КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

На правах рукопису

ВОЗНЮК ТАРАС ГРИГОРОВИЧ

УДК 681.3

ЗАСТОСУВАННЯ СЕМАНТИКО-СИНТАКСИЧНОЇ ТЕНЗОРНОЇ МОДЕЛІ ПРИРОДНОЇ МОВИ ДЛЯ АНАЛІЗУ КОРЕФЕРЕНТНИХ ЗВ'ЯЗКІВ У ТЕКСТАХ

01.05.01 – теоретичні основи інформатики та кібернетики

Дисертація на здобуття наукового ступеня кандидата
фізико-математичних наук

Науковий консультант
МАРЧЕНКО ОЛЕКСАНДР ОЛЕКСАНДРОВИЧ
доктор фізико-математичних наук,
доцент

Київ – 2015

Зміст

Список умовних позначень

ДЗ – дерево залежностей

КП – керуючий простір

ЛСА – латентний семантичний аналіз

НТФ – невід’ємна тензорна факторизація

ПАРКС – паралельні асинхронні рекурсивно керовані системи

ШІ – штучний інтелект, AI, artificial intelligence

BCU – block coordinate update, поблоковий координатний спуск

ConLL – the Conference on Natural Language Learning

FP – false positive, хибний позитивний результат

MUC – message understanding conference,

SAX - Simple API for XML

TP – true positive, правильний позитивний результат

XML - Extensible Markup Language, розширювана мова розмітки

WSD – word sense disambiguation, вирішення лексичної багатозначності

Вступ

Актуальність теми дослідження. В наш час бурхливого розширення сфер застосування інформаційних технологій задачі обробки текстів природною мовою набувають великого значення в науці, економіці та інших сферах життя суспільства. Автоматичний переклад тексту допомагає людям з різних країн розуміти один одного без докладання значних зусиль та витрат часу. Компанії-гіганти, такі як Google та Facebook застосовують методи комп'ютерної обробки текстів для покращення точності цільової реклами, аналізуючи переписку та пошукові запити користувачів. Фірми, що займаються соціологічними дослідженнями, мають змогу оцінювати ставлення людей до певних осіб чи подій на основі автоматичного аналізу публікацій у пресі, залишених користувачами коментарів, повідомлень на форумах. І це далеко не повний перелік практичних задач, для розв'язання яких використовуються методи та алгоритми комп'ютерної лінгвістики.

Для того, щоб автоматичний переклад текстів одразу видавав результат, який не потребує кропіткої додаткової вичитки, необхідно, щоб комп'ютер був здатний розуміти текст на рівні людини. Для оцінки складності задач штучного інтелекту аналогічно до класу NP-повних задач, введено клас AI-повних (**A**rtificial **I**ntelligence) задач, повне вирішення яких вимагає побудови штучного інтелекту, близького до рівня інтелектуальних здібностей дорослої людини.

Задача повного розуміння текстів є вкрай складною, тому її розбивають на підзадачі – розуміння значень окремих слів, розуміння певних типів зв'язків між словами та їх залежностей, розуміння речень природною мовою в контексті всього текста в цілому чи в контексті знань людини про конкретну предметну область. При автоматичному аналізі

природномовних текстів, необхідно різні слова ототожнювати з однією сутністю. Ця підзадача важлива для більш глибокого розуміння тексту. Зв’язок що виникає між такими словами називається кореферентним. Він виникає між словами та словосполученнями, що посилаються на один і той самий об'єкт позамовної дійсності. Так як іменник “дівчина” та займенник “вона” в загальному випадку не обов’язково вказують на одну і ту саму людину, наявність такого зв’язку можна визначити лише з контексту в якому слова були вжиті. Вирішенню задачі кореферентного аналізу були присвячені роботи Шолома Лапіна, Герберта Ліса, Руслана Міткова, Хійан Лі, Анатолія Анісімова, Олександра Марченка та багатьох інших вчених.

Для аналізу текстів довільної тематики традиційні підходи зі складанням словників та правил їх обробки вимагають багато кропіткої праці. Саме тому актуальною є розробка систем, що здатні до самонавчання із автоматизованим виділенням даних про навколишній світ з нерозмічених природномовних текстів. Для вирішення даної задачі можуть бути використані такі потужні моделі як керуючі простори синтаксичних структур речень природною мовою, реалізовані за допомогою методів невід’ємної факторизації лінгвістичних тензорів з елементами машинного навчання. Це в свою чергу вимагає розвитку зазначених моделей семантико-синтаксичних структур природної мови та алгоритмів їх обробки. Важливість розв’язання задачі автоматизації побудови кореферентних зв’язків при обробці природномовних текстів і визначає актуальність дисертаційної роботи.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота є складовою частиною наукових робіт, які ведуться на кафедрі математичної інформатики факультету кібернетики Київського національного університету імені Тараса Шевченка при виконанні

фундаментальної теми “Створення теоретичних основ методів та програмних засобів інтелектуалізації інформаційно–комунікаційних та трансформерних технологій” (державний номер реєстрації – 0111U005416, 2011–2015 рр.)

Мета і задачі дисертаційного дослідження. Метою дисертаційної роботи є побудова математичних моделей представлення семантико-синтаксичних структур текстів та розробка алгоритму кореферентного аналізу на основі створених моделей.

З огляду на мету в роботі ставляться такі задачі:

1. Розробити нові методи оцінки наявності кореферентного зв’язку між парою сутностей за допомогою методів машинного навчання на розширеному семантико-синтаксичними ознаками просторі.
2. Побудувати та дослідити алгоритми оцінки синтаксичного та семантичного паралелізму на основі тензорної моделі.
3. Розробити та математично обґрунтувати алгоритм побудови багатовимірного тензору опису структур природної мови та архітектуру системи, що здатна обробляти великі текстові корпуси.
4. Розробити алгоритм побудови керуючих просторів синтаксичних структур для збагачення тензорної моделі мови та довести його коректність та обчислити складність в термінах швидкодії та пам’яті.
5. Провести експерименти з оцінки точності роботи алгоритму знаходження кореферентних зв’язків та порівняти його результати з іншими алгоритмами.

Об’єкт дослідження – моделі семантико-синтаксичних структур природномовних текстів та методи аналізу текстів на їх основі.

Предмет дослідження – алгоритми кореферентного аналізу в контексті семантико-синтаксичних тензорних моделей.

Методи дослідження. Дослідження базуються на методах та алгоритмах теорії графів, теорії синтаксичного аналізу, тензорного числення, штучного інтелекту, машинного навчання, методики побудови комп'ютерно-лінгвістичних систем.

Наукова новизна одержаних результатів. У дисертаційній роботі розроблено та математично обґрунтовано алгоритми для вирішення задачі ідентифікації та аналізу кореферентних зв’язків у природномовних текстах і отримано такі нові наукові результати:

1. Розроблено нові методи оцінки наявності кореферентного зв’язку між парою слів за допомогою машинного навчання.
2. В методі опорних векторів удосконалено алгоритм навчання для класифікації кореферентних сутностей. Це дало змогу одержати більш точні результати класифікації для типової задачі знаходження кореферентностей, коли кількість негативних прикладів на декілька порядків перевищує кількість позитивних прикладів.
3. Для підвищення точності роботи класифікатора було розроблено розширений простір ознак із додаванням семантико-синтаксичних ознак.
4. Для обчислення нових ознак для класифікатора було вперше побудовано алгоритми оцінки синтаксичного та семантичного паралелізму на основі тензорної моделі.
5. Для тензорної моделі мови розроблені алгоритм наповнення багатовимірного тензору опису структур природної мови та потокову архітектуру системи обробки великих текстових корпусів. Тестування системи було проведено на корпусі розміром 100Гб.
6. Для покращення тензорної моделі мови розроблено алгоритм побудови керуючих просторів синтаксичних структур, доведено

його коректність та обчислено складність в термінах швидкодії та пам’яті.

7. В результаті реалізації даних алгоритмів було підвищено точність системи знаходження кореферентних зв’язків на 3.5%.

Теоретичне і практичне значення одержаних результатів. Наукове значення роботи полягає в розробці алгоритмів побудови керуючих просторів для природномовних текстів за допомогою конвертації дерев виведення та дерев залежностей, а також у побудові тензорної моделі керуючих просторів на основі виділення закономірностей з багатовимірних частотних словників типових керуючих просторів, що є розвитком математичних моделей представлення семантико-синтаксичних структур текстів.

Практичне значення роботи полягає в покращенні результатів вирішення проблеми побудови відношення кореферентності. Зменшення помилок роботи даної підсистеми автоматично покращує результати наступних прикладних задач:

- автоматичний переклад текстів;
- автореферування;
- природномовні інтерфейси до експертних систем та баз даних
- пошукові системи.

Отримані результати впроваджуються для досліджень в області розробки засобів інтелектуальної обробки текстів природною мовою та для читання курсів "Штучний інтелект" та "Комп'ютерна лінгвістика" на факультеті кібернетики Київського національного університету імені Тараса Шевченка.

Особистий внесок здобувача. Всі результати дисертаційної роботи отримані автором самостійно, сформульовані у вигляді теорем та

алгоритмів та строго доведені з використанням допоміжних лем та тверджень, обґрунтовані з посиланнями на використані джерела.

За результатами дисертації опубліковано вісім робіт у наукових фахових виданнях України [5-7, 87, 90-93], одна стаття у науковому журналі, внесеному до міжнародних наукометричних баз [87].

У роботах, опублікованих у співавторстві:

– у статті [87] пошукачу належать результати роботи над розробкою моделі природної мови та програмна реалізація алгоритмів побудови розробленої моделі.

– у роботах міжнародних конференцій [5–7] пошукачу належать результати досліджень над розробкою та реалізацією потокової архітектури обробки великих текстових корпусів, уточненням деталей алгоритмів та їх реалізації, проведення експериментів, розроблено методи оцінки якості результатів та проведення тестування згідно з зазначеними методами.

Апробація результатів дисертації. Результати дисертації були представлені на міжкафедральних семінарах факультету кібернетики Київського національного університету імені Тараса Шевченка та Національного університету «Києво-Могилянсько академія» та доповідались на міжнародних конференціях, зокрема на:

1. ANLDB’2014, 19-th International Conference on Applications of Natural Language to Information Systems, Natural Language Processing and Information Systems, Montpellier, France, June 18-20, 2014;
2. TSD’2014, 17th International Conference on Text, Speech and Dialogue, Brno, Czech Republic, September 8–12, 2014;
3. PolTAL’2014, 9th International Conference on Natural Language Processing, Warsaw, Poland 17–19 September 2014;

Публікації. За результатами дисертації опубліковано 8 наукових праць, у тому числі 5 – статті у фахових виданнях наукових праць, затверджених МОН України, 1 стаття у журналі, внесеному до міжнародних наукометричних баз, 3 – публікації у матеріалах і працях наукових конференцій.

Структура та обсяг дисертації. Дисертаційна робота складається із вступу, трьох розділів, висновків та списку використаних джерел. Загальний обсяг роботи становить 141 сторінка, основний текст роботи викладено на 95 сторінках, список використаних джерел налічує 102 найменування на 11 сторінках. Текст роботи написаний українською мовою.

