析して得られた UDepLambda による意味表現をそれぞれ U_A, U_B とする．本アルゴリズムでは最初に U_A と U_B から tuple を抽出するが，それぞれの tuple 集合 T_A と T_B とする．アライメントは，ランダムな初期値から出発してより高いアライメントスコアが獲得できるアライメントを山登り法で探索していく．意味グラフから tuple への分解と tuple 間のアライメントは次のような手順で行う．括弧内の行番号は，図 4.7 に示すアルゴリズムの擬似コードの対応する行番号を示している．

1. 意味グラフのノードから各変項 v を抽出する (7 行目)．UDepLambda による意味表現 U_A と U_B から抽出した変項の集合 V_A と V_B をそれぞれ $V_A = \{v_{A1}, ..., v_{Am}\}$, $V_B = \{v_{B1}, ..., v_{Bn}\}$ $(m \leq n)$ と定義する．

2. 変項 v とインスタンス e (entity または event) を組にした tuple $(\mathbf{inst}, v, e)$ を tuple 集合に追加する (8 行目)．**inst** は対応する変項とインスタンスの関係を表す．文 S_A と S_B から抽出した tuple をそれぞれ tuple 集合 T_A と T_B に追加する．

3. 変項 v_i, v_j と関係 r を抽出し，tuple (r, v_i, v_j) を tuple 集合に追加する (9 行目)．

4. 主辞 が同一の変項 v_i を含む複数の tuple $(r_h, v_i, v_j), (r_k, v_i, v_l)$ について，2 つの変項 v_j, v_l 間の関係を含む tuple に縮約した tuple $(rel*, v_j, v_l)$ を新たに追加する (10 行目)．

5. 文 S_A，S_B から抽出した変項の集合 V_A と V_B の間の初期アライメント $A_0 = (a_1, ..., a_m)$ をランダムに決定する (16 行目)．アライメント候補集合 C，探索済みアライメント候補集合 E，アライメントスコアの最大値 S_{max}，最大のアライメントスコアを持つアライメント A_{best} を初期化し，C に初期アライメント A_0 を加える (17-20 行目)．ここで， $a_i = j$ は 変項 v_{Ai} と v_{Bj} がアライメントされてい

ることを示す．また，$a_i = 0$ は v_{Ai} がどの変項にもアライメントされていないことを示す[5]．

6. アライメント候補集合 C 中の最大アライメントスコア S_{max0}，最良アライメント A_{best0} を初期化する (22 行目)．

7. アライメント候補集合 C の中の全てのアライメント A_c について以下を行う (24-29 行目)．

   (a) アライメントスコア $S_c = \sigma_{align}(A_c)$ を計算する (25 行目)．

   (b) アライメント A_c を探索済みアライメントリスト E に追加する (26 行目)．

   (c) もし，$S_c > S_{max0}$ ならば $A_{best0} = A_c$， $S_{max0} = S_c$ として更新する．

8. もし，$S_{max} < S_{max0}$ であれば，$A_{best} = A_{best0}$， $S_{max} = S_{max0}$ として更新する (30-31 行目)．さらに，以下の手順により新たなアライメント候補を作り，探索済みアライメントリスト E に含まれていない全てのアライメント候補を次のアライメント候補集合 C_{next} に入れる (32 行目)．

   (a) A_{best} の 1 つの対応付けを他の変項への対応付けに変更する．

   (b) A_{best} の 2 つの対応付けを入れ替える．

9. 現在のアライメント候補集合 C を C_{next} に置き換える (33 行目)．アライメント候補集合 C にアライメント候補が存在するならば 6 へ戻る．

10. A_{best}, S_{max} を最終的なアライメントおよびアライメントスコアとして出力する (35 行目)．

上記のアライメントスコア $\sigma_{align}(\cdot)$ は，4.4.4 節 で述べた各 tuple 間のマッチングスコア $\sigma_T(t_A, t_B)$ を式 (4.5) のように足し合わせることにより計算する．

$$\sigma_{align}(A_k; V_A, V_B, T_A, T_B) = \frac{2}{|T_A| + |T_B|} \sum_{\{k,l|a_k=l\}} \sigma_T(t_{Ak}, t_{Bl}) \tag{4.5}$$

実例によるアライメントアルゴリズムの説明　以下では，本節の意味グラフのアライメントアルゴリズムを図 4.5 および図 4.6 の例を使って説明する．文 A を “A man with a straw hat is dancing.”, 文 B を “A man wearing a straw hat is dancing.” とする．説明を簡単にするためインスタンス間の類似度 $sim(e_1, e_2)$ は，同一文字列の場合 1，それ以外の場合 0 とし, オントロジに基づく類似度 $sim_{wn}(e_1, e_2)$ および単語ベクトルに基づく類似度 $sim_{vec}(e_1, e_2)$ は考慮しない．

[5]本論文におけるアライメントは，1 対 1 対応あるいは 1 対 0 の対応かのどちらかである．

Algorithm 3: Semantic graph alignment

```
# UDepLambda 1 組のアライメント
01  function alignSemGraphs (UDepL U_A, UDepL U_B):
02    Set<Tuple> T_A = convertUDepL2Tuples (U_A)
03    Set<Tuple> T_B = convertUDepL2Tuples (U_B)
04    (Score S_max, Align A_best) = alignTuples (T_A, T_B)

# UDepLambda から tuple 集合への変換
05  function convertUDepL2Tuples (UDepL U):
06    Set<Tuple> T = {}
07    Set<Variable> V = extractVariables (U)
08    Set<Tuple> T_inst = composeInstTuples (V, U)
09    Set<Tuple> T_rel = composeRelTuples (V, U)
10    Set<Tuple> T_ex = addExtendedTuples (T_rel)
11    T.add(T_inst)
12    T.add(T_rel)
13    T.add(T_ex)
14  return T

# 2 文から抽出した tuple 集合間のアライメント
15  function alignTuples (Set<Tuple> T_A, Set<Tuple> T_B):
16   Align A_0 = getInitAlignment (T_A, T_B)
17   Set<Align> C = {}
18   Set<Align> E = {}
19   C.add(A_0)
20   Score S_max = 0; Align A_best = 0
21   while (C is not null):
22     Score S_max0 = 0; Align A_best0 = 0
23     Set<Align> C_next = {}
24     foreach A_c in C:
25       Score S_c = calcAlignmentScore (A_c, T_A, T_B)
26       E.add(A_c)
27       if (S_max0 < S_c) :
28         S_max0 = S_c; A_best0 = A_c
29     end
30     if (S_max < S_max0) :
31       S_max = S_max0; A_best = A_best0
32       C_next = createNextCandList (A_best, E)
33     C = C_next
34   end
35   return (S_max, A_best)
```

図 4.7: 意味グラフアライメントのアルゴリズム.

1. 初めに，文 A，B それぞれの UDepLambda U_A, U_B から，変項の集合 V_A, V_B を抽出する．

$$V_A = \{x^A, y^A, z^A, v^A\} \tag{4.6}$$
$$V_B = \{x^B, y^B, z^B, w^B, v^B\} \tag{4.7}$$

2. tuple 集合 T_A. T_B を初期化した後，変項とインスタンスを組にした tuple $(\mathbf{inst}, v, e)$ を tuple 集合 T_A, T_B に追加する．

$$T_A = \{(\mathbf{inst}, x^A, \text{dance}), (\mathbf{inst}, y^A, \text{man}), (\mathbf{inst}, z^A, \text{hat}), (\mathbf{inst}, v^A, \text{straw})\} \tag{4.8}$$
$$T_B = \{(\mathbf{inst}, x^B, \text{dance}), (\mathbf{inst}, y^B, \text{man}), (\mathbf{inst}, z^B, \text{hat}), (\mathbf{inst}, w^B, \text{wear}), (\mathbf{inst}, v^B, \text{straw})\} \tag{4.9}$$

3. 変項 v_i, v_j とその間の関係 r を抽出し， tuple (r, v_i, v_j) を tuple 集合 T_A, T_B に追加する．

$$T_A \leftarrow T_A \cup \{(arg_1, x^A, y^A), (nmod.with, y^A, z^A)\} \tag{4.10}$$
$$T_B \leftarrow T_B \cup \{(arg_1, w^B, y^B), (arg_2, w^B, z^B), (arg_1, x^B, y^B)\} \tag{4.11}$$

4. 主辞が同一の変項を含む複数の tuple $(r_{k1}, v_i, v_j), (r_{k2}, v_i, v_l)$ から 2 つの概念 v_j, v_l 間の関係を含む tuple に縮約した tuple $(rel*, v_j, v_l)$ を tuple 集合に追加する．例では (arg_1, w^B, y^B), (arg_2, w^B, z^B) から共通の主辞 w^B を消去した $(rel*, y^B, z^B)$ を生成して追加する．

$$T_B \leftarrow T_B \cup \{(rel*, y^B, z^B)\} \tag{4.12}$$

5. 変項の集合 V_A, V_B の間の初期アライメント A_0 を決定する．ここでは変項の出現順に初期アライメント $A_0 = (1, 2, 3, 4)$ と定める．この時変項のアライメントは，$x^A \leftrightarrow x^B$, $y^A \leftrightarrow y^B$, $z^A \leftrightarrow z^B$, $v^A \leftrightarrow w^B$ である．