ОСНОВНИЙ ЗМІСТ

У **вступі** обґрунтовано актуальність роботи, наведено короткий огляд основних результатів досліджуваної галузі, сформульована мета дисертаційної роботи, проаналізовані основні результати та наведено їх новизну.

У **розділі 1** "Аналіз сучасного стану галузі вирішення кореферентностей" дано системний аналіз існуючих моделей мови та алгоритмів, що використовуються для вирішення кореферентностей.

У **підрозділі 1.1** "Постановка задачі вирішення кореферентностей" визначено місце задачі вирішення кореферентностей в практичних задачах комп'ютерної лінгвістики. Сформульовано означення кореферентного зв'язку та наведені приклади речень з кореферентними сутностями.

У **підрозділі 1.2** "Алгоритми вирішення проблеми анафори" проаналізовано деталі роботи існуючих алгоритмів одного з типів кореферентних зв'язків – анафоричних займенників. Один з перших

алгоритмів для вирішення займенникової анафори був розроблений Джеррі Хоббсом [36]. В статті пропонується 2 підходи до вирішення проблеми анафори. Перший наївний алгоритм обходить дерево виводу спеціальним чином, пов'язуючи іменники коректного роду та числа з займенниками. В наступному алгоритмі Хобса продемонстровано, як за допомогою складної системи семантичного аналізу англомовних текстів можна знаходити пари антецедент-анафора з більшою точністю. Робота цього алгоритму була продемонстрована на декількох реченнях.

Іншим оригінальним алгоритмом вирішення проблеми анафори є алгоритм Шалома Лаппіна та Герберта Лісса [65]. Алгоритм застосовується до формальної моделі тексту, що породжена парсером граматик слотів МакКорда і спирається на міри характерних особливостей, отриманих з синтаксичної структури і динамічної моделі станів. На простій тестовій вибірці з 360 займенниками, алгоритм успішно ідентифікував антецедент в 86% випадків. Алгоритм Лаппіна та Ліса показав на 4% кращий результат, ніж алгоритм Хоббса.

Один з найбільш розвинених на даний момент алгоритмом вирішення проблеми анафори є алгоритм Міткова [55, 56]. Даний алгоритм є розвитком ідей Лапіна і Ліса. В роботі вводяться нові критерії оцінки: синтаксичний паралелізм, повторюваність кандидата, схоже положення, підмет, доповнення, часте згадування. Також з'являються штрафні критерії, наприклад у випадку не визначеної граматичної ролі слова.

В роботі [85] для вирішення задачі знаходження анафоричних зв'язків було використано методи машинного навчання. Для прийняття рішення алгоритмом використовувалася система алгоритмів опорних векторів (Support Vector Machines). В роботі представлені модифікації класичного алгоритму навчання виходячи з особливостей тренувальної вибірки поставленої задачі. В якості прецеденту виступає пара анафора-

антецедент, яка належить до одного з двох класів залежно від наявності в ній референції.

В **підрозділі 1.3** “Алгоритм вирішення кореферентностей” досліджено алгоритми знаходження кореферентностей довільного типу. Один з перших побудованих алгоритмів для знаходження кореферентностей в нерозмічених текстах без обмежень тематики був розроблений в 2001 році [35]. Метод заснований на машинному навчанні. Для кожної пари слів та словосполучень (i,j), що перевіряються на кореферентність, розглядаються 12 простих евристичних критеріїв. Для побудови класифікатора формується тренувальна вибірка з пар слів, для яких відомо, чи вони є кореферентними. Також для них відомо значення 12 сформованих критеріїв.

В роботі [39] представлено нове архітектурне рішення розв’язку проблеми кореферентність в вигляді поєднання декількох “решіт”. Кожне решето реалізовує деякий алгоритм оцінки пари слів чи словосполучень на кореферентність, працює незалежно, та може повертати одне з трьох значень : сутності кореферентні, не кореферентні чи “не знаю”.

Робота Лі [2] є розширенням розробленої в попередній дослідженій системі архітектури заснованої на решетах. В ній вводяться додаткові реалізації решіт, а також алгоритм визначення кандидатів слів та словосполучень на кореферентність. Ця робота продемонструвала найкращі результати на тестовій вибірці конференції CoNLL-2011 [22].

В **підрозділі 1.4** “Формальні моделі мови” проаналізовано існуючі моделі мови, що можуть бути використані для розв’язку кореферентностей.

В **підрозділі 1.4.1** досліджено латентний семантичний аналіз (ЛСА) [44]. ЛСА – це метод обробки інформації природною мовою, що дозволяє проаналізувати взаємозв'язок між колекцією документів і термінами, які в них зустрічаються. Алгоритм зіставляє деякі фактори (теми) всім

документам і термам. В основі методу латентно-семантичного аналізу лежать принципи факторного аналізу, зокрема, виявлення латентних зв'язків досліджуваних явищ або об'єктів. При класифікації чи кластеризації документів цей метод використовується для вилучення контекстно-залежних значень лексичних одиниць за допомогою статистичної обробки великих корпусів текстів.

Недоліком ЛСА є його обмеженість обробкою двовимірних матриць, а значить виділяються тільки бінарні зв'язки. В природномовних текстах зв'язки в загальному випадку можуть бути складнішими, і охоплювати одразу групу слів. Для вирішення цієї проблеми, було використано невід'ємну факторизацію тензорів.

В **підрозділі 1.4.2** проаналізовано невід'ємну факторизацію тензорів [75] як розвиток ідеї ЛСА.

Означення. Нехай $I_1, I_2, \dots, I_N \in \mathbb{N}$ – розміри тензора по кожному з напрямків. В роботі тензор $Y \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ порядку N визначається як N-вимірний масив з елементами $y_{i_1, i_2, \dots, i_N}, i_n \in \{1,2,\dots, I_N\}, 1 \leq n \leq N$.

Зовнішній добуток тензорів $Y \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ та $X \in \mathbb{R}^{J_1 \times J_2 \times \dots \times J_M}$ визначається за допомогою формули:

$$Z = Y \circ X \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N \times J_1 \times J_2 \times \dots \times J_M},$$

де $\circ$ – символ операції зовнішнього добутку.

По-елементно операція зовнішнього добутку визначається як:

$$z_{i_1, i_2, \dots, i_n, j_1, j_2 \dots, j_m} = y_{i_1, i_2, \dots, i_n} x_{j_1, j_2 \dots, j_m},$$

де

$z_{i_1, i_2, \dots, i_n, j_1, j_2 \dots, j_m}$, $y_{i_1, i_2, \dots, i_n}$ та $x_{j_1, j_2, \dots, j_m}$ –

елементи тензорів Z, Y, Z відповідно

Особливими випадками є зовнішні добутки двох векторів $a \in \mathbb{R}^I$ та $b \in \mathbb{R}^J$:

$$A = a \circ b = ab^T \in \mathbb{R}^{I \times J}$$

та трьох векторів $a \in \mathbb{R}^I$, $b \in \mathbb{R}^J$ та $C \in \mathbb{R}^Q$:

$$A = a \circ b \circ c \in \mathbb{R}^{I \times J \times Q},$$

де $z_{ijq} = a_i b_j c_q$

Постановка задачі факторизації тензорів. Для даного тензора $Y \in \mathbb{R}^{I \times T \times Q}$ і додатнього індексу *J* знайти три матриці, що називаються матриці навантаження або фактори, $A = [a_1, a_2, \ldots, a_J] \in \mathbb{R}^{I \times J}$, $B = [b_1, b, \ldots, b_J] \in \mathbb{R}^{T \times J}$, $C = [c_1, c, \ldots, c_j] \in \mathbb{R}^{Q \times J}$, що задовільняють наступним умовам:

$\|E\| \to min$

$$Y = \sum_{j=1}^{J} a_j \circ b_j \circ c_j + E \qquad (*)$$

Умова (*) в поелементній формі:

$$y_{itq} = \sum_{j=1}^{J} a_{ij} b_{tj} c_{qj} + e_{itq}$$

Індекс *J* в даній задачі називають кількістю фактор-множин.

В **розділі 2** “Побудова тензорної моделі керуючих просторів природномовних речень” розроблено алгоритми побудови формальних моделей мови для заданого речення та встановлено їх обчислювальну складність.

В **підрозділі 2.1** “Аналіз формальної моделі керуючих просторів природномовних речень” проаналізовано зазначену формальну модель синтаксичного представлення [86]. На відміну від суто лінгвістичного

підходу, речення в цій моделі розглядається як деякий динамічний обчислювальний рекурсивний процес, що розвивається в керуючому просторі, що пов'язує синтаксично згруповані частини пропозиції інформаційними каналами. Структура керуючого простору відображає семантику визначальних і предикативних конструкцій мови.

Якщо два об'єкти *A* і *B* вступають у відношення *C*, то ми виділяємо об'єкт (припустимо *A*), що викликає (ініціює, породжує) це відношення, і об'єкт, на який передається це відношення. Таким чином, виділяємо два види спрямованих зв'язків: від об'єкта-генератора відношення до самого відношення і від відношення до підлеглого об'єкту. Перший вид зв'язку називаємо *α*-зв'язком (зв'язок генерування), другий – *β*-зв'язком (зв'язок поширення). Дієслова визначають відносини між об'єктами, тому в стандартній схемі простого речення: “іменник – дієслово – іменник” α-зв'язок направлений від першого іменника до дієслова і β-зв'язок направлений від дієслова до іменника-визначення.

Розглянемо приклад. Дівчинка збирає квіти. Об'єкт “дівчинка” генерує відношення збирає і направляє його на об'єкт “квіти”. Тому *α-β*-структура цього речення має вигляд Рис. 1.а). Розглянемо фразу: Красива дівчинка. Тут об'єкт дівчинка генерує унарне ставлення красива і передає це відношення собі ж (Рис. 1.б). Виникає кільцевий зв'язок, що характеризує зміст словосполучення.

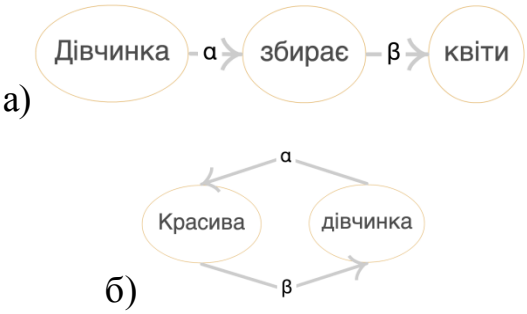


Рис. 1. Керуючий простори фрагментів речень а) “Дівчинка збирає квіти” та б) “Красива дівчинка”.

У **підрозділі 2.2** “Постановка задачі побудови керуючого простору” наведено вхідні та вихідні дані алгоритму та спроектовано рекурсивні структури даних для представлення керуючого простору.

Вхідними даними алгоритму є не розмічений текст природньої мови.

Вихідними даними алгоритму є керуючий простір природномовних текстів в деякому представленні. Для повного опису вихідних даних алгоритму залишилося дати формальний опис цієї структури даних. Проведемо декомпозицію КП та формалізуємо його. Керуючий простір складається з елементів та зв’язків між ними.

Зв’язки бувають 2 типів – альфа та бета. Окрім цього типовим для КП синтаксичних структур є ще один складений зв’язок – кільцевий альфа-бета зв’язок. Тому для зручності представлення даних виділимо його в окремий тип зв’язку.

Зв’язок об’єднує 2 елементи утворюючи новий елемент керуючого простору. Елементи утворені таким чином будемо називати складеними. Інформація про тип зв’язку, а також посилання на елементи, що зв’язуються, будуть зберігатися в самих елементах. Крім цього є елементи КП, що вказують на одне слово вхідного речення. Такі елементи будемо називати базовими.

Кожний елемент керуючого простору має деяке синтаксичне та семантичне значення. Синтаксичне значення базових елементів визначається частиною мови слова, якому він відповідає. Синтаксичне значення складених елементів визначається тегом синтаксичної групи до якої він входить. Семантичне значення елементу визначимо деяким головним словом.

За допомогою розробленої структури даних (Рис. 2.) ми представили керуючий простір синтаксичних структур (Рис. 3. а) в вигляді бінарного дерева (Рис. 3. б).

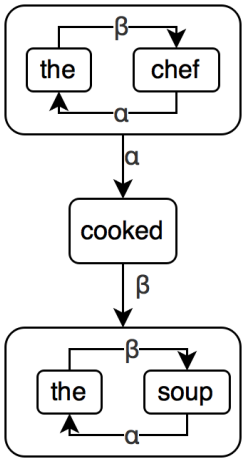
Рис. 2. UML діаграма структури даних керуючого простору

Теорема 1. Для будь-якого керуючого простору синтаксичних структур природномовних речень існує єдине представлення його в вигляді бінарного дерева

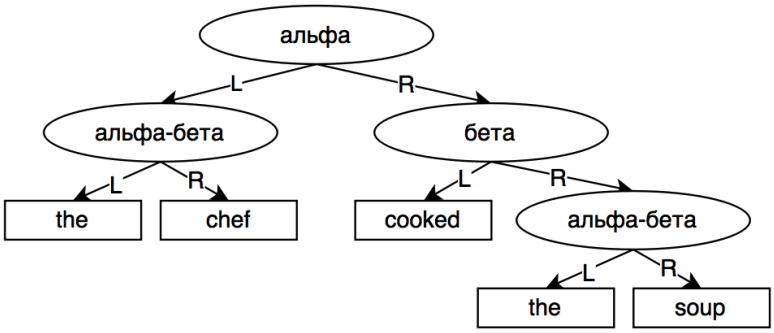
Теорема 2. Для будь-якого представлення КП в вигляді бінарного дерева існує єдиний КП в класичному представленні.

Дані теореми доводять існування бієктивного відображення між двома різними представленнями керуючих просторів.

У **підрозділі 2.3** “Попередній синтаксичний аналіз” проведено системний аналіз кожного етапу синтаксичного аналізу текстів з погляду на їх використання при побудові керуючих просторів.



а)



б)

Рис. 3. а) Класичне представлення КП синтаксичних структур. б) представлення КП в формі бінарного дерева

У **підрозділі 2.3.1** досліджено методи розбиття тексту на речення, проаналізовано типові помилки таких алгоритмів та обгрунтовано методи їх усунення.

У **підрозділі 2.3.2** проаналізовано проблеми формалізації обсягу лексеми природномовного тексту та досліджено принцип роботи алгоритмів розбиття речення на лексеми.

У **підрозділі 2.3.3** було сформульовано постановку задачі морфологічного аналізу, досліджено формат вихідних даних системи морфологічного аналізу стенфордського парсера та розглянуто приклади його роботи.

У **підрозділі 2.3.4** досліджено задачу виділення іменованих сутностей, їх типи, та приклад роботи стенфордського парсера та типові помилки.

У **підрозділі 2.3.5** розроблено алгоритм виділення іменованих сутностей, що заснований на аналізі сторінок Вікіпедії, та представлено деталі його роботи. Особливу увагу було приділено інтеграції алгоритму в стенфордський парсер.

У **підрозділі 2.3.6** наведено опис дерев виводу Хомського та проаналізовано приклад дерева виводу природномовного речення англійською мовою побудованого стенфордським парсером.

У **підрозділі 2.3.7** проаналізовано граматику залежностей, типи дерев залежностей та їх приклади.

У **підрозділі 2.4** “Алгоритм побудови шестивимірного тензора для задачі пошуку прихованих семантичних зв’язків в корпусах природномовних текстів” розроблено нову модель мови, обгрунтовано необхідність її побудови та побудовано алгоритм та структури даних.

У **підрозділі 2.4.1** обгрунтовано причини збільшення розмірності простору. Тензори розмірності три були введені через неповноту моделей заснованих на тензорах розмірності два.

Проаналізуємо наступний приклад : “радіо грає на піаніно”. Оскільки пари слів речення “радіо грає” та “грає на піаніно” є коректними, то і вся фраза буде вважатися коректною при аналізі за допомогою ЛСА. Розглянемо тензор розмірності три. Його вісі – підмети, присудки та додатки. Трійка “радіо грає на піаніно” буде входити в нього з значенням нуль, оскільки в природномовних текстах такі фрази можуть зустрічатися тільки в казках, а їх ми не будемо брати до уваги. Тоді як “піаніст грає на піаніно” буде входити з вагою більшою за нуль.

Користуючись аналогічними міркуваннями можна користуватися тензорами ще більших розмірностей. Новими вісями можуть бути прямий об’єкт, не прямий об’єкт, модифікатор заперечення, кількості, часу, прийменник тощо.

У **підрозділі 2.4.2** розроблено рекурсивний алгоритм обчислення елементів тензора з дерев виводу (ДВ) та дерев залежностей (ДЗ). Алгоритм виділяє всі можливі типи зв’язків, з яких на наступному етапі обираються лише шість найголовніших для збереження в тензорі. Алгоритм працює в декілька кроків:

1. За допомогою рекурсивної реалізації методу пошуку в глибину починаючи з кореня ДВ обходимо синівські вершини. Для кожної синівської вершини, що є простим реченням, виконуємо крок 2.

2. Для кожного простого речення дерева виведення знаходимо піддерево дерева залежностей, що повністю належить даному простому реченню.
3. Знаходимо корінь дерева залежностей для знайденого піддерева
4. Рекурсивною реалізацією пошуку в глибину обходимо дерево залежностей, починаючи з визначеного на попередньому етапі кореня, акумулюючи пари тип зв'язку ДЗ – слово.
5. Для кожного простого речення з акумульованих пар тип зв'язку ДЗ – слово виділяємо слова шести визначених типів. Це і є шуканий елемент тензору.

У **підрозділі 2.4.3** було проаналізовано вимоги до способу збереження багатовимірного тензору. Ця задача має важливе значення для аналізу великих текстових корпусів (Big data). Система збереження має гарантувати цілісність даних у випадку аварійного завершення, можливість продовження роботи з останньої обробленої статті та швидкий доступ до багатовимірного тензору. Швидкий доступ до елементів тензору забезпечується за допомогою хеш-таблиць.

У **підрозділі 2.4.4** було описано використання факторизації природномовних тензорів, обґрунтовано необхідність цього кроку. Для невеликого прикладу проведено всі необхідні обчислення.

У **підрозділі 2.5** "Алгоритм побудови керуючого простору природномовних речень" детально описано принцип роботи розробленого алгоритму. Входом алгоритму є дерево залежностей, дерево виводу та інші результати попереднього синтаксичного аналізу. Виходом алгоритму є керуючі простори природномовних структур.

У **підрозділі 2.5.1** побудовано алгоритм конвертація типу зв'язку дерева залежностей в тип зв'язку керуючого простору. Цей алгоритм вирішує часткову задачу – визначення типу зв'язку КП між існуючими КП, пов'язаних між собою зв'язком ДЗ.

Головна ідея алгоритму полягає в припущенні, що кожному з типу зв’язків дерева залежностей відповідає один з типів зв’язку керуючого простору – альфа-, бета-, та кільцевий альфа-бета-зв’язок певного напрямку. Конвертація типу відбувається за допомогою звернення до попередньо побудованої хеш-таблиці словника. За рахунок вдалого вибору хеш-функції та завантаження словника типів повністю в пам'ять на етапі попередньої обробки даних визначення типу зв’язку КП відбувається за константний час.

Лема 1. Алгоритм конвертації працює за час *O(1)* та потребує *O(N)* додаткової пам’яті, де *N* – кількість типів зв’язків Дерева залежностей.

У **підрозділі 2.5.2** наведено та обгрунтовано алгоритм створення елементу керуючого простору для двох слів. Нехай дано 2 слова, їх POSTag’и, тип зв’язку ДЗ між ними та тег найменшого піддерева дерева виводу, що їх містить. Тоді тегом синтаксичного значення КП буде тег найменшого піддерева виводу. Тип зв’язку визначається за допомогою алгоритму конвертації типу зв’язку.

У **підрозділі 2.5.3** розроблено алгоритм створення складного елементу керуючого простору для піддерева дерева виводу. Нехай для деякого вузла дерева виведення дано його прямі нащадки – піддерева з відповідними вже побудованими керуючими просторами. Дані КП знаходяться на одному рівні дерева виводу. Рівні ієрархії в керуючому просторі для них будемо визначати пріоритетом тегів дерева залежностей. Найвищий пріоритет мають конкретні та специфічні типи зв’язку, що впливають лише на маленькі групи слів (наприклад зв’язок типу прикметник-іменник). Найменший пріоритет мають абстрактні зв’язки, що зв’язують великі групи слів, наприклад зв’язок до сполучних слів між простими реченнями.

Розглянемо оптимізований алгоритм об’єднання:

```
function об’єднатиДекількаКПВОдин
```

```
    (Базовий_елемент кп[]) : Базовий_елемент
begin
    while sizeof(кп) > 1
    begin
        найбільшийПріоритет ← 0
        параЗНайвищимПріорітетом ← -1
        n ← sizeof(кп)
        for i ← n step -1 until 2
        begin
            if існує зв'язок зв в граматичних
            відносинах між словами кп[i] і кп[i-1] and
            зв.пріоритет > найбільшийПріоритет then
            begin
                найбільшийПріоритет ← зв.пріоритет
                параЗНайвищимПріорітетом ← i
            end
        end
        if параЗНайвищимПріорітетом != -1 then
        begin
            i ← параЗНайвищимПріоритетом
            замінити кп[i] та кп[i-1] простором
            об'єднати2КП(кп[i], кп[i-1], тип
            граматичного зв'язку)
        end
    end
    return кп[0]
end
```

Лема 2. Оптимізований варіант алгоритму об'єднання працює за час $O(N^2)$, та потребує $O(M)$ додаткової пам'яті, де N – кількість КП, що необхідно об'єднати, M – кількість типів зв'язків КП.