6. アライメント候補集合 C，探索済みアライメント候補集合 E，最大のアライメントスコア S_{max}，最良のアライメント A_{best} を初期化し，初期アライメント A_0 をアライメント候補集合 C に加える．

$$C = \{A_0\} \tag{4.13}$$
$$E = \emptyset \tag{4.14}$$
$$S_{max} = 0 \tag{4.15}$$
$$A_{best} = 0 \tag{4.16}$$

7. 現在のアライメント候補集合 C 中の最大アライメントスコア S_{max0}，最良アライメント A_{best0} ，次ステップのアライメント候補集合 C_{next} を初期化する．

$$S_{max0} = 0 \tag{4.17}$$
$$A_{best0} = 0 \tag{4.18}$$
$$C_{next} = \emptyset \tag{4.19}$$
$$\tag{4.20}$$

8. アライメント候補集合 C の中の全てのアライメント A_C について，アライメントスコア S_c を計算する．アライメント候補集合 C には A_0 のみが存在するため，$\sigma_{align}(A_0)$ を計算する．アライメントスコアを計算したアライメント A_0 を探索済みアライメント候補集合 E に加える．

$$\begin{aligned}
S_c &= \sigma_{align}(A_0) = \frac{2}{|T_A| + |T_B|} \sum \sigma_T(t_{Ak}, t_{Bl}) \\
&= \frac{2}{6+9}\{sim(I(x^A), I(x^B)) + sim(I(y^A), I(y^B)) \\
&+ sim(I(z^A), I(z^B)) + sim(I(v^A), I(w^B)) \\
&+ \sigma_T((arg_1, x^A, y^A), (arg_1, x^B, y^B)) + \sigma_T((nmod.with, y^A, z^A), (rel*, y^B, z^B))\} \\
&= \frac{2}{15}\{sim(I(x^A), I(y^A)) + sim(I(y^A), I(y^B)) \\
&+ sim(I(z^A), I(z^B)) + sim(I(v^A), I(w^B)) \\
&+ sim(I(x^A), I(x^B))\cdot sim(I(y^A), I(y^B)) \\
&+ sim(I(y^A), I(y^B))\cdot sim(I(z^A), I(z^B))\} \\
&= \frac{2}{15}\{sim(\text{dance}, \text{dance}) + sim(\text{man}, \text{man}) \\
&+ sim(\text{hat}, \text{hat}) + sim(\text{straw}, \text{wear}) \\
&+ sim(\text{dance}, \text{dance})\cdot sim(\text{man}, \text{man}) + sim(\text{man}, \text{man})\cdot sim(\text{hat}, \text{hat})\} \\
&= \frac{2}{15}(1+1+1+0+1+1) = 0.67 \qquad (4.21) \\
E &= \{A_0\} \qquad (4.22)
\end{aligned}$$

9. $S_{max0} < S_c = \sigma_{align}(A_0)$ なので，現在の C 中の最大のアライメントスコアを $\sigma_{align}(A_0)$，最良のアライメントを A_0 に更新する．

$$S_{max0} = \sigma_{align}(A_0) = 0.67 \tag{4.23}$$
$$A_{best0} = A_0 \tag{4.24}$$

10. 現在のアライメント候補集合 C 中の全てのアライメントについて処理を行った結果，$S_{max} < S_{max0}$ なので，最大アライメントスコア S_{max} を S_{max0} $(= 0.67)$ に最良

のアライメント A_{best} を A_{best0} (ここでは A_0) に更新する．さらに，最良のアライメント A_0 に隣接するアライメント候補のうち，探索済みアライメント候補集合 E に存在しないアライメント候補を次ステップのアライメント候補集合 C_{next} に加える．隣接するアライメント候補とは，元になるアライメントの対応付けの 1 つを他の対応に変更するか，2 つの対応を入れ替えるかのいずれかで作られるアライメントである．A_0 のアライメントの 4 点のうち 2 点を入れ替えたアライメントが 6 通り，1 点を v_B の対応 (5) に変更したアライメントが 4 通り，1 点の対応付けをなくしたもの (0) が 4 通りある．このとき，次ステップのアライメント候補集合 C_{next} は以下のようになる．

$$\begin{aligned} C_{next} &= \{(2,1,3,4),(3,2,1,4),(4,2,3,1),(1,3,2,4),(1,4,3,2),(1,2,4,3), \\ &\quad (5,2,3,4),(1,5,3,4),(1,2,5,4),(1,2,3,5), \\ &\quad (0,2,3,4),(1,0,3,4),(1,2,0,4),(1,2,3,0)\} \end{aligned} \tag{4.25}$$

11. アライメント候補集合 C を $C = C_{next}$ として次ステップのアライメント候補集合 C_{next} で更新する．

12. 更新されたアライメント候補集合 C が空でないので，アライメント候補集合 C の中の全てのアライメントに対して，7-11 と同様の手順を繰り返し，最大のアライメントスコア S_{max} および最良のアライメント A_{best} を更新する．

    最大のアライメントスコア S_{max} が更新されなくなったとき，アライメント候補集合 C が空になり，最大のアライメントスコア S_{max}，最良のアライメント A_{best} を最終結果として出力して終了する．

    この例の場合，最終的に $A = (1,2,3,5)$ のとき，すなわち変項のアライメントが $x^A \leftrightarrow x^B$, $y^A \leftrightarrow y^B$, $z^A \leftrightarrow z^B$, $v^A \leftrightarrow v^B$ であるとき，アライメントスコアは式 (4.26) のように計算され，最大のアライメントスコア 0.8 となる．このアライメントは，それぞれの変項に対応している概念に直すと，dance ↔ dance, man ↔ man, hat ↔ hat, straw ↔ straw であり正しいアライメントが得られている．

$$\begin{aligned} S_{max} &= \sigma_{align}(A_{best}) \\ &= \frac{2}{15}\{sim(I(x^A),I(x^B)) + sim(I(y^A),I(y^B)) \\ &+ sim(I(z^A),I(z^B)) + sim(I(v^A),I(v^B)) \\ &+ \sigma_T((arg_1,x^A,y^A),(arg_1,x^B,y^B)) \\ &+ \sigma_T((rel*,y^A,z^A),(nmod.with,y^B,z^B))\} \end{aligned}$$

$$
\begin{aligned}
&= \frac{2}{15}\{sim(\text{dance},\text{dance}) + sim(\text{man},\text{man}) \\
&+ sim(\text{hat},\text{hat}) + sim(\text{straw},\text{straw}) \\
&+ sim(\text{dance},\text{dance})\cdot sim(\text{man},\text{man}) + sim(\text{man},\text{man})\cdot sim(\text{hat},\text{hat})\} \\
&= \frac{2}{15}(1+1+1+1+1+1) = 0.8
\end{aligned} \tag{4.26}
$$

意味グラフの重複度合いに基づくスコア

表 4.1 に示すように，本論文の手法ではアライメントスコア $\sigma_{align}(\cdot)$ の他に意味グラフの重複度合いに基づくスコア σ_{ov} と単語・単語列に基づくスコアを組み合わせて類似度測定モデルを構築する．意味グラフ間の重複度合いを表すスコア σ_{ov} は，SPICE のように意味グラフから変換した tuple 集合間の重複度合いを F1 スコアとして計算したもので式 (4.27) で計算される．

$$
\begin{aligned}
\sigma_{ov} &= \frac{2PR}{P+R} \\
\text{ただし} & \\
P &= \frac{|T_A \oplus T_B|}{|T_B|} \\
R &= \frac{|T_A \oplus T_B|}{|T_A|}
\end{aligned} \tag{4.27}
$$

ここで，$T_A \oplus T_B$ は tuple 集合 T_A, T_B 中の変項をインスタンスに置き換えて T_A, T_B の積集合をとったものを表す．

単語・単語列に基づくスコアについては次節で述べる．

4.4.6 単語・単語列に基づくスコア

意味グラフに基づく類似度測定は，意味構造を詳細に比較できることが利点であるが，構文解析よりも難易度が高い意味解析に依存しているため，意味グラフの比較のみで頑健に類似度スコアを算出することが難しい．そこで，意味グラフに基づく類似度スコア算出を補完するために文献 [69], [70] で述べられているような単語あるいは単語列に基づいた類似度スコアを導入する．導入する類似度スコアは，表 4.1 に示すように単語表層の重複度合いに関するものが 2 種類 (ϕ_1, ϕ_2)，WordNet の synset の重複度合いに関するものが 1 種類 (ϕ_3)，n-gram の重複度合いに関するものが 3 種類 (ϕ_4, ϕ_5, ϕ_6) である．以下それぞれについて説明する．

単語の重複度合いの一つ ϕ_1 は，比較する 2 文中に含まれている全単語数に対して共通に含まれる単語数の割合を表すスコアである．文 S_A，S_B に含まれる単語集合をそれぞれ

W_A，W_Bとしたとき，W_A と W_B の積集合の要素数の W_A と W_B の要素数の合計に対する割合で，式 (4.28) で計算される．

$$\phi_1 = sim_{wo}(W_A, W_B) = \frac{2 \cdot |W_A \cap W_B|}{|W_A| + |W_B|}. \tag{4.28}$$