У **підрозділі 2.5.4** розроблено та досліджено узагальнюючий алгоритм побудови КП, що полягає рекурсивному обході дерева виводу та використанні описаних вище допоміжних алгоритмів.

```
function побудуватиКП(Дерево р) : Базовий_елемент
begin
    if р – листок
    then return Базовий_елемент(синтаксичний тег
    отриманим з морфологічної розмітки, слово-термінал
    для цього листку)
    else
    begin
        сини ← список синів вершини р
        for син ← сини
            кп ← union(кп,  {побудуватиКП(син)})
        return
            об'єднатиДекількаКПВОдин(керуючіПростори)
    end
end
```

Теорема 3 (про обчислювальну складність). Алгоритм побудови КП працює за час $O(N*K)$ та потребує $O(M+K*log_K(N))$ пам'яті, при умові, що кожен вузол має K синів, N – сумарна кількість терміналів в дереві, M – кількість типів дерева залежностей. $K>1$

При доведення часової складності алгоритму було визначено, що кількість викликів алгоритму об'єднання з часовою складністю $O(K^2)$ має порядок $O(N/K)$.

Теорема 4 (повна коректність). Алгоритм будує керуючий простір за скінченну кількість кроків, що відповідає вхідним Дереву виводу та Дереву залежностей.

У **підрозділі 2.5.5** обгрунтовано метод покращення створеного КП за допомогою використання інформації про виділені іменовані сутності.

Результат оформлено як алгоритм попередньої модифікації дерева виводу перед викликом алгоритму побудови КП. Алгоритм модифікації ДВ складається з наступних кроків:

1. Всі слова іменованої сутності були видалені з існуючого дерева виведення разом з тегами
2. Створено нове піддерево дерева виводу, що містить одну батьківську вершину з тегом NP – іменникова група, та всі слова іменованої сутності в якості синів-листків цієї вершини.
3. Для сформованого піддерева було створено КП.
4. Головне слово КП визначає місце вставки нового піддерева в існуюче дерево виводу.

У **підрозділі 2.6** “Виділення типових елементів природномовних речень” спроектовано систему для обробки великих текстових корпусів.

У **підрозділі 2.6.1** розроблено ER-модель бази даних для збереження тензора. Для створеної моделі було доведено відповідність нормальній формі Бойса-Кода.

У **підрозділі 2.6.2** було розроблено архітектуру обробки великих текстових корпусів. Побудовано наступні фільтри : читання файлу, розархівування потоку даних, розбір XML формату за допомогою SAX та інтерфейс процесора сторінок Вікіпедії.

У **підрозділі 2.7** “Профілювання та оцінка результатів” було оцінено якість та швидкодію розробленої підсистеми. Оцінку якості роботи алгоритму представлено в термінах точності та покриття. Результати наведено окремо по кожному з типів зв’язків КП та в цілому. Проведено експериментальне встановлення швидкодії реалізації алгоритму за допомогою засобів профілювання додатків. В середньому алгоритм побудови керуючого простору вимагає менше ніж на порядок часу ніж попередній синтаксичний аналіз. Обробка одного рівня Дерева виведення

займає близько 400 мілісекунд на одному ядрі процесора з тактовою частотою 2.5 ГГц.

У **розділі 3** “Алгоритми визначення кореферентних зв’язків за допомогою статистичної інформації типових структур керуючих просторів” було використано результати попереднього розділу для вирішення практичної задачі комп’ютерної лінгвістики. Для визначення кореферентностей використано архітектуру засновану на ситах (фільтрах), до якої було додано декілька нових решіт, що реалізують класифікатор методу опорних векторів.

У **розділі 3.1** “Пошук сутностей” проаналізовано принципи роботи алгоритму виділення сутностей в тексті за допомого інформації представленою деревом виводу. З дерева виводу обираються всі іменникові фрази (мають тег NP та NNP). Також до переліку знайдених сутностей додаються іменовані сутності (власні назви). В результаті отримуємо список кандидатів в сутності для знаходження кореферентних зв’язків. Наступним кроком список фільтрується згідно визначених правил. Наприклад видаляються сутності, що входять в інші сутності, тому що вони посилаються на один і той же об’єкт, але довший варіант цієї сутності містить більше додаткової інформації.

У **розділі 3.2** “Зведення задачі знаходження кореферентностей до тернарного класифікатора пари сутностей” досліджено метод зведення задачі кластеризації кореферентностей до задачі класифікації. Спочатку список виділених сутностей сортується за номером речення. Всередині одного речення сутності сортуються за відстанню від кореня дерева виводу. Таким чином надається перевага найбільш великим групам слів, що йдуть в тексті найпершими. В кожному новоутвореному кластері сутностей ми будемо порівнювати тільки перші з них. Спочатку буде застосовано перше решето до всіх пар сутностей, потім друге і так далі. Кожного разу, коли ми знаходимо пару кореферентних сутностей, ми

починаємо процес спочатку, оскільки властивості сутностей в кластерах могли збагатитися новою інформацією. Всі сутності одного кластеру вказують на один об’єкт, а тому діляться між собою синтаксичними атрибутами (число, рід, істота чи не істота, тощо). Внаслідок використання інформації представленою кожною з сутностей в деякому кластері кореферентних сутностей, алгоритм робить більш точний вибір, ніж при використанні інформації лише одної сутності з цього кластеру.

У **розділі 3.3** “Порівняння піддерев” досліджені принципи синтаксичного та семантичного паралелізму та розроблено алгоритм реалізації цих принципів за допомогою методу оцінки подібності дерев керуючих просторів синтаксичних структур для пари сутностей. Побудований алгоритм рекурсивно обходить одночасно два КП, аналізуючи типи зв’язків та семантичну близькість слів за допомогою порівняння фактор-множин розкладених частотних словників типових КП.

Теорема 5. Алгоритм оцінки семантико-синтаксичного паралелізму працює за час $O(min(N_1,N_2)K)$ та використовує $O(log_2 min(N_1,N_2)$ пам’яті, де N_1 та N_2 – кількість елементів в КП, що аналізуються, K – кількість фактор-множин, обрана при факторизації тензора.

У **розділі 3.4** “Оцінка наявності кореферентного зв’язку за допомогою виявлення семантичного паралелізму керуючих просторів” розроблено алгоритм виявлення кореферентних зв’язків.

Вхід алгоритму : дві сутності “а” та “б” та відповідні ним керуючі простори синтаксичних структур “А” та “Б”.

Вихід : оцінка сумісності.

Основні кроки алгоритму:

1. Підставимо слово чи словосполучення сутності “а” в керуючий простір “Б” замість слова чи словосполучення сутності “б”. Отримаємо фрагмент КП “Б” з сутністю “а”.

2. Зробимо запит до тензору відповідних зв’язків для даного елементу.
3. Результатом запиту буде перше число, що характеризує оцінку семантичної сумісності першої сутності в контексті другої сутності.

Теорема 6. Алгоритм оцінки семантичного паралелізму працює за час *O(K)*, де *K* – кількість фактор-множин, обрана при факторизації тензора.

У **розділі 3.5** “Визначення слів-індикаторів як фактор-множин розкладених тензорів” розвивається ідея використання слів індикаторів для вирішення кореферентностей. В розробленому алгоритмі список слів індикаторів будується автоматизовано, на відміну від традиційного підходу, де список слів будується повністю вручну. Крім того було зменшено обчислювальну складність роботи порівняно з традиційним підходом, що працює за час *O(N)*, де *N* – кількість слів-індикаторів, наступним чином:

Теорема 7. Алгоритм оцінки наявності кореферентного зв’язку за допомогою фактор-множин працює за час *O(K)*, де *K* – кількість фактор-множин використаний в невід’ємній факторизації.

У **розділі 3.6** “Особливості тензорної факторизації розгорнутих та кільцевих альфа-бета зв’язків” проаналізовано алгоритм поблокового координатного спуску, що дозволяє провести невід’ємну факторизацію великих текстових корпусів частинами. Більшість алгоритмів тензорної факторизації вимагають розміщення всіх даних в оперативній пам’яті ПК, що неможливо при вирішенні даної задачі через їх великий об’єм. Наведений підхід позбавлений цього недоліку. В результаті експериментів було обрано 50 фактор-множин для кільцевих альфа-бета зв’язків, та 80 фактор-множин для розгорнутих альфа-бета зв’язків.

В **підрозділі 3.7** “Модифікація алгоритму опорних векторів для великих об’ємів негативних прикладів” було проаналізовано метод опорних векторів та тренувальну вибірку та модифіковано алгоритм

навчання, виходячи з їх особливостей. Метод опорних векторів – це набір схожих алгоритмів виду «навчання із вчителем» [16]. Ці алгоритми зазвичай використовуються для задач класифікації та регресійного аналізу. Метод належить до розряду лінійних класифікаторів. Основна ідея методу опорних векторів – перевід вихідних векторів у простір більш високої розмірності та пошук роздільної гіперплощини з максимальним проміжком у цьому просторі (Рис. 4.). Дві паралельні гіперплощини будуються по обидва боки гіперплощини, що розділяє вказані класи. Роздільною гіперплощиною буде та, що максимізує відстань до двох паралельних гіперплощин.

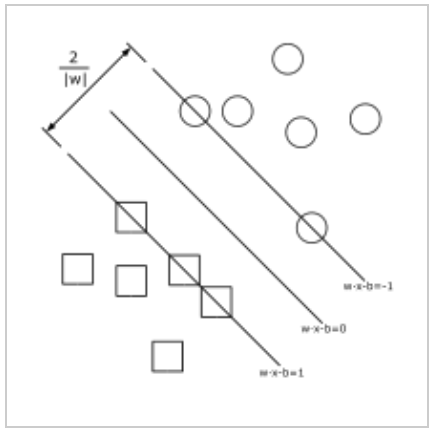


Рис. 4. Оптимальна січна гіперплощина для метода опорних векторів, побудована на точках з двох класів. Найближчі точки до паралельних гіперплощин називаються опорними векторами

В використаній навчальній вибірці є близько 20 тисяч кореферентних пар сутностей та більше мільйона не кореферентних. Головна ідея модифікації алгоритму полягає в генеруванні тренувальної вибірки під час процесу навчання. В кожний момент часу є навчений класифікатор, за допомогою якого алгоритм знаходить кореферентні пари сутностей в наступному не опрацьованому реченні вихідної тренувальної вибірки. В цільову тренувальну вибірку додаються тільки ті негативні приклади, результат класифікації яких не збігається з еталонним. Ця

евристика відсікає більшість негативних прикладів, що ніколи не стануть опорною точкою методу навчання.

У **розділі 3.8** “Простір ознак машинного навчання” побудовано простір з 28 різноманітних ознак пари сутностей для побудови класифікатора методом опорних векторів. З них 5 ознак визначені як оцінки кореферетності пари сутності алгоритмами семантичного, семантико-синтаксичного паралелізму та використання слів індикаторів. 15 ознак простору використовують результати порівняння морфологічного властивостей слів пари сутностей. 2 ознаки засновані на аналізі іменованих сутностей та 1 ознака обчислюється з Дерева залежностей. В процесі навчання для кожної кореферентної та не кореферентної пари сутностей обчислюються всі 28 ознак. В результаті ми отримуємо точку в 28-вимірному просторі ознак, що належить до відповідного класу. Метод опорних векторів будує гіперплощину, що розділяє точки класів кореферентних та не кореферентних сутностей.