単語の重複度合いのもう一つ ϕ_2 は，W_A, W_B の包含関係を表すフラグであり，式 (4.29) で表されるように，一方の単語集合が他方を完全に包含しているとき 1，それ以外のとき 0 となる．

$$\phi_2 = \begin{cases} 1 & \text{iff } (W_A \subseteq W_B) \vee (W_A \supseteq W_B) \neq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{4.29}$$

単語の重複度合い ϕ_1 を類義語に拡張したものが，WordNet を用いた重複度 ϕ_3 である．ϕ_3 は，一方の文に含まれている単語それぞれについて，他方の文にどのぐらい類似した単語が含まれているかという度合い $P_{wn}(\cdot,\cdot)$ に基いて算出する値であり，文 S_A から見た値 $P_{wn}(W_A, W_B)$ は，式 (4.30) で計算される [69]．

$$P_{wn}(W_A, W_B) = \frac{1}{|W_B|} \sum_{w \in W_A} score(w, W_B) \tag{4.30}$$

ここで，

$$score(w, W) = \begin{cases} 1 & \text{if } w \in W \\ \max_{w' \in W} sim_{wnp}(w, w') & \text{otherwise} \end{cases}. \tag{4.31}$$

$sim_{wnp}(w_1, w_2)$ は，WordNet の経路類似度 (path similarity) であり，単語 w_1, w_2 の synset をそれぞれ c_1 と c_2 とすると，4.4.4 節で述べたのと同様に WordNet 上で c_1 から c_2 へ辿る最短経路の長さを pathlen(c_1, c_2) としたとき，

$$sim_{wnp}(w_1, w_2) = \frac{1}{\text{pathlen}(c_1, c_2)} \tag{4.32}$$

である．最終的に算出する WordNet の synset を用いた重複度合い ϕ_3 は，式 (4.33) のように $P_{wn}(W_A, W_B)$ と $P_{wn}(W_B, W_A)$ の調和平均で計算される．

$$\phi_3 = \frac{2}{1/P_{wn}(W_A, W_B) + 1/P_{wn}(W_A, W_B)}. \tag{4.33}$$

n-gram の重複度合い ϕ_4, ϕ_5, ϕ_6 は，2 文中に含まれる unigram, bigram, trigram それぞれの数に対して，2 文で共通する unigram, bigram, trigram の数の割合に基づいて計算する．文 S_A, S_B に含まれる n-gram の集合を M_A^N， M_B^N (N は n-gram の種類であり，例えば bigram のとき N=2 と表す) とすると，文 S_A に含まれる n-gram 数に対する 2 文で共

通する n-gram 数の割合と，文 S_B に含まれる n-gram 数に対する 2 文で共通する n-gram 数の割合の調和平均として式 (4.34) のように計算する．

$$sim_{ng}(M_A^N, M_B^N) = 2 \cdot \left(\frac{|M_A^N|}{|M_A^N \cap M_B^N|} + \frac{|M_B^N|}{|M_A^N \cap M_B^N|} \right)^{-1} \tag{4.34}$$

この計算を unigram (N=1), bigram (N=2), trigram (N=3) について行ったものをそれぞれ ϕ_4, ϕ_5, ϕ_6 とする．

4.4.7 類似度スコアの算出

2 つのテキストの類似性を定量的に評価するための類似度スコアとして，評価対象文対に対して人間が主観評価により評定したスコアを用いる．本論文では，後述するように SemEval 2017 で使用された STS ベンチマーク [57] の 文対ごとに付加された類似度スコアを基準として用いる．このベンチマークの類似度スコアは 0 から 5 までの値であり，文対の内容が全く関係がないときに 0 ，文対の意味内容が同じであるときに 5 のスコアが付与されている．未知の文対に対する類似度スコアは，STS ベンチマークの訓練データを用いて前述した意味グラフに基づくスコアと単語・単語列に基づくスコアを予測変数（特徴量）として回帰により算出する．

4.4.8 多言語間の単語類似度

異なる言語間で 4.4.5 節 で述べたような意味グラフのアライメントを行うためには，異なる言語の単語間の対応付けを行う必要がある．本節では，本論文で用いる多言語の単語間の類似度の算出方法について述べる．

多言語オントロジを用いた単語の対応付け

単語で表された表現を概念に抽象化して扱うために，4.4.4 節で述べたように WordNet のようなオントロジと呼ばれる言語資源を用いる．WordNet は英語を対象言語として作成されたものが基本になっているが，EuroWordNet や日本語 WordNet 等の多言語化が Global WordNet Association [6] を通じて進められている．多言語化された WordNet の基本的な構造は対象言語に関わらず同一の synset の体系に基づいているため，異なる言語間であっても概念の対応付けが行える．例えば，英語と日本語の単語間の類似度は英語と日本語の WordNet を用いて，4.4.5 節 で述べているのと同様に式 (4.2) で計算することができる．

[6] http://globalwordnet.org

英語文:	say good-bye to your friends .
日本語文:	友達 に お 別れ を 言い なさい 。

(a) 対訳文

say 友達 good-bye に to を your 別れ friends を . 言い なさい 。

(b) 対訳文を混合した文

図 4.8: 英日対訳文の例.

単語分散表現による単語類似度

word2vec や GloVe 等に代表される単語分散表現を用いて単語間の類似度を計測する方法では，大量の文書集合に含まれる文脈情報を使用して単語をある空間上のベクトルに変換し，これらのベクトル間のコサイン距離等によって類似度を測定することができる．多言語間の単語分散表現による類似度測定では，それぞれの言語の単語が同一のベクトル空間に写像されるような分散表現が必要である．本論文では，対訳コーパスを用いた多言語単語混合分散表現を構築して適用することを試みた．word2vec や GloVe などの単語分散表現では,「周辺で同時に使われている単語が類似している単語同士は意味が近い」という仮説に基づいて単語間の距離を求めている．通常はこの考え方を元に大量の単一の言語のコーパスから単語分散表現を生成するが，2 言語の文に適用する場合対訳コーパス中の対訳になった 2 文を混ぜ合わせて分散表現を生成すれば，2 言語間で類似した単語が近い距離に配置されるベクトル空間が生成されると考えられる．例えば，図 4.8(a) のように英日の単語に区切られた対訳文に対して図 4.8(b) のように単語を 1 語ごとに交互に並べた文を作成し，この英日の単語を混合した単一の文書から単語分散表現を作成する．本論文では，これを対訳混合単語分散表現と呼ぶ．対訳混合単語分散表現の作成は，word2vec で用いられている skip-gram, continuous bag-of-words (cbow) の 2 種類 および GloVe の方法の合わせて 3 種類の方法で行った．

対訳混合単語分散表現とは別の方法として，ある言語の単語ベクトルを線形変換して別のベクトル空間にマッピングすることによって，多言語の単語分散表現を生成する方法が提案されている．本論文では代表的な方法である fastText multilingual [71] を用いた結果とも比較する．fastText multilingual では，78 言語にわたる言語間での単語分散表現の生成方法が提案されている．英日を例にとると，最初に英語 Wikipedia と日本語 Wikipedia からそれぞれ単一言語の単語分散表現を生成する．生成された二つの分散表現を，対訳辞書等による 2 言語間の単語対応情報または二つのベクトルの共通インデックス，あるいはその両方を使って一つのベクトル空間にマッピングする．fastText multilingual は Wikipedia をベースに分散表現を生成しているため含まれている単語数は多い．本研

究では前処理方法や単語の分割方法を合わせるため，公開されている分散表現そのものではなく Wikipedia のソースから前処理と単語分割処理を行って独自に構築した分散表現を使用した．英語の形態素解析は Stanford CoreNLP を，日本語の形態素解析は MeCab に UniDic 辞書を用いて行った，

4.5 関連研究

意味解析については，語彙化文法に基づく構文解析との組み合わせで実現されることが多い．第 2 章で述べたように日本語の組合せ範疇文法 (CCG) を提案した戸次らは，CCG から lambda 式への変換を行う ccg2lambda の開発を行い，テキストの類似度測定やテキスト含意認識 (Recongnizing Texutual Entailment, RTE) などへの適用研究を行っている [33]．また，別の語彙化文法である主辞駆動句構造文法 (HPSG) と統合した意味表現に MRS がある．CCG も HPSG も各言語の語彙を基盤として詳細な文法を記述するため，各語彙と対応する語義ごとの意味構造へ自然に対応させることが可能である．ただし，記述する文法は UD のような依存構造文法に比較してかなり複雑であり，多言語に対応させるために各言語ごとに詳細な文法を作成するのは容易ではない．

テキスト意味類似度測定手法については，前述した Smatch, SPICE の他に SemEval のシェアードタスクに参加したシステムが用いている手法がある．SemEval 2017 参加システムの ECNU [72] は，深層学習に数多くの特徴量により構築したモデルによるアンサンブルシステムである．深層学習のモデルは単語分散表現と LSTM (Long short-term memory) 等に基づいている．使用している特徴量は n-gram の重複度や編集距離，翻訳評価尺度などあらゆる尺度を取り込んでいる．また，BIT [73] は WordNet や BNC コーパスの単語頻度に基づいた semantic information content (IC) を使用しており，さらに単語分散表現によるコサイン類似度を組み合わせて利用している．ECNU, BIT ともに人間の主観評価とかなり相関の高い類似度スコアを出しているが，意味構造情報を扱わないため文間の類似点や相違点の根拠を表示することはできない．