У **розділі 3.9** “Оцінка результатів роботи алгоритму пошуку кореферентностей” детально описано процедуру тестування CoNLL’2011. Точність роботи алгоритмів пошуку кореферентностей визначається за допомогою експерименту. Розмічений корпус конференції складається з 3 частин : тренувальна вибірка, вибірка для налаштування, та офіційна тестова вибірка. Під час розробки системи згідно до процедури тестування використовувалася лише тренувальна вибірка для навчання класифікатора та вибірка для налаштування для визначення порогових значень для прийняття рішення. Розмітка тестової вибірки має об’єм 25Мб. Великий розмір та її варіативність забезпечує покриття текстів різної тематики.

Процедура визначення точності на тестовій вибірці полягає в оцінці правильності визначених кореферентних кластерів сутностей. Оцінка точності роботи системи за мірою MUC обчислюється за допомогою наступної формули:

$$MUC = \frac{\sum_{i=0}^{n'} |C'_i \cap C_{j(i)}| - 1}{\sum_{i=0}^{n} |C_i| - 1}, j(i) = argmax(|C'_i \cap C_j|), j = 0,1,\ldots,n,$$

де C_i – множина сутностей в і-тому кластері ручної розмітки,

C'_i – множина сутностей в і-тому кластері отриманий за допомогою досліджуваної системи,

j(i) – номер кластера досліджуваної системи, що має найбільший перетин з кластером і з ручної розмітки. Вважається, що кластер з найбільшим перетином є відповідником кластера з ручної розмітки.

Міра MUC для побудованої в роботі системи дорівнює 64.45%, що майже на 3.5% більше ніж попередній найкращий результат Stanford Deterministic Coreference Resolution System, що дорівнює 61.03%.

Розділ 1. Аналіз сучасного стану галузі вирішення кореферентностей

1.1 Постановка задачі вирішення кореферентностей

В наш час бурхливого розширення сфер застосування інформаційних технологій задачі обробки текстів природною мовою набувають великого значення в науці, економіці та інших сферах життя суспільства. Автоматичний переклад тексту допомагає людям з різних країн розуміти один одного без докладання значних зусиль та витрат часу. Компанії-гіганти, такі як Google та Facebook застосовують методи комп'ютерної обробки текстів для покращення точності цільової реклами, аналізуючи переписку та пошукові запити користувачів. Фірми, що займаються соціологічними дослідженнями, мають змогу оцінювати ставлення людей до певних осіб чи подій на основі автоматичного аналізу публікацій у пресі, залишених користувачами коментарів, повідомлень на форумах. І це далеко не повний перелік практичних задач, для розв'язання яких використовуються методи та алгоритми комп'ютерної лінгвістики.

Для того, щоб автоматичний переклад текстів одразу видавав результат, який не потребує кропіткої додаткової вичитки, необхідно, щоб комп'ютер був здатний розуміти текст на рівні людини. Для оцінки складності задач штучного інтелекту аналогічно до класу NP-повних задач, введено клас AI-повних (**A**rtificial **I**ntelligence) задач, повне вирішення яких вимагає побудови штучного інтелекту, близького до рівня інтелектуальних здібностей дорослої людини.

Задача повного розуміння текстів є вкрай складною, тому її розбивають на підзадачі – розуміння значень окремих слів, розуміння певних типів зв'язків між словами та їх залежностей, розуміння речень природною мовою в контексті всього текста в цілому чи в контексті знань людини про конкретну предметну область. Однією з систем, що намагається підійти до вирішення цих задач є онтологія WordNet [8]. Основним об'єктом цієї системи є семантична мережа. Елементами WordNet є синсети (synset, synonymous set, множина синонімів). Синсети цієї системи складають словник семантичних значень, що описують певний об'єкт, дію чи ознаку. Окремі слова не можуть складати словник семантичних значень, оскільки вони можуть вживатися в різних значеннях, не враховуючи навіть омонімію слів. Так наприклад слово брат може вживатися в значенні родича, сина мого батька, або ж може вживатися в значенні друга, знайомого. В обох випадках я навів множину слів синонімів до слова брат, якими можна замінити це слово для уточнення його семантичного значення. Саме цей принцип використовується для задання значення слова за допомогою синсету. Перший синсет - родич, син мого батька, брат; другий синсет - друг, знайомий, брат, бро. Задача word sense disambiguation, WSD - полягає в співставлення слову в реченні конкретного синсету в залежності від контексту. Кожен синсет онтології був написаний лінгвістами вручну,

тому її розмір обмежений та не включає багатьох понять, що зустрічаються в природномовних текстах.

Іншою складовою частиною мережі WordNet є зв'язки між словами. Зв'язки бувають декількох типів : гіперніми і гіпоніми, мероніми, голоніми та антоніми. В онтології є лише базові зв'язки між поняттями, що не залежать від контексту в якому вживається даний синсет, та від самого слова. Тобто в онтології відсутні специфічні типи зв'язків в які можуть входити деякі сутності.

Одним з більш складних зв'язків, що не може бути представлений в виді онтології, і має зміст для природномовного тексту є кореферентний зв'язок. Під кореференцією традиційно розуміють референційну тотожність імен [34]. Іншими словами, два слова чи словосполучення тексту, що вказують на одну і ту ж сутність, пов'язані кореферентним зв'язком.

Боже (небо)_1 голубеє

І (те)_1 помарніло.

Поглянув я на (ягнята)_2 —

(Не мої ягнята)_2!

Обернувся я на (хати)_3 —

(Нема в мене хати)_3!

(Т.Г. Шевченко "Мені тринадцятий минало", уривок)

В даному уривку з творчості Тараса Григоровича можна знайти 3 кореферентні зв'язки:

1. "небо" та "те"
2. "ягнята" та "не мої ягнята". Тобто ті ягнята на які поглянув ліричний герой, це саме не його ягнята. В реченні "Я поглянув на панських ягнят, і згадав що мені треба напоїти своїх ягнят", перше згадування ягнят не є кореферентним другому
3. "хати" - "нема в мене хати"

Серед кореферентних імен можна виділити наступні класи [33] :

1. дейктичні (я, ти, воно, це, цей , свій , все, той, тощо)
2. власні назви (Венеція, Наполеон, Чорне море)
3. іменникові групи - дескрипції - вирази з вершиною - власним ім’ям (цей стіл , король Франції)

Серед типів зв’язків кореферентності окремо виділяють анафоричні зв’язки - це зв’язок між словами та словосполучення, де перше слово належить до 1) дейктичної групи слів, а друге належить до 2 чи 3 класу.

Для прикладу розглянемо речення:

Якщо покласти морозиво в вогонь, то воно розтане.

Тут слово “воно” - це анафоричний займенник. “Морозиво” та “вогонь” - кандидати в антецеденти. Вирішення проблеми анафори в даному випадку означає знаходження коректної пари антецедент-анафора “воно-морозиво”.

Продемонструємо важливість знаходження кореферентного зв’язку для коректного автоматичного перекладу на наступному прикладі. Нехай необхідно перекласти наступні речення з англійської мови на українську:

If you put ice cream to fire it will melt.

Слово “it” в англійській мові не вказує на рід, і перекладається українською як одне з слів “він”, “вона”, “воно” в залежності від роду. Тому для перекладу треба встановити на яку сутність вказує слово “it”. Ми робимо припущення, що це слово має бути вжито десь в цьому ж тексті, хоча можна придумати текст, де таке слово треба буде прочитати між рядків і в тексті його взагалі не буде. В даному реченні для анафоричного займенника “it” є два вірогідних кандидати в антецеденти - “fire” та “ice cream”. Якщо алгоритм правильно зв’яне пару “ice cream” та “it” кореферентним зв’язком, то переклад буде коректним:

Якщо покласти морозиво в вогонь, то воно розтане.

В іншому випадку при перекладі втратиться зміст:

Якщо покласти морозиво в вогонь, то він розтане.

Побудова моделі формального представлення семантичних зв'язків між словами та використання цієї інформації для знаходження більш складного кореферентного зв'язку є метою даної роботи.

1.2 Алгоритми вирішення проблеми анафори

Вирішення проблеми знаходження кореферентностей почалося з часткового випадку - вирішення займенникової анафори.

Один з перших алгоритмів для вирішення займенникової анафори був запропонований Джері Хобсом [10, 1978]. В статті пропонується 2 підходи до вирішення проблеми анафори. Перший наївний алгоритм обходить дерево виводу спеціальним чином, аналізуючи іменники коректного роду та числа. Алгоритм не працює в всіх можливих випадках правильно, проте на наборі тестових даних, що складається з декількох сотень уривків публікацій він знаходить досить багато зв'язків. Наведемо алгоритм обходу дерева:

1) Почнемо з вузла NP, що безпосередньо включає в себе займенник
2) Піднімаємся вгору по дереву, доки не зустрінеться інший вузе NP або S. Цей вузол будемо називати X, а шлях до нього - p.
3) Обходимо зліва на право всі гілки піддерева, що починається в X та знаходяться лівіше шляху p. Вважаємо кандидатами в антецеденти всі NP вузли, що мають вузол NP чи S між даним вузлом та X.
4) Якщо X - це найвища вершина типу S в цьому реченні, тоді обходимо попередні речення аналогічно як в кроці 3). Інакше переходимо до пункту 5)
5) Від поточного вузла X піднімаємося вище по дереву до першого вузла типу NP чи S. Цей вузол буде новим поточним значенням X, а шлях до нього новим значенням p.
6) Обхід аналогічний кроку 3.

7) Обходимо зліва на право всі гілки піддерева, що починається в X та знаходяться правіше шляху p, але не нижче довільного вузла NP чи S. Будь-який вузол типу NP вважаємо кандидатом в антецеденти.

8) Повертаємось до кроку 4)

В іншому алгоритмі Хобса продемонстровано, як за допомогою складної системи семантичного аналізу англомовних текстів можна знаходити пари антецедент-анафора з більшою точністю. Робота цього алгоритму була продемонстрована на декількох реченнях.

Іншим оригінальним алгоритмом вирішення проблеми анафори є алгоритм Шалома Лаппіна та Герберта Лісса [18, 1994]. Алгоритм застосовується до деякого синтаксичного представлення, що породжений парсером граматик слотів МакКорда [13, 1990] і спирається на міри характерних особливостей, отриманих з синтаксичної структури і простої динамічної моделі станів підвищеної уваги. Як і синтаксичний парсер, алгоритм реалізований на мові Пролог. Автори протестували його широко на текстах комп'ютерних посібників і провели сліпий тест на штучному тексті, що містить 360 входжень займенників. Алгоритм успішно ідентифікує антецедент для кожного займенника в 86% випадків. Внесок кожного з компонентів алгоритму був розглянутий та оцінений окремо. Алгоритм було порівняно з іншими відомими підходами до вирішенням анафори. Зокрема, алгоритм Хобса був реалізований в рамках граматики слотів, і протестований на тому ж наборі тестів. Алгоритм Лаппіна та Ліса показав на 4% кращий результат, ніж алгоритм Хобса. Було обговорено взаємозв’язок алгоритму до центруючого алгоритму, а також до моделей вирішення анафори, які покладаються на різні додаткові фактори для ранжування кандидатів.