また，多言語のテキスト意味類似度測定は，翻訳品質の評価と関連性の高い技術と見ることができる．翻訳品質自動評価の代表的な手法には BLEU，sentBLEU, NIST などがあるが，基本的には評価対象テキストと正解の翻訳テキストとを比較し，n-gram など文字列の類似性が高いものに高い評価値を与える方法である．最も基本的かつ代表的な自動評価尺度である BLEU は，システムの出力文と正解の翻訳文の間で単語 n-gram が一致する割合に基づいて算出される．BLEU は，通常 2 文間で unigram から 4-gram までの比較を行うが，4-gram については出力文と正解の翻訳文の間で一致するものが全くないために評価値が 0 になることが頻発するという問題がある．これを回避するため，改良手法である sentBLEU では通常の BLEU にスムージング処理を加え文単位の比較・評

価を行うようにしている[7]．sentBLEU は，n-gram の重複度合いに基づく単純な方法だが，翻訳の正解文との類似度合いに基づく翻訳評価では人間の主観評価と高い相関を示す．ただし，表層的な単語列の照合に基づくため異なる言語間の比較には適用できない．

4.6 評価実験

本節では，テキスト意味類似度 (STS) 測定方法，翻訳評価尺度の 2 つの観点から提案手法を評価する．前者は，文と文が対になった 1 文対に対して文間の類似度スコアを算出するタスクである．後者は，機械翻訳システムが出力結果を評価するタスクであり，システムが出力した翻訳と正解の翻訳の類似度スコアを算出することにより行う．

4.6.1 STS ベンチマークによる意味類似度測定の評価

本節では対象言語を英語と日本語として提案する STS 手法の基本的な性能と多言語対応性を評価する実験について述べる．本評価実験では，SemEval 2017 のシェアードタスクで行われた評価と同様に STS ベンチマークを用いてシステムが出力した類似度と人がアノテーションした類似度の間で Pearson の積率相関係数 r により評価する．STS ベンチマークのオリジナルの言語は英語であるが，ニュース記事 (news)，キャプション (caption) の約 7,500 文対について日本語訳を作成した．表 4.2 に類似度の定義とともにオリジナルの英文の例文および日本語翻訳を示す．元のベンチマークと翻訳データを利用することにより，日本語と英語それぞれの単一言語の STS の評価に使用するだけでなく日英間の言語横断 STS の評価に用いることができる．STS ベンチマークの文類似度スコアは，文対が意味内容がほぼ同一のときに最高の値 5 を持ち，文対の意味内容が無関係であるときに最低の値 0 を持つ．表 4.3 にベンチマークの統計値を示す．類似度スコアは，複数のアノテータが付与した整数値の平均をとっているため実数値となっている．

提案手法では 2 文間の類似度スコアを，意味グラフに基づくスコア (σ_{align} と σ_{ov}) と単語・単語列に基づくスコア ($\phi_1, ..., \phi_6$) を使って回帰モデルにより算出する．回帰モデルとして，Liu ら [74] と同様に RBFS カーネルを使った SVR モデルを構築した．パラメータは，STS ベンチマークの訓練データから構築した回帰モデルを開発データに適用した結果から決定し，ペナルティ項 $C = 10$, $\gamma = 0.2$，$\epsilon = 0.5$ とした．

また，UDepLambda による意味解析は，GitHub に公開されているソフトウェア[8] を使用した．UDepLambda の日本語解析モデルは公開されていないため，本論文では日本語の UD の構文木（依存構造）から UDepLambda の lambda 式に変換する規則を追加することにより日本語の解析モデルを構築した．本論文で構築した日本語解析モデルは，UD

[7]統計的機械翻訳システム Moses のツールキットに含まれる．<http://www.statmt.org/moses/>

[8]https://github.com/sivareddyg/UDepLambda

表 4.2: STS ベンチマークの類似度の定義と例文.

類似度	定義 文 1/文 2
5	2 文は完全に等価であり，意味内容が等しい.
	The bird is bathing in the sink. 鳥はシンクの中で水浴びをしています。
	Birdie is washing itself in the water basin 小鳥が洗面器の中で体を洗っています。
4	2 文はほぼ等価であるが，重要でない一部が異なる.
	Two boys *on a couch* are playing video games ソファの上で 2 人の少年がビデオゲームをしています。
	Two boys are playing a video game. ２人の少年がビデオゲームをしています。
3	文は概略的には等価であるが，いくつか重要な点が異なる.
	John said he is considered a witness but not a suspect. ジョンは，彼は今はもはや容疑者ではないと言った.
	“He is not a suspect anymore.” John said. 「彼は目撃者だと考えられたが，容疑者ではない」 とジョンは言った.
2	2 文は等価ではないが，いくつか具体的な共通点がある.
	They flew out of the nest in groups. それらは，群れで巣から飛び出した.
	They flew into the nest together. それらは，一緒に巣に飛び込んだ.
1	2 文は，まったく似ていない.
	The black dog is runining through the snow. 黒い犬が雪の中を走っています.
	A race car driver is driving his car through the mud. レーシングカーの運転手が車で泥の中を走り抜けています.

からの単純な変換のみの対応であるが，意味内容の比較に重要と考えられる基本的な述語項構造の関係は捉えられると考えられる.

多言語単語分散表現の STS への適用評価

多言語 STS (言語横断 STS) の評価には，4.4.8 節で述べた 2 つのタイプの 2 言語単語分散表現を使用した．英日間の多言語意味類似度測定における使用する多言語単語分散表現精度への影響を調べるため，前述した 3 種類の混合表現による単語分散表現と fastText multilingual を使って提案手法による STS ベンチマークを評価した．対訳混合単語分散表現は，対訳コーパスとして日本語学習者である外国語話者のために作られた Tatoeba

表 4.3: 意味的テキスト類似度 (STS) ベンチマーク [57] の統計量（数字は文対）.

Genre	Set			Total	Score		
	Train	Dev	Test		≥ 4	≥ 2	≥ 0
news	3299	500	500	4299	1079	2141	1079
caption	2000	625	625	3250	721	1235	1294
forum	450	375	254	1079	208	432	439
Total	5749	1500	1379	8628	2008	3808	2812

コーパス [9] を使用し，日英の対訳文を混合したものから直接構築した．Tatoeba コーパスでは 1 文について複数の翻訳文が付けられており，15 万文以上の日英対訳文が含まれている．fastText multilingual により 2 つの単一言語の単語分散表現を組み合わせて生成する単語分散表現は英語と日本語の Wikipedia を使用した．STS ベンチマークの開発データ（英日）に対して STS の評価を行った結果を表 4.4 に示す．STS の結果に関しては，全体に対訳混合による分散表現が良い結果となっている．その中でも cbow の結果が最良となった．fastText の結果では，GloVe の結果が最も良い値となっており対訳混合の結果とは一致していない．表 4.4 の右端に，fastText のテスト用の対訳 5,000 語対のコサイン距離の平均を記載している．この値が 1 に近いほど単語レベルでの類似度測定の精度は高いと考えられるが，cbow のよる対訳混合単語分散表現はこの値が高くないことが興味深い．STS で評価するスコアは，単一の単語の類似性の絶対評価性能よりも，多くの単語対での意味の近さと類似度の相関が求められ，cbow は総合的な単語の類似度測定に効果があったと考えられる．これらの結果から単言語（英英，日日）と多言語（英日）の STS 評価には対訳混合分散表現には cbow を， fastText の分散表現には GloVe を用いた．

STS 評価結果

単言語，多言語の STS 評価として，提案手法，意味グラフベースの手法 (Smatch, SPICE)，SemEval 2017 シェアードタスクの参加システムを比較する．AMR による意味解析，比較手法である Smatch，SPICE の類似度計算は GitHub 上で利用できるツールを使用した[10] [11] ．これらの手法のうち提案手法以外については日本語の解析モデルがないため，英語単言語のベンチマークのみの結果と比較する．日本語および英日間の STS が行える方法として，Gravas らが提案している 2 言語単語分散表現による単語アライメントをベースとする方法 (resource-light STS, RL-STS) [75] をベースラインとして比較す

[9] https://tatoeba.org/

[10] https://github.com/mdtux89/{amr-eager,amr-evaluation}

[11] https://github.com/peteanderson80/SPICE

表 4.4: 単語分散表現による STS ベンチマークの評価 (英日).