Один з найбільш розвиненим на даний момент алгоритмом вирішення проблеми анафори є алгоритм Міткова [15, 2005]. Даний алгоритм розвиває ідею Лапіна і Ліса, проте вводяться нові критерії

оцінки: синтаксичний паралелізм, повторюваність кандидата, схоже положення, підмет, доповнення, часте згадування. Також з'являються штрафні критерії, наприклад у випадку не визначеної граматичної ролі слова. Крім власне алгоритму, робота Міткова цікава оглядом та систематизацією існуючих підходів. Так всі критерії поділені на дві категорії : обмеження та преференції. Серед обмежуючих критеріїв виділяються:

- Узгодження роду та числа. Анафора та антецедент обов'язково мають мати однаковий рід та число.
- Теорії синтаксичних зв'язків. Задають деякі синтаксичні правила які обмежують кількість кандидатів в антецеденти. Наприклад правило "Не займенникова NP група не може перекриватися в посиланнях з будь-якою NP групою, що керує нею". He_i told them about $John_j$.
- Семантична узгодженість. Якщо деякому семантичному обмеженню задовольняє анафора, то їй має задовільнять і антецедент. Приклад:

Vincent removed the diskette from the $computer_i$ and then disconnected it_i. (Дискета не може бути відключеною. Вона може бути вийнятою, але це різні дії)

Vincent removed the $diskette_i$ from the computer and then copied it_i. (Комп'ютер не може бути скопійованим)

В статті [27] під вимоги розпізнавання анафори було видозмінено відомий класифікатор. Для прийняття рішення алгоритмом використовувалася система алгоритмів опорних векторів (Support Vector Machines) [4]. Ці алгоритми добре зарекомендували себе при вирішенні великої кількості практичних завдань, пов'язаних з аналізом даних. В якості прецеденту виступає пара анафора-антецедент, яка належить до одного з двох класів залежно від наявності в ній референції. Очевидно, що серед гіпотетичних антецедентів, що передують даній анафорі, знайдеться, принаймні, один, пов'язаний з нею референцією. Для того щоб уникнути

неоднозначності, можна вважати, що такий об'єкт єдиний (справді, зв'язаність анафори хоча б з одним антецедентом, наприклад, з найближчим, в силу транзитивності відносини автоматично гарантує її зв'язність з усіма іншими гіпотетичними антецедентами, пов'язаними з даними). Для вирішення анафори використовувався наступний алгоритм:

1. Фіксуємо чергову анафору і присвоюємо $n = 1$.
2. Переходимо до наступного, n-го гіпотетичного антецеденту, погодженим в роді і числі, розташованому раніше в тексті починаючи з найближчого до розглянутої анафорі.
3. Для пари анафора-антецедент запускаємо метод опорних векторів n - го рівня.
4. Вихід методу опорних векторів y_n перетворимо в оцінку ступеня впевненості в тому, що дана пара пов'язана зв'язком референції за такою формулою

$$p_n = \frac{1}{1 + \exp(\lambda y_n)}$$

5. Збільшуємо n на одиницю. Якщо n <N, то перехід до кроку 2, інакше - до кроку 6.
6. Отримані оцінки ступеня впевненості множимо на релаксаційні коефіцієнти r_n, що відображають апріорні знання про статистичний розподіл реферованих пар

$$s_n = p_n r_n$$

7. Пов'язуємо розглянуту анафору з m-тим антецедентом, де $m = \text{argmax}_n\, s_n$. Потім перехід до кроку 1.

Метод опорних векторів n-го рівня навчався за такими парами анафора-антецедент, в яких між ними було рівно *n-1* іменників або займенників, узгоджених з анафорою в роді і числі (тобто *n-1* потенційних

антецедентів для розглянутої анафори). Пари, в яких мала місце референція, були віднесені до першого класу, а решта - до другого. Далі для такої навчальної вибірки запускався стандартний метод опорних векторів, який розділяв два класи. Для нової пари об'єктів з вектором ознак x, вихід методу визначається як

$$y(x)\,,$$

де w_i, w_0 - ваги алгоритму, які знаходяться в процесі навчання,

$K(x', x'')$ - функція ядра, що забезпечує нелінійність одержуваної границі між класами. У експериментах використовувався найбільш поширений вид функції $K(x', x'') = \exp(-\gamma\|x' - x''\|^2)$, де параметр γ знаходився під час процесу крос-валідації [24].

Виходячи з апріорних знань відомо, що найчастіше анафори реферуються з найближчим гіпотетичним антецедентом і що ймовірність реферованих падає з ростом n. Як коефіцієнтів релаксації були обрані регуляризовані частоти появи реферованих пар n -го рівня щодо всіх пар n-го рівня, обчислюється за формулою

$$r_n = \theta\frac{1}{N} + (1-\theta)v_n$$

Коефіцієнт регуляризації θ також підбирався за допомогою крос-валідації. З урахуванням кількісного розподілу реферованих анафор, значення N в цьому дослідженні було вибрано рівним 4. Частка анафор, найближчі гіпотетичні антецеденти яких, знаходяться поза зоною аналізу становить менше 10%.

1.3 Алгоритм вирішення кореферентностей

Один з перших запропонованих алгоритмів для знаходження кореферентностей в нерозмічених текстах без обмежень був запропонований в 2001 році групою вчених в Сінгапурі [19]. Метод

заснований на машинному навчанні. Для кожної пари слів та словосполучень *(i,j)*, що перевіряються на наявність кореферентного зв'язку між ними, визначаються 12 критеріїв. Більшість з них є бінарними (значення істина та хиба), інші є тернарними (істина, хиба та невідомо). Наведемо ці критерії :

1. Слово *i* та *j* знаходяться в одному реченні
2. *i* - це не займенник
3. *j* - це не займенник
4. *i* та *j* не збігаються
5. *j* не визначена іменникова фраза
6. *j* не демонстративна іменникова фраза
7. *i* та *j* знаходяться в множині
8. *i* та *j* вказують на особу (приймає тернарне значення)
9. *i* та *j* є словами чоловічого роду
10. Тільки *i* є власним ім'ям
11. *j* - це не аліас *i*, тобто не вказують на одну і ту ж іменовану сутність
12. *j* не йде одразу за *i* в тексті.

Для побудови класифікатора формується тренувальна вибірка з пар слів, для яких відомо, чи вони є кореферентним. Також для них відомо значення 12 сформованих критеріїв. [17].

В статті [1] представлено цікаве архітектурне рішення розв'язку проблеми знаходження кореферентних зв'язків в вигляді поєднання декількох "решіт". Кожне решето реалізовує деяких алгоритм оцінки пари слів чи словосполучень на наявність кореферентного зв'язку. Результатом застосування решета до пари слів може бути 3 значення :

1. Алгоритм встановив, що пара слів не може буде кореферентними. Однією з реалізацій решета, що дає досить високу точність відсіювання є перевірка роду слів. Якщо наприклад одне слово має жіночий рід, інше - чоловічий, то ці слова не можуть бути кореферентними. З іншого боку,

тотожність синтаксичних властивостей не є достатньою умовою наявності кореферентного зв'язку. Проте навіть такий простий з першого погляду алгоритм не дає 100% точності, оскільки на етапі попереднього синтаксичного аналізу могли бути помилки і рід не був встановлений точно.

2. Алгоритм встановив, що пара слів є кореферентними. Прикладом такого решета може бути перевірка на тотожність власних назв. Нехай перше словосполучення - "Т. Г. Шевченко", інше слово - "Шевченка". Просто звівши дані власні назви до нормальної форми і порівнявши їх посимвольно, можна прийти до висновку що більш за все мова йдеться про одну і ту ж особу.

3. "Не знаю" є найбільш частим результатом застосування решета до пари слів. Для наведених вище прикладів в першому випадку такий результат буде отримано, якщо обидва слова мають однаковий рід. В другому випадку до такого результату призведе пара слів, хоча б одне з яких не є власною назвою.

Робота Лі [20] є розширенням запропонованого вище підходу. В ній вводяться додаткові реалізації решіт, а також алгоритм виявлення сутностей, що є кандидатами на наявність кореферентного зв'язку. Ця робота є дуже цікавою, оскільки вона продемонструвала найкращі результати при тестуванні за допомогою системи Conll-2011 [6]. З моменту першої публікації система була неодноразово покращена авторами статті та інтегрована до системи Stanford CoreNLP [22]. Розглянемо принцип роботи кожного з решіт більш детально:

1. *Решето обробки дискурсу.* Даний метод дозволяє краще оброблювати тексти з прямою мовою. Виділяються всі репліки, на основі знаків пунктуації - лапок в синтаксичних структурах цитування та тире в діалозі. Решето порівнює авторів реплік до всіх сумісних займенників. Спочатку виділяються автори реплік в тексті. Для не діалогів

використовується проста евристика, що знаходить осіб, які виконують деяку дію пов’язану з донесенням інформації, в тому ж самому реченні або поряд з реченням з цитатою. В випадку діалогів, інформація про користувача має бути розміченою в даних.

Виділення авторів реплік дозволяє реалізувати наступні евристики для відсіювання:

- Всі займенники “Я”, що цитуються одним і тим же автором, є кореферентними
- Всі займенники “Ти”, що цитуються одним і тим же автором, є кореферентними
- Автор та “Я” в його цитатах є кореферентними

Для прикладу, “я”, “мій”, та “вона” в наступному реченні є кореферентними:

Вона сказала : “Я проголосувала за Миколу, тому що його ідеї були найбільш близькими до моїх”

Крім названих вище правил, також вводяться наступні правила, що забороняють поєднувати групи кореферентних слів:

- Автор та всі слова, що не є “я” в висловлюванні цього автора не можуть бути об’єднані кореферентним зв’язком іншими решетами.
- Два вживання прийменника “я” (або “ти”, або “ми”), що вживаються різними авторами не можуть бути кореферентними
- Два різних особових займенника, що вжиті в цитатах того ж автора не можуть бути кореферентними.
- Іменникові фрази не можуть бути кореферентними з “я”, “ти” та “ми” в тому ж самому висловлюванні.
- В діалозі “ти” може бути кореферентним тільки автору попередньої цитати.

2. Повне посимвольне збігання двох сутностей

3. Слабке посимвольне збігання. Це решето вважає дві іменникові фрази кореферентними, якщо їх частини, утворені відкиданням всіх слів, що йдуть після головного слова, є однаковими. Наприклад, “Клінтон” та “Клінтон, чий термін закінчується в січні” є кореферентними, з головним словом “Клінтон”.

4. Решето псевдонімів. Два кандидата вважаються кореферентними, якщо вони:

- в Вікіпедії ці слова вказують на одну і ту ж статтю
- в WordNet зустрічаються в одному і тому ж самому синсеті
- зустрічаються в Freebase записі в полі “name” чи “alias”

Для прикладу це решето коректно визначає “America Online” та “AOL” як псевдоніми.

Також були спроби використати категорії Вікіпедії, але в результаті отримали погіршення результатів.