2 言語分散表現		Pearson の積率相関係数	対訳語の平均コサイン距離
対訳混合	skipgram	0.5578	**0.747**
	cbow	**0.5693**	0.471
	GloVe	0.5318	0.362
fastText	skipgram	0.5309	0.473
	cbow	0.5438	0.375
	GloVe	0.5512	0.223

る．RL-STS は，単言語あるいは 2 言語間の単語分散表現のみを用い，他に各言語に関する詳細な辞書や解析器を必要としないため，言語資源が潤沢にない言語にも STS を適用できる特徴をもつ．RL-STS は，fastText multilingual で行っているように言語ごとに生成した分散表現を線形変換で同じベクトル空間にマッピングする．線形変換のパラメータは少数の単語対訳辞書を用いて決定する．

評価結果を表 4.5 に示す．提案手法は，ベースライン手法と比較すると英語－英語，日本語－日本語の単言語の STS においては僅かながら良い結果を，2 言語間の STS においては大きく上回る結果を示している．このことは意味表現を比較することで，単言語でも従来法と同等以上の結果を示すのみならず，多言語間の類似度測定に同一の手法を適用することで，より効果を発揮することを示している．さらに，提案手法は意味表現をアライメントすることにより，意味表現間の類似点と相違点を可視化できるという利点がある．意味グラフを用いた手法 Smatch と SPICE は提案手法と同様に意味構造の比較を行い類似点と相違点を抽出することが可能であるが，提案手法の精度の方が大きく上回っている．これは，意味グラフに基づく類似度指標の他に単語・単語列に戻づく指標を組み合わせることにより頑健性を高めている効果であると考えられる．一方で，提案手法は SemEval 2017 のトップシステムである ECNU, BIT の精度には及ばない結果となっている．特に ECNU は，あらゆる特徴量を盛り込んだ学習モデルと，LSTM 等ニューラルネットワークとのアンサンブルシステムが評価値を向上させることに強力に働いている．本論文の提案方法の目的は評価値を最大化することではないが，意味構造の情報と基本的な単語・単語列の類似度の組み合わせで達成している水準としては妥当と考えられる．

文長の STS 精度への影響

提案手法の課題は，意味構造を構築する処理そのものの難易度が高く意味解析に完璧な結果を求めるのが厳しい点である．意味解析エラーによる不完全な意味構造や誤った

表 4.5: 単言語および多言語間の STS の結果.

システム / 方法	単語分散表現	英英	日日	英日
提案手法	Wikipedia ◇	.741	**.764**	.525
	Tatoeba †	—	—	**.575**
RL-STS	Wikipedia ◇	.737	.731	.456
Smatch (AMR)	—	.511	—	—
SPICE (Scene Graph)	—	.543	—	—
ECNU	Wikipedia，Gigaword	**.810**∗	—	—
BIT	Gigaword	.809∗	—	—

数値は Pearson の積率相関係数 r を表している.

∗: SemEval 2017 の結果を掲載している.

◇: 英日は fastText multilingual による 2 言語分散表現.

†: 英日は対訳混合単語分散表現.

意味構造は意味情報の有用性を下げ，類似度の確信度の低下を招く恐れがある．正しい解析が行われた場合，ある文に含まれる内容語の数とその解析結果 UDepLambda に含まれる述語の数はおよそ比例すると考えられるので，文に含まれる内容語の数に対して UDepLambda に含まれる述語の数が半分以下である場合に UDepLambda の解析が失敗し解析結果の情報が不完全であるとみなす．全文数に対する情報が不完全な文の割合を情報不完全率と定義し，この率を意味解析に失敗した割合と近似した．表 4.6 は，単言語（英語－英語）の STS において文長に対する解析精度の変化の傾向を表している．表には，意味解析の精度の影響を示すため UDepLambda 情報不完全率を示している．一般的に文長が長くなるほど，解析の曖昧性が増大し解析エラーの発生頻度が高まると考えられる．文長が長くなるほど，意味グラフに基づくのモデルの相関係数は単語・単語列に基づくのモデルに比較して精度の低下度合いが大きくなっている．一方で，UDepLambda の情報不完全率は予想よりは文が長くなっても大きく変化しないが，文長が 31-40 では情報不完全率が大きく上昇しており意味グラフに基づくモデルの精度の低下に影響を与えていると考えられる．意味グラフに基づくスコアと単語・単語列に基づくスコアを組み合わせたモデルは，意味グラフ単独のモデルよりも文長の増加に伴う相関係数の低下の割合が緩やかである．このことは，単語・単語列に基づくのスコアが意味グラフに基づくスコアのみによる情報の欠損を補完している結果とみることができる．

特徴量

類似度スコアを算出する際に特徴量として用いる意味グラフに基づくスコアと単語・単語列に基づくスコアの貢献度合いについて述べる．表 4.7 は，単言語（英語－英語）の STS において 4.4.5 節，4.4.6 節 で述べた STS について使用する特徴量のセットを変化

表 4.6: 文の長さと Pearson の 積率相関係数 r の関係（英英）.

文長	単語・単語列	意味グラフ	全体	UDepLambda 情報不完全率
0-10	.547	.585	.639	.0177
11-20	.577	.620	.715	.0180
21-30	.527	.525	.605	.0154
31-40	.658	.527	.675	.0231
41-	.595	.491	.639	.0070
全体	.571	.601	.690	.0174

表 4.7: STS 評価における特徴量セットと Pearson の積率相関係数（英英）.

特徴量	Pearson の積率相関係数
- 単語の重なり (ϕ_1,ϕ_2)	.687
- Synset の重なり (ϕ_3)	.688
- N-gram の重なり (ϕ_4-ϕ_6)	.684
(- 単語・単語列の特徴量)	.601
- 意味グラフのアライメント (σ_{align})	.680
- 意味グラフの重なり (σ_{ov})	.634
(- 意味グラフの特徴量)	.633
全体	.690

記号 '-' はモデル構築に使用しない特徴量.

させたときの相関係数の結果である．各行は全ての特徴量から先頭に "-" のついた該当する特徴量のみを除いた特徴量を用いて構築したモデルを用いていることを示している．意味的な重なり度合いに基づく特徴量 (σ_{ov}) および単語アライメントの特徴量 (σ_{align}) が，相関係数の向上に大きな貢献をしていることがわかる．図 4.9 は，単言語 (英語－英語) の STS 評価において学習コーパスの量を変換させた場合の相関係数の変化を表している．全ての特徴量を用いた場合の結果は，学習コーパスの量が約 4,000 文に達した時点で最良の相関係数に達している．意味グラフに基づく特徴量を用いた場合 200 文までに急激に相関係数を向上させて最高水準に達しているのに対し，単語・単語列に基づくの特徴量を用いた場合 200 文以下の結果は安定していない．このことは，意味グラフに基づく特徴量を用いた類似度スコアは，単語・単語列に基づく特徴量と比較して少量の訓練データで高い相関を得ることができるという利点を示している．また，単語・単語列に基づく特徴量を用いた場合は，相関係数の上限をさらに押し上げる効果を持ち，大量の訓練データが利用可能な場合に有効であると考えられる．

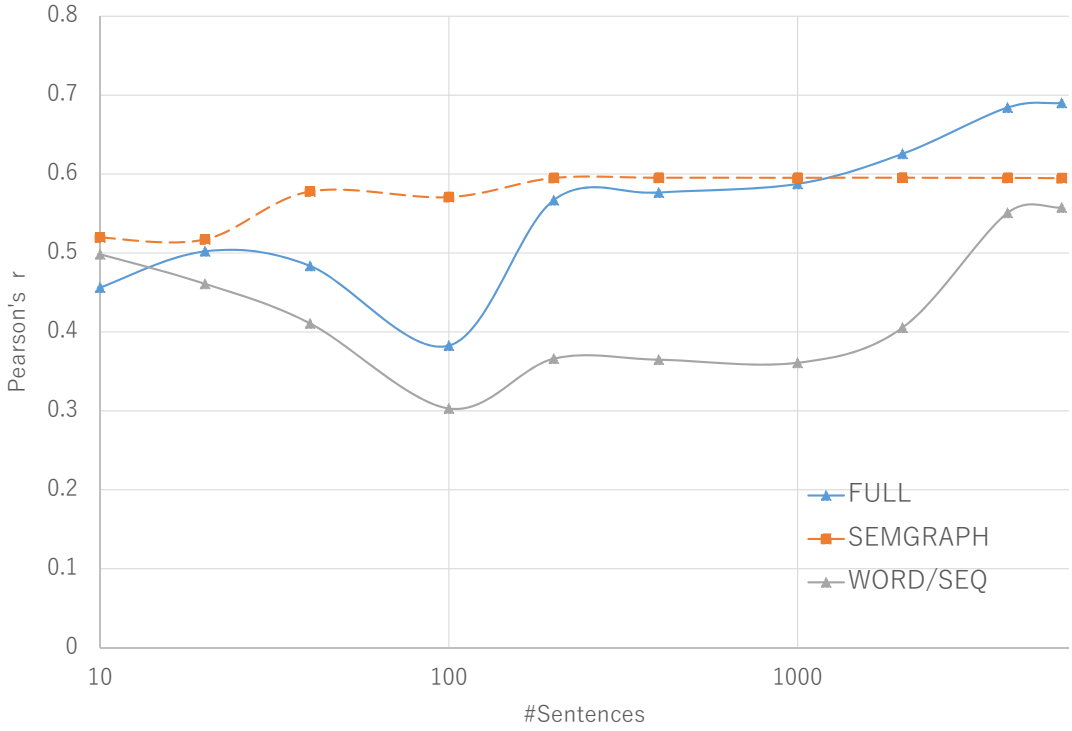


図 4.9: STS benchmark の評価における訓練データの量との関係（英英）.