5. Решето послабленого порівняння головних слів

Це решето позначає дві групи слів, з власним іменником в якості головного слова, кореферентними, якщо їх головні слова збігаються та виконуються наступні твердження:

- Немає невідповідності місця події. Два кандидати на кореферентність не можуть мати різних сутностей місця, інших власних назв, чи просторових деталізацій. Наприклад, “Лебанон” та “південний Лебанон” не є кореферентними.
- Немає числових невідповідностей. Друге словосполучення не може мати числа, що не входить до антецеденту, тобто “люди” та “близько 200 людей” не є кореферентними.

До наведених вище правил, було додано ще декілька щоб отримати кращу якість розпізнавання.

- Відстань до займенника. Відстань між займенником та антецедентом не може бути більшою за 3.

- “Bare plurals”. В англійськомовних текстах, це іменникові групи, в яких головне слово немає артикля. Такі словосполучення є загальними фразами та не можуть мати кореферентного антецедента.

6. Решето лексичних ланцюгів

Це решето позначає дві іменникові групи як кореферентні, якщо вони пов’язані лексичним ланцюгом з типом зв’язків гіпернімії та синонімії в онтології WordNet. Для слів не знаходиться конкретне семантичне значення в WordNet, а використовуються всі можливі значення, але обмежуються значеннями в трьох останніх реченнях та ланцюги максимальною довжиною чотири. Це решето коректно знаходиться зв’язок між “Британією” та “країною”, між “літаком” та “літальним апаратом”.

Для підвищення точності роботи решета, кандидати повинні задовольняти додаткові обмеження. Такі як узгодження числа, роду, істота - не істота, відсутність невідповідних чисел чи місць, а також не використовуються абстрактні сутності WordNet, крім ланцюгів що включають в себе “організація”.

1.4 Формальні моделі мови

1.4.1 Латентний семантичний аналіз.

Для розв’язання кореферентностей можна застосувати методи, що є розвитком латентна-семантичного аналізу (ЛСА). ЛСА - це метод обробки інформації природною мовою, що дозволяє проаналізувати взаємозв'язок між колекцією документів і термінами, які в них зустрічаються. Зіставляє деякі фактори (теми) всім документам і термам [11]. В основі методу латентно-семантичного аналізу лежать принципи факторного аналізу, зокрема , виявлення латентних зв'язків досліджуваних явищ або об'єктів. При класифікації / кластеризації документів цей метод використовується для вилучення контекстно-залежних значень лексичних одиниць за

допомогою статистичної обробки великих корпусів текстів. Однак більшість методів, що засновані на ЛСА, факторизують матриці, а значить знаходять тільки бінарні зв'язки. В природномовних текстах зв'язки в загальному випадку можуть бути складнішими, і охоплювати одразу групу слів. Розглянемо для прикладу наступні речення:

Мій годинник зупинився.

Я зупинився відпочити.

Словосполучення "годинник зупинився" та "зупинився відпочити" є коректними фразами. Проте "годинник зупинився відпочити" не має сенсу, і може використовуватися лише в казках. В роботі [23] запропонований підхід, що здатний аналізувати тернарне зв'язки між словами. Цей метод заснований на методі невід'ємного розкладання тензорів.

Частковим випадком невід'ємного розкладання тензорів є невід'ємний розклад матриць. Факторизація невід'ємних матриць (Non-negative matrix factorization, NMF) - це група алгоритмів в багатовимірному аналізі даних і лінійної алгебри, де відбувається факторизація матриці X в , як правило, дві матриці W і H: $nmf\ (X) \rightarrow WH$. Відмінною рисою NMF є таке обмеження - фактори W і H повинні бути невід'ємні, тобто всі їх елементи повинні бути ≥ 0. Фактично, NMF є удосконаленим варіантом сингулярного розкладання і створений саме для усунення слабких сторін SVD. Основними областями застосування NMF є: кластеризація текстів, спектральний аналіз даних і розпізнавання образів.

Досить популярним та простим алгоритмом для факторизації матриць є метод перемінних найменших квадратів (Alternating Least Squares, ALS) [16] :

```
W = abs(randn(m,k));
for i = 1 : maxiter
LS solve matrix equation W^TWH = W^TA for H
```

```
nonneg H = H. * (H >= 0)
LS solve matrix equation HH^TWT = HA^T
nonneg W = W. * (W >= 0)
end
```

Як бачимо, цей алгоритм зводиться до ітеративного виконання операцій вирішення матричного рівняння звичайним методом найменших квадратів та операцій занулення від'ємних елементів. Збіжність алгоритму до розв'язку наведена в [16].

1.4.2 Невід'ємна тензорна факторизація.

(від лат. tendere, «тягнутись, простиратися») — математичний об'єкт, що узагальнює такі поняття як скаляр, вектор, ковектор, лінійний оператор і білінійна форма. Вивченням тензорів займається тензорне числення. Нехай $I_1, I_2, \ldots, I_N \in \mathbb{N}$ – розміри тензора по кожному з напрямків. В роботі тензор $Y \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ порядку N визначається як N-вимірний масив з елементами $y_{i_1, i_2, \ldots, i_N}, i_n \in \{1,2,\ldots,I_N\}, 1 \leq n \leq N$. (Рис 1.1).

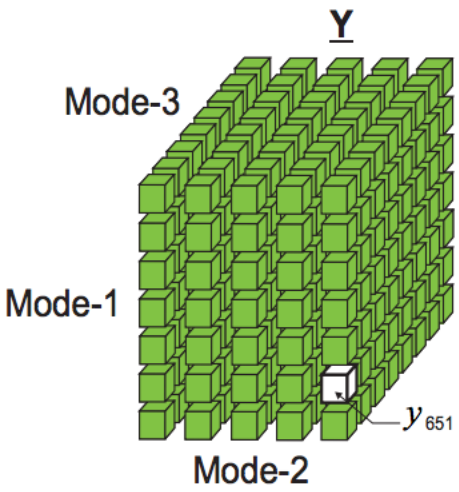


Рис. 1.1 Приклад тривимірного тензора розміру 7x5x8

Стовпчиком тензора будемо називати одновимірний фрагмент тензора, що утворений фіксуванням всіх координатних індексів крім одного.

Зрізом тензора будемо називати двовимірний фрагмент тензора, що утворений фіксуванням всіх індексів крім двох.

Зовнішній добуток тензорів $Y \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ та $X \in \mathbb{R}^{J_1 \times J_2 \times \ldots \times J_M}$ визначається за допомогою формули:

$$Z = Y \circ X \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N \times J_1 \times J_2 \times \ldots \times J_M},$$

де $\circ$ – символ операції зовнішнього добутку.

Звертаємо увагу на те, що тензор Z містить в собі всі можливі комбінації попарного добутку між елементами Y та X.

По-елементно операція зовнішнього добутку визначається як:

$$z_{i_1,i_2,\ldots i_n,j_1,j_2,\ldots j_m} = y_{i_1,i_2,\ldots,i_n} x_{j_1,j_2,\ldots j_m},$$

де

$z_{i_1,i_2,\ldots i_n,j_1,j_2,\ldots j_m}$, $y_{i_1,i_2,\ldots,i_n}$ та $x_{j_1,j_2,\ldots,j_m}$ – елементи тензорів Z, Y, Z відповідно

Особливими випадками є зовнішні добутки двох векторів $a \in \mathbb{R}^I$ та $b \in \mathbb{R}^J$:

$$A = a \circ b = ab^T \in \mathbb{R}^{I \times J}$$

та трьох векторів $a \in \mathbb{R}^I$, $b \in \mathbb{R}^J$ та $C \in \mathbb{R}^Q$:

$$A = a \circ b \circ c \in \mathbb{R}^{I \times J \times Q},$$

де $z_{ijq} = a_i b_j c_q$

Постановка задачі факторизації тензорів (Рис 1.2). Для даного тензора $Y \in \mathbb{R}^{I \times T \times Q}$ і додатнього індексу J знайти три матриці, що

називаються матриці навантаження або фактори, $A = [a_1, a_2, \ldots, a_J] \in \mathbb{R}^{I \times J}$, $B = [b_1, b, \ldots, b_J] \in \mathbb{R}^{T \times J}$, $C = [c_1, c, \ldots, c_j] \in \mathbb{R}^{Q \times J}$, що задовільняють наступним умовам:

$\|E\| \to min$

$$Y = \sum_{j=1}^{J} a_j \circ b_j \circ c_j + E \qquad (*)$$

Умова (*) в поелементній формі:

$$y_{itq} = \sum_{j=1}^{J} a_{ij} b_{tj} c_{qj} + e_{itq}$$

Індекс J в даній задачі називають кількістю фактор-множин.

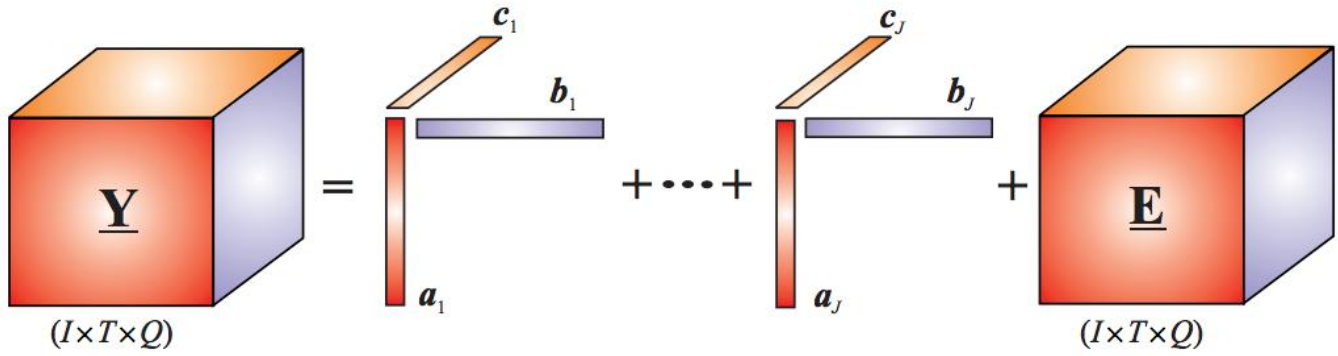


Рис. 1.2. Графічне представлення розкладення матриці в три фактори

Метод невід'ємного розкладання тензорів є досить гнучким, що дозволяє його використовувати не лише для вирішення задач комп'ютерної лінгвістики. В роботі [9] даний метод використовується для виділення шаблонів на зображеннях з метою їх кластеризації.

1.4 Висновки

В наш час системи комп'ютерної лінгвістики вирішують багато прикладних задач : автоматичний переклад текстів, природномовні інтерфейси до експертних систем, системи автореферування та створення портрету користувача для цільового показу реклами. Однією з підзадч

аналізу природномовних текстів є вирішення кореферентності. Покращення точності результатів роботи цієї підсистеми підвищує точність роботи всієї системи аналізу текстів в цілому.

Під кореференцією традиційно розуміють референційну тотожність імен [34]. Іншими словами, два слова чи словосполучення тексту, що вказують на одну і ту ж сутність, пов'язані кореферентним зв'язком.