4.6.2 WMT 2015 評価データに基づく評価

翻訳評価については，WMT 2015 で行われた翻訳評価尺度のシェアードタスクで使用されたベンチマークデータ [76] を用いた．このベンチマークは，原文と正解訳，複数の機械翻訳システムによる翻訳文およびそれぞれに対して人手で優劣を順位付けすることにより付加された相対評価から構成されている．具体的には，英語から 5 か国語への翻訳，5 か国語から英語への翻訳の計 10 方向に対する正解訳と 87 の機械翻訳システムの出力である．本論文の評価実験では，目的言語が英語である 2,169 文対の独英の対訳と 1,500 文対の仏英翻訳文を使った．翻訳評価者によるアノテーションは，各原文ごとに翻訳システムの出力の順位付けを行い相対的な評価が付与されている．それぞれの文対ごとに絶対的な類似度スコアを持たない点が，STS ベンチマークとは異なる．セグメントレベルで翻訳評価システムが出力する翻訳順位と翻訳評価者による順位付けについて，Kendall の 順位相関係数 τ を測ることにより各翻訳評価システムの翻訳評価尺度の質を評価する．本論文の提案手法は，翻訳ベンチマークデータから絶対的な類似度スコアが利用できないため 4.6.1 節と同様に STS ベンチマークの訓練セットから構築した類似度スコア算出モデルを用いて翻訳システムの出力と正解訳との類似度スコアを算出して，類似度スコアの高い順に翻訳システムの順位付けを行った．

比較する翻訳評価手法は，翻訳評価手法のベースラインである sentBLEU，意味グラ

表 4.8: 自動評価尺度による順位付けと人間の順付けの間の比較.

システム / 手法	意味表現	独英	仏英
DPMFcomb	—	**.482**	.395
RATATOUILLE	—	.441	**.398**
提案手法	UDepLambda	.398	**.398**
sentBLEU	—	.360	.358
SPICE	Scene Graph	.211	.211
Smatch	AMR	.303	.285

数値はセグメント単位の Kendall の順位相関係数 τ を示す.

フベースの方法 Smatch, SPICE の他に，WMT 2015 の翻訳評価尺度タスクでの上位システムである DPMFcomb [77] と RATATOUILLE [78] とした．DPMFcomb は，構文解析ベースの尺度 DPMF にいくつかの他の方法を組み合わせている．RATATOUILLE は，BLEU, Meteor, BEER を組み合わせた手法であり，BEER は，文字 n-gram と permutation tree を特徴量に組み込んだ線形モデルを用いている．

表 4.8 に評価結果を示す．提案手法による順位相関係数は，仏英翻訳の評価においてはトップシステムと同等の結果に達している．一方，独英翻訳評価においてはトップシステムとの差が大きくある．この差の主要な原因としては，翻訳評価の場合対象の文が機械の出力した文であり必ずしも文法的な文とは限らないため，構文解析，意味解析を行うのに厳しい条件であることが考えられる．これらのトップシステムの方法は，頑健性の高い単語レベルの解析を有効活用して高いスコアに到達している代わりに，意味グラフのアライメントを使用していないので翻訳評価の根拠となった点を示すことができない．また，翻訳評価用のベンチマークには絶対的な類似度が付加されていないため，提案手法のモデルは翻訳評価データとは関係のない STS の訓練データを用いて構築しており，翻訳評価タスクのためのチューニングは行っていないが，ベースライン手法である sentBLEU の結果を上回っていることは，提案手法の汎用的な性能を表していると考えられる．また，翻訳ベンチマークデータは STS ベンチマークと比較して複雑な文が多く，意味解析のエラーに敏感な本論文の提案手法にとっては厳しい条件であったと考えられるが，意味解析モデルの改善により類似度スコア向上の余地があると考えられる．

4.7 まとめ

本章ではテキストが表す意味的な内容を比較する問題について扱い，意味グラフに基づくスコアと単語・単語列に基づくスコアを組み合わせることにより日本語を含む多言語に適用可能なテキストの意味類似度を測定する方法を提案した．提案手法は，多言語共通の構文解析の枠組みである UD と UD に基づいた意味グラフ UDepLambda を用いること

により幅広い言語に対応可能である点と，意味構造間のアライメントを行うことにより比較対象の文対の類似部分と相違部分を可視化できる点が特徴である．個々の言語への依存度の低い意味グラフの比較をベースとして，多言語対応のオントロジである WordNet，多言語分散表現を組み合わせることにより，単一の言語を対象とした場合と同様の方法で異なる言語間のテキスト意味類似度測定を行えることが大きな利点である．評価実験では，提案手法は意味グラフに基づくスコアに従来型の単語・単語列に基づくスコアを組み合わせてテキスト意味類似度を測定することにより，STS のタスク，機械翻訳評価タスクいずれにおいてもベースライン手法，他の意味グラフに基づく方法の精度を上回る結果を示した．意味解析は対象文の文長が長くなるほど解析の曖昧性が増大し解析精度が低下すると考えられるが，単語・単語列に基づくスコアが情報を補完することによって，長い文での類似度測定の精度を保持している点が提案手法の頑健性を示している．ベースラインとして用いた RL-STS は提案手法と同様に多言語分散表現と単語アライメントを用いるが構文解析が不要で多言語対応がより容易な手法であるが，提案手法は RL-STS と比較しても日本語，英語それぞれ単一言語の STS および英語－日本語間の STS いずれにおいても精度が上回り提案手法の有効性を示している．

提案手法の大きな特徴は多言語対応の構文構造と意味構造の枠組みに基づいている点であるが，本章で用いた構文解析である日本語 UD は単語依存構造による構文解析の枠組みの一つであり，第 2 章，第 3 章で扱ったように新たな日本語の単語依存構造に基づく構文解析が高精度で実現可能であるという事実が多言語処理に有効であることを示す一例となっている．意味解析として用いた UDepLambda の解析器自体には日本語に特化した仕組みは含まれていないが，UD を扱える構文解析器と日本語 UD の解析モデルがあれば，他の言語と互換性のある意味構造を得ることができ，日本語同士および日本語を含む多言語 STS への適用が可能であることを実証したことが本章の研究内容の貢献である．本章では，多言語対応の UD に基づく利点を活かし最低限の対応で日本語の意味解析を実現したが，省略表現の補完など日本語の特徴への対応を取り入れて意味解析を高度化することでテキスト意味類似度測定の精度を改善する余地があると考える．

第5章　結論

本論文は，従来日本語の自然言語処理で扱われてきた独自体系の構文構造に基づく言語解析の課題を解消し，複数の異なる言語を横断して扱う多言語自然言語処理への対応を容易にする新たな日本語解析についてこれまで行ってきた研究成果をまとめたものである．本論文の貢献は，多言語処理に対応した新たな日本語構文解析の設計と大規模データ構築による有効性の検証，新たな日本語構文解析に適合した解析手法の考案，新たな構文解析と拡張した意味解析との組み合わせによる多言語対応のテキスト意味類似度測定方法の考案の 3 点である．以下でその成果を要約し，最後に今後の課題を述べる．

第 1 章では，構文解析と意味解析について概観しながら，従来の日本語の構文解析を他の言語との対応付けをする際や意味解析と連携した処理を行う際の問題点とその解決のための基本方針を示し，論文の構成について述べた．

第 2 章では，日本語の伝統的な構文解析である文節依存構造の問題点を述べ，多言語処理への対応に適した単語依存構造への適用方針と検証結果について述べた．日本語の単語依存構造を考えるにあたっては，(1) 依存構造の単位である単語をどう定めるか，(2) 依存構造の形（依存構造スキーマ）をどう定めるか，(3) 依存構造のラベル（文法機能タイプ）をどう定めるか，について検討と検証を行った結果について述べた．(1) については，形態素情報を保持するのに適した短単位と，構文構造を表すのに適した長単位を単位とする場合を考え，(2) については，日本語の句構造木をベースに主辞の決定の仕方，述語句の結合の仕方で，依存構造のタイプを 6 種類考案し，(3) については，英語の単語依存構造をベースにしながら，日本語の特色である連体修飾句の曖昧性を区別可能な文法機能タイプを追加する依存構造ラベル体系を考案した．(1) から (3) の検討結果に基づき，実際に 4 万文規模の言語コーパスを構築するとともに，既存の構文解析器における解析モデルを実装し，それぞれの要素の精度面への影響を検証した．日本語の単語依存構造による構文解析が従来の文節依存構造による構文解析に比べて，精度の面で同等以上であるとともに，述語項構造情報を単独の述語項構造解析器の解析結果以上の精度で獲得できることを示した．

第 3 章では，第 2 章で述べた日本語の単語依存構造による構文解析を，形態素情報を保持する短単位と，文法機能情報を保持する長単位からなる階層的単語構造に基づいて行う場合の解析方法，効果を検証した結果について述べた．Transition ベースの構文解析方法において長単位チャンキングと長単位依存構造解析を同時に行う手法を提案した．提案

手法である同時解析手法は，逐次処理で長単位チャンキングと依存構造解析を順次行う方法に比べて，複合辞等の機能語を同定する精度を向上させることができることを示した．また，機能語の同定精度を高めることにより，格関係や連用修飾節等の意味解析との連携において重要性が高くかつ難易度が高い依存構造についても精度良く解析できることを示した．

第 4 章では，第 2 章，第 3 章で扱った構文構造より抽象度の高い意味構造を扱いテキスト間の意味内容の類似性を測定する問題を扱った．単一の言語同士でも異なる言語同士でも同一の枠組みで意味内容を比較できるように，多言語化が容易に可能な意味表現を扱えることを方針として，単語依存構造に基づいた多言語共通の構文解析の枠組みである UD と，UD を変換して構成される意味構造である UDepLambda を用いたテキスト意味類似度の測定方法を提案した．本論文の提案手法は，意味表現に UDepLambda を用いることにより UD で解析可能な多様な言語への適用が可能であることを特徴としている．また，比較対象の文の意味表現を意味グラフとして対応付けを行うことにより，比較対象の文において類似している点，相違がある点を明示できる．また，意味グラフのアライメントスコアに加えて，構造情報を含まないオントロジや単語分散表現を組み合わせることにより，人間の主観評価と高い相関をもつ類似度スコアを算出できることを確かめた．

本論文では，日本語を多言語処理の中で有効かつ効率的に扱うことができるように，日本語の特性を保持した形での単語依存構造解析について提案した．日本語の単語依存構造により，多言語対応が容易になるだけでなく，構文構造と意味構造を見通し良く統合でき，解析精度の向上およびより抽象度の高い高度な意味処理の実現に寄与すると考える．一方で，単語依存構造による構文構造は句構造や HPSG 等の語彙化文法と比較すると，より広い範囲で文法機能を保持する単位（長い名詞句や並列構造など）を扱うことはあまり効率的に行うことはできず，長単位を超える単位をどう扱うかが課題の一つであると考える．

また，日本語の UD から UDepLambda による意味構造への変換とその意味構造によるテキスト意味類似性測定への適用結果は，莫大なコストをかけることなく多様な幅広い言語を同一の枠組みで扱うことの可能性を示したと考える．UD からの共通的な変換により生成された日本語の意味表現 UDepLambda は主要な述語項構造の一部を捉えたものに限定されているにも関わらず，構造情報を扱うこととベースライン手法を超える精度を両立している．さらに，日本語の省略表現の補完等を扱うことのできるようなより詳細な意味処理に発展させようとすれば，言語ごとに特有な変換処理を加えなければならない．また，これは UD で共通化した構文解析により各言語特有の言語現象や文法機能が省略されていることと関係しており，今回提案した解析をベースとして日本語特有の言語現象を捉えられるように解析を拡張することによりさらに高度な意味類似性測定が行える可能性があると考える．

謝辞

本論文をまとめるにあたり，ご指導，ご教示を賜った大阪大学大学院情報科学研究科の荒瀬由紀准教授に深く感謝いたします．また，同研究科の原隆浩教授， データビリティフロンティア機構の春本要教授には，本論文の内容に関し，貴重なご助言を頂きましたことを心より感謝いたします．本研究を行う環境と機会を与えていただきました同研究科の鬼塚真教授に深く感謝いたします．また，本研究の内容をご審議いただきましたマルチメディア工学専攻の藤原融教授，松下康之教授，下條真司教授，前川卓也准教授，義久智樹准教授（現大阪大学サイバーメディアセンター），安永憲司准教授，菅野裕介准教授 (現東京大学生産技術研究所)，大倉史夫准教授，伊達進准教授，小島一秀講師，木戸善之講師， 天方大地助教，矢内直人助教，佐々木勇和助教に感謝いたします．

本論文をまとめる機会を与えていただくとともに数々のご支援をいただきましたNTTコミュニケーション科学基礎研究所の前田英作氏，山田武士氏，澤田宏氏，永田昌明氏に心より感謝いたします．

また，日本語や他の言語における単語依存構造について議論いただいている Universal Dependencies プロジェクトに参画している皆様に感謝いたします．

最後に，本研究を進める上で様々な面でご支援いただきました全ての方々に感謝の意を表します．

参考文献

[1] Joakim Nivre. Towards a universal grammar for natural language processing. In *Proceedings of the 16th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2015)*, pp. 3–16, 2015.

[2] 金山博, 宮尾祐介, 田中貴秋, 森信介, 浅原正幸, 植松すみれ. 日本語 Universal Dependencies の試案. 言語処理学会第 21 回年次大会予稿集, pp. 505–508, 2015.

[3] Takaaki Tanaka, Yusuke Miyao, Masayuki Asahara, Sumire Uematsu, Hiroshi Kanayama, Shinsuke Mori, and Yuji Matsumoto. Universal Dependencies for Japanese. In *Proceedings of the Tenth International Conference on the Language Resources and Evaluation (LREC 2016)*, pp. 1651–1658, 2016.

[4] Masayuki Asahara, Hiroshi Kanayama, Takaaki Tanaka, Yusuke Miyao, Sumire Uematsu, Shinsuke Mori, Yuji Matsumoto, Mai Omura, and Yugo Murawaki. Universal Dependencies Version 2 for Japanese. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[5] 田中貴秋, 永田昌明. 日本語の文法機能タイプ付き依存構造解析. 自然言語処理, Vol. 26, No. 2, pp. 441–481, 2019.

[6] Takaaki Tanaka and Masaaki Nagata. Word-based Japanese typed dependency parsing with grammatical function analysis. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP 2015)*, Vol. 2, pp. 237–242, 2015.

[7] 浅原正幸, 金山博, 宮尾祐介, 田中貴秋, 大村舞, 村脇有吾, 松本裕治. Universal Dependencies 日本語コーパス. 自然言語処理, Vol. 26, No. 1, pp. 3–36, 2019.

[8] Takaaki Tanaka and Masaaki Nagata. Constructing a practical constituent parser from a Japanese treebank with function labels. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages (SPMRL 2013)*, pp. 108–118, 2013.

[9] Takaaki Tanaka, Katsuhiko Hayashi, and Masaaki Nagata. Hierarchical word structure-based parsing: A feasibility study on UD-style dependency parsing in Japanese. In *Proceedings of the 15th International Conference on Parsing Technologies (IWPT 2017)*, pp. 56–60, 2017.

[10] 田中貴秋, 林克彦, 永田昌明. 日本語の単語依存構造のための長単位解析. 第 30 回人工知能学会全国大会論文集, 1K32, 2016.

[11] 田中貴秋, 永田昌明, 荒瀬由紀, 鬼塚真. 意味グラフに基づくテキスト類似性尺度の提案. 第 32 回人工知能学会全国大会論文集, 3G1-02, 2018.

[12] 田中貴秋, 荒瀬由紀, 永田昌明, 鬼塚真. Universal Dependencies に基づく多言語間テキスト意味類似性測定. 第 34 回人工知能学会全国大会論文集, 3Q5GS902, 2020.

[13] Taku Kudo and Yuji Matsumoto. Japanese dependency analysis using cascaded chunking. In *Proceedings of the 6th Conference on Natural Language Learning (CoNLL 2002)*, Vol. 20, pp. 1–7, 2002.

[14] Daisuke Kawahara and Sadao Kurohashi. A fully-lexicalized probabilistic model for Japanese syntactic and case structure analysis. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL 2006)*, pp. 176–183, 2006.

[15] Alastair Butler, Zhen Zhou, and Kei Yoshimoto. Problems for successful bunsetsu based parsing and some solutions. 言語処理学会第 18 回年次大会予稿集, pp. 951–954, 2012.

[16] 星野翔, 宮尾祐介, 須藤克仁, 林克彦, 永田昌明. 統計的機械翻訳のための統語に基づく単純な事前並べ替え手法. 情報処理学会論文誌, Vol. 60, No. 3, pp. 890–902, 2019.

[17] Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. Universal Dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, Vol. 2, pp. 92–97, 2013.

[18] Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International*

Conference on Language Resources and Evaluation (LREC 2014), pp. 4585–4592, 2014.

[19] Shinsuke Mori, Hideki Ogura, and Tetsuro Sasada. A Japanese word dependency corpus. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC 2014)*, pp. 753–758, 2014.

[20] Kikuo Maekawa, Makoto Yamazaki, Toshinobu Ogiso, Takehiko Maruyama, Hideki Ogura, Wakako Kashino, Hanae Koiso, Masaya Yamaguchi, Makiro Tanaka, and Yasuharu Den. Balanced corpus of contemporary written Japanese. *Language Resources and Evaluation*, Vol. 48, No. 2, pp. 345–371, 2014.

[21] Yasuharu Den, Junpei Nakamura, Toshinobu Ogiso, and Hideki Ogura. A proper approach to Japanese morphological analysis: Dictionary, model and evaluation. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC 2008)*, pp. 1019–1024, 2008.

[22] 小椋秀樹, 小木曽智信, 小磯花絵, 冨士池優美, 相馬さつき. 「現代日本語書き言葉均衡コーパス」の短単位解析について. 言語処理学会第 13 回年次大会予稿集, pp. 720–723, 2007.

[23] 浅原正幸, 松本裕治. 『現代日本語書き言葉均衡コーパス』に対する文節係り受け・並列構造アノテーション. 自然言語処理, Vol. 25, No. 4, pp. 331–356, 2018.

[24] Sadao Kurohashi and Makoto Nagao. Building a Japanese parsed corpus – while improving the parsing system. In *Treebanks: Building and using parsed corpora*, chapter 14, pp. 249–260. Kluwer Academic Publishers, 2003.

[25] Ivan A. Sag, Thomas Wasow, and Emily M. Bender. *Syntactic Theory: A Formal Introduction*. 2nd Edition, CSLI Publications, 2003.

[26] Takao Gunji. *Japanese Phrase Structure Grammar: A Unification-Based Approach*. D.Reidel(Kluwer), 1987.

[27] Masaaki Nagata and Tsuyoshi Morimoto. A unification-based Japanese parser for speech-to-speech translation. *IEICE Transaction on Information and Systems*, Vol. E76-D, No. 1, pp. 51–61, 1993.

[28] Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. Minimal recursion semantics: An introduction. *Research on Language and Computation*, Vol. 3, No. 4, pp. 281–332, 2005.

[29] Melanie Siegel and Emily M. Bender. Efficient deep processing of Japanese. In *Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization at the 19th International Conference on Computational Linguistics*, Vol. 12, pp. 1–8, 2002.

[30] Francis Bond, Sanae Fujita, and Takaaki Tanaka. The Hinoki syntactic and semantic treebank of Japanese. *Language Resources and Evaluation*, Vol. 42, No. 2, pp. 243–251, 2008.

[31] Ronald M. Kaplan and Joan Bresnan. Lexical-functional grammar: a formal system for grammatical representation. *the Mental Representation of Grammatical Relations*, pp. 173–281, 1982.

[32] 増市博, 大熊智子. Lexical functional grammar に基づく実用的な日本語解析システムの構築. 自然言語処理, Vol. 10, No. 2, pp. 79–109, 2003.

[33] 戸次大介. 日本語文法の形式理論. くろしお出版, 2010.

[34] Sumire Uematsu, Takuya Matsuzaki, Hiroaki Hanaoka, Yusuke Miyao, and Hideki Mima. Integrating multiple dependency corpora for inducing wide-coverage Japanese CCG resources. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, pp. 1042–1051, 2013.

[35] Tomoya Noro, Taiichi Hashimoto, Takenobu Tokunaga, and Hotsumi Tanaka. Building a large-scale Japanese syntactically annotated corpus for deriving a CFG. In *Proceedings of Symposium on Large-Scale Knowledge Resources*, pp. 159–162, 2005.

[36] Alstair Butler, Hinoko Hotta, Ruriko Otomo, Kei Yoshimoto, Zen Zhou, and Houng Zhu. Keyaki Treebank: Phrase structure with functional information for Japanese. テキストアノテーションワークショップ，国立国語学研究所, 2012.

[37] Tsaiwei Fang, Alstair Butler, and Kei Yoshimoto. Parsing Japanese with a PCFG treebank grammar. 言語処理学会第 20 回年次大会予稿集, pp. 432–435, 2014.

[38] Kiyotaka Uchimoto and Yasuharu Den. Word-level dependency-structure annotation to corpus of spontaneous Japanese and its application. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC 2008)*, pp. 3118–3122, 2008.

[39] Marie-Catherine de Marneffe and Christopher D. Manning. The Stanford typed dependencies representation. In *Proceedings of COLING 2008 Workshop on Cross-framework and Cross-domain Parser Evaluation*, pp. 1–8, 2008.

[40] Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. Applying conditional random fields to Japanese morphological analysis. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2004)*, pp. 230–237, 2004.

[41] 小澤俊介, 内元清貴, 伝康晴. 長単位解析器の異なる品詞体系への適用. Vol. 21, No. 2, pp. 379–401, 2014.

[42] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryiğit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. MaltParser: A language-independent system for data-driven dependency parsing. *Journal of Natural Language Engineering*, Vol. 13, No. 2, pp. 95–135, 2007.

[43] Milan Straka and Jana Straková. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 88–99, 2017.

[44] Joakim Nivre. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pp. 351–359, 2009.

[45] 池原悟, 宮崎雅弘, 白井諭, 横尾昭男, 中岩浩巳, 小倉健太郎, 大山芳史, 林良彦. 日本語語彙大系. 岩波書店, 1997.

[46] Ryu Iida, Mamoru Komachi, Kentaro Inui, and Yuji Matsumoto. Annotating a Japanese text corpus with predicate-argument and coreference relations. In *Proceedings of the Linguistic Annotation Workshop*, pp. 132–139, 2007.

[47] Ryu Iida and Massimo Poesio. A cross-lingual ILP solution to zero anaphora resolution. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT 2011)*, pp. 804–813, 2011.

[48] 寺村秀夫. 日本語のシンタックスと意味 I-III. くろしお出版, 1984.

[49] Ryohei Sasano and Sadao Kurohashi. Japanese named entity recognition using structural natural language processing. In *Proceedings of the Third International Joint Conference on Natural Language Processing*, ICJNLP 2008, pp. 607–612, 2008.

[50] 福島健一, 鍛冶伸裕, 喜連川優. 日本語固有表現抽出における超大規模ウェブテキストの利用. 電子情報通信学会第 19 回データ工学ワークショップ/第 6 回日本データベース学会年次大会, DEWS2008, 2008.

[51] Liang Huang and Kenji Sagae. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010)*, pp. 1077–1086, 2010.

[52] Katsuhiko Hayashi, Shuhei Kondo, and Yuji Matsumoto. Efficient stacked dependency parsing by forest reranking. *Transactions of the Association for Computational Linguistics (TACL)*, Vol. 1, pp. 139–150, 2013.

[53] Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. Character-level Chinese dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, Vol. 1, pp. 1326–1336, 2014.

[54] Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Incremental joint approach to word segmentation, pos tagging, and dependency parsing in Chinese. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL 2012)*, Vol. 1, pp. 1045–1053, 2012.

[55] Matthieu Constant and Joakim Nivre. A transition-based system for joint lexical and syntactic analysis. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, Vol. 1, pp. 161–171, 2016.

[56] Ryan McDonald, Kevin Lerman, and Fernando Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X 2006)*, pp. 216–220, 2006.

[57] Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and cross-lingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017)*, pp. 1–14, 2017.

[58] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, pp. 311–318, 2002.

[59] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Mrtha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pp. 178–186, 2013.

[60] Sebastian Schuster, Ranjay Krishna, Angel Chang, Li Fei-Fei, and Christopher D. Manning. Generating semantically precise scene graphs from textual descriptions for improved image retrieval. In *Proceedings of the Workshop on Vision and Language (VL15)*, pp. 70–80, 2015.

[61] Shu Cai and Kevin Knight. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, Vol. 2, pp. 748–752, 2013.

[62] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. SPICE: semantic propositional image caption evaluation. In *Proceedings of the 14th European Conference on Computer Vision (ECCV 2016)*, pp. 382–398, 2016.

[63] Siva Reddy, Oscar Täckström, Slav Petrov, Mark Steedman, and Mirella Lapata. Universal semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2017)*, pp. 89–101, 2017.

[64] Martha Palmer, Daniel Gildea, and Paul Kingsbury. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, Vol. 31, No. 1, pp. 71–106, 2005.

[65] George A. Miller. WordNet: A lexical database for English. *Communications of the ACM*, Vol. 38, pp. 39–41, 1995.

[66] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR 2013) workshop paper*, 2013.

[67] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pp. 1532–1543, 2014.

[68] Zhibiao Wu and Martha Stone Palmer. Verb semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL 1994)*, pp. 133–138, 1994.

[69] Frane Šarić, Goran Glavaš, Mladen Karan, Jan Šnajder, and Bojana Dalbelo Bašić. Takelab: Systems for measuring semantic text similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics (SemEval-2012)*, pp. 441–448, 2012.

[70] 羅文涛, 林良彦. 機械翻訳を利用した異言語文間の意味的類似度計算の評価. 言語処理学会第 22 回年次大会予稿集, pp. 883–886, 2016.

[71] Samuel L. Smith, David H. P. Turban, Steven Hamblin, and Nils Y. Hammerla. Offline bilingual word vectors, orthogonal transformations and the inverted softmax. In *Proceedings of the 5th International Conference on Learning Representations (ICLR 2017) Conference Track*, 2017.

[72] Junfeng Tian, Zhiheng Zhou, Man Lan, and Yuanbin Wu. ECNU at SemEval-2017 Task 1: Leverage kernel-based traditional nlp features and neural networks to build a universal model for multilingual and cross-lingual semantic textual similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017)*, pp. 191–197, 2017.

[73] Hao Wu, Heyan Huang, Ping Jian, Yuhang Guo, and Chao Su. BIT at SemEval-2017 Task 1: Using semantic information space to evaluate semantic textual similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017)*, pp. 77–84, 2017.

[74] Wenjie Liu, Chengjie Sun, Lei Lin, and Bingquan Liu. ITNLP-AiKF at SemEval-2017 Task 1: Rich features based SVR for semantic textual similarity computing. In *Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017)*, pp. 159–163, 2017.

[75] G. Glavaš, M. Franco-Salvador, Ponzetto S. P., and P. Rosso. A resource-light method for cross-lingual semantic textual similarity. *Knowledge-Based Systems*, Vol. 143, pp. 1–9, 2018.

[76] Miloš Stanojevìc, Amir Kamran, Philipp Koehn, and Ondřej Bojar. Results of the WMT15 Metrics Shared Task. In *Proceedings of the Tenth Workshop on Statistical Machine Translation (WMT 15)*, pp. 256–273, 2015.

[77] Hui Yu, Qingsong Ma, Xiaofeng Wu, and Qun Liu. CASICT-DCU Participation in WMT2015 Metrics Task. In *Proceedings of the Tenth Workshop on Statistical Machine Translation (WMT 15)*, pp. 417–421, 2015.

[78] Benjamin Marie and Marianna Apidianaki. Alignment-based sense selection in meteor and the ratatouille recipe. In *Proceedings of the Tenth Workshop on Statistical Machine Translation (WMT 15)*, pp. 385–391, 2015.