Спочатку науковці розв'язували більш вузьку задачу – вирішення займенникової анафори. Один з перших алгоритмів для вирішення займенникової анафори був запропонований Джері Хобсом [10, 1978]. Іншим оригінальним алгоритмом вирішення проблеми анафори є алгоритм Шалома Лаппіна та Герберта Лісса [18, 1994]. Один з найбільш розвиненим на даний момент алгоритмом вирішення проблеми анафори є алгоритм Міткова [15, 2005]. В іншій статті [27] під вимоги розпізнавання анафори було видозмінено відомий класифікатор, що є досить цікавим результатом, оскільки система навчається на розмічених текстах. Недоліками всіх цих систем є вирішення тільки частини задачі. Ці системи не знаходять кореферентних іменникових фраз, хоча ця інформація є не менш важливою.

Алгоритми що вирішують більш загальну задачу з'явилися пізніше. Один з перших запропонованих алгоритмів для знаходження кореферентностей в нерозмічених текстах без обмежень був запропонований в 2001 році групою вчених в Сінгапурі [19]. Алгоритм використовує морфологічні властивості антецедентів для прийняття рішення та заснований на машинному навчанні. Найкращою системою в наш час є робота Лі, що заснована на ситах [20].

Система що продемонструвала найкращі результати не використовує машинного навчання та використовує лише універсальні формальні моделі тексту. Крім таких моделей тексту існують латентний семантичний аналіз та його багатовимірний варіант – невід'ємна

факторизація тензорів, що можуть бути модифіковані саме для таких задач. Безпосередньо виражальна здібність невід'ємної факторизації тензорів може бути вдосконалена за допомогою керуючих просторів природномовних текстів. В наступному розділі буде побудована така система невід'ємної факторизації тензорів керуючих просторів, що потім буде використана для вирішення задачі пошуку кореферентності.

Розділ 2. Побудова тензорної моделі керуючих просторів природномовних речень

2.1 Аналіз формальної моделі керуючих просторів природномовних речень

Формальним моделям синтаксичних структур природної мови приділяється значна увага в сучасній проблематиці систем штучного інтелекту та комп'ютерної лінгвістики. Це пов'язано з необхідністю створення дієвих програм генерації та аналізу пропозицій природної мови в експертних та інформаційних системах, у багатьох системах управління та прийняття рішень, у перспективних комп'ютерних системах майбутніх поколінь. Зараз також стає зрозумілим, що будь-яке просування в цій області тягне за собою прогрес у розумінні еволюційного процесу розвитку мови і мислення людини.

На неточному рівні основні синтаксичні конструкції описуються в класичних схемах граматики мови, що беруть початок в період античності і мало змінилися до теперішнього часу. Більш тонкі відносини керування між словами вивчаються у відомих лінгвістичних моделях дерев підпорядкування і систем складових, що з'явилися в 50-х роках двадцятого століття.

Очевидною перевагою перерахованих моделей є їх правильність - адекватне відображення тих чи інших специфічних характеристик синтаксичної структури тексту. Модель дерев підпорядкування орієнтована на керуючі зв'язку тільки за словами, а модель систем складових враховує тільки ієрархічне відношення вкладеності словосполучень в лінійній структурі тексту. Ці фактори лише наближено описують дійсні комунікативні властивості, що містяться в синтаксичних

структурах тексту. Тому були зроблені спроби побудови моделей, узагальнюючих кошти дерев підпорядкування і систем складових. Таким чином, уточнення моделей синтаксичних структур йде від уточнення керуючих зв'язків між словами і словосполученнями до уточнення зв'язків між групами синтаксичних одиниць. При цьому з неминучістю відбувається переміщення точки розгляду синтаксичних структур з лінійного порядку, нав'язаного послідовністю запису тексту, до складного простору, утвореного синтаксично пов'язаними групами об'єктів. При граничному переході (залишаючись у рамках синтаксису) приходимо до простору подання, не залежному від порядку запису тексту, а значить і від національної мови, виражає все предикативні і визначають відносини, що містяться в синтаксичних структурах. Цей простір і є керуючим простором синтаксичних структур природномовних речень.

В роботі запропонована алгоритмічна модель речення природної мови. На відміну від суто лінгвістичного підходу, речення розглядається як деякий динамічний обчислювальний рекурсивний процес, що розвивається в керуючому просторі, що пов'язує синтаксично згруповані частини пропозиції інформаційними каналами. Структура керуючого простору відображає семантику визначальних і предикативних конструкцій мови. Несподіваним виявилося те, що керуючі простору подібного виду вже розглядалися в якості обчислювальної моделі для рекурсивно-паралельних процесів. Реалізація цієї моделі орієнтована на ПАРКС-системи програмування, що підтримують концепцію рекурсивно-паралельного програмування в керуючих просторах. В роботі автори обмежилися концептуальним рівнем. Уточнення деталей та розробка алгоритму побудови КП були розроблені в даній роботі.

Обмежимося неформальними визначеннями.

Вважається, що в реченні слово α управляє словом β (β підпорядковане α), якщо β виступає безпосереднім уточненням

(коментарем) слова α. Таку залежність зображують орієнтованої дугою, спрямованої від слова α до слова β. Присудок - головне в реченні, інші слова завжди мають безпосередніх «господарів». Відношення безпосереднього підпорядкування слів утворює дерево підпорядкування речення. При графічному зображенні таких дерев зазвичай враховують порядок слів самого речення, тобто співвідносять саме дерево підпорядкування з його відображенням в лінійний запис. У термінах дерев підпорядкування можна успішно виражати багато стилістичних характеристик текстів.

Іншою відомою моделлю, що відбиває ієрархічну структуру частин пропозиції, є системи складових, які зручно задавати у вигляді дужкових структур. У дужки ставлять синтаксично зв'язкові словосполучення, наприклад, (Онєгін, (добрий (мій приятель))), (народився (на (берегах Неви))).

Очевидно, розглянуті вище моделі дають важливу інформацію про синтаксичній структурі речення. Однак обидві ці моделі мають ряд суттєвих недоліків. Дерева підпорядкування не враховують зв'язків між словосполученнями і синтаксично цілісними групами слів. У складних реченнях групи слів можуть служити для уточнення одного слова чи іншої групи слів, що важко висловити зв'язками дерев підпорядкування. Системи складових ігнорують спрямовані зв'язки і, крім того, не дозволяють описувати розривні словосполучення. Тому кожна з цих моделей не дає повного уявлення про синтаксичну структуру речення. Тому розглянемо більш загальну модель, названу ним системою синтаксичних груп. Синтаксична група - це підмножина слів (частина тексту), пов'язаних між собою згідно до певних критеріїв. Між синтаксичними групами встановлюється відношення безпосереднього підпорядкування. Дерева підпорядкування та системи складових можуть бути інтерпретовані як деякі різні види синтаксичних груп. Розкладання

речення на синтаксичні групи неоднозначно і залежить від вибору критерію об'єднання слів у синтаксичну групу. Особливо корисний для практики, на погляд автора роботи, аналіз з'єднання складнопідрядних речень. Викладену нижче алгоритмічну модель речення можна розглядати як розвиток і уточнення моделі синтаксичних груп при виборі критерію синтаксичної цілісності, що задається відносинами визначень словосполучень.

Дерева підпорядкування та системи складових, незважаючи на зазначені недоліки, володіють тією перевагою, що вони адекватно відображають істотні властивості синтаксичної структури речення. Тому модель, яку необхідно розробити, повинна давати одночасно інформацію, що міститься як в деревах підпорядкування, так і в системах складових.

Крім властивості давати імена об'єктів навколишнього світу, мова має фундаментальною властивістю виражати динамічні відносини, в які вступають об'єкти. Так, дієслово пов'язує в відношення об'єкти, що знаходяться у схемі дії цього дієслова, прикметник задає відношення об'єкта з самим собою. З синтаксичної моделі ми повинні знати, які частини речення пов'язані між собою через відносини і якого типу ці відносини. Існують лише два види синтаксичних відносин - предикативне і синтагматичне. Предикативне ставлення виражає залежність між синтаксичними об'єктами через поняття, що означає дію і зазвичай виражається за допомогою присудка. Синтагма - це поєднання двох синтаксичних об'єктів, з яких один є визначенням іншого. Крім того, в такому широкому розумінні синтагми повинні утворювати синтаксичні групи.

Адекватна модель синтаксичної структури повинна також відображати основну властивість рекурсивності мови - здатність розгортати власні визначення, тобто давати уточнення, характеристики, коментарі до своїх частин, а також будувати визначення визначень.

Віднесення подібних питань до сфери семантики не правомочні. Вони повинні вирішуватися на рівні синтаксичної моделі, так як проявляються на рівні загальної схеми, що не залежить від сенсу висловлювань.

Перш за все слід зупинитися на ролі керуючих зв'язків між словосполученнями. В роботі було порушено традиційний лінгвістичний підхід, при якому присудок вважається головним членом реченням, від якого поширюються зв'язки керування. Це йде від звички вважати ім'я функції більш головним, ніж її аргументи. Для зазначених вище цілей зручніше задавати синтаксичні відносини зв'язками генерації і передачі відносин. При цьому досягається більш точна характеристика керуючих зв'язків у порівнянні з традиційним підходом.

Якщо два об'єкти *A* і *B* вступають у відношення *C*, то ми виділяємо об'єкт (припустимо *A*), що викликає (ініціює, породжує) це відношення, і об'єкт, на який передається це відношення. Таким чином, виділяємо два види спрямованих зв'язків: від об'єкта-генератора відношення до самого відношення і від відношення до підлеглого об'єкту. Перший вид зв'язку називаємо «α-зв'язком (зв'язок генерування), другий - β-зв'язком (зв'язок поширення). Об'єкти *A*, *B* і відношенні *C* розміщуються в точках керуючого простору і тому графічне представлення відношення *C*, що зв'язує *A* і *B*, має вигляд, зображений на Рис. 2.1. Дієслова визначають відносини між об'єктами, тому в стандартній схемі простого речення: “іменник - дієслово - іменник» α-зв'язок направлений від першого іменника до дієслова і β-зв'язок направлений від дієслова до іменника-визначення. Розглянемо приклад: Дівчинка збирає квіти. Об'єкт дівчинка генерує відношення збирає і направляє його на об'єкт квіти. Тому α-β-структура цього речення має вигляд (Рис. 2.2). Розглянемо фразу: Красива дівчинка. Тут об'єкт дівчинка генерує унарне ставлення красива і передає це відношення собі ж (Рис. 2.3). Виникає кільцевий зв'язок, що характеризує визначення (Рис. 2.4).

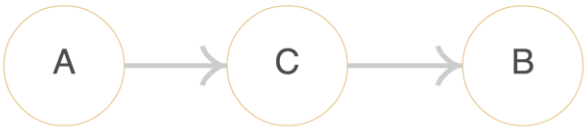


Рис. 2.1 Об’єкт *A* гарантує відношення *C*, що передається на об’єкт *B*

Рис. 2.2 Керуючий простір реченні “Дівчинка збирає квіти”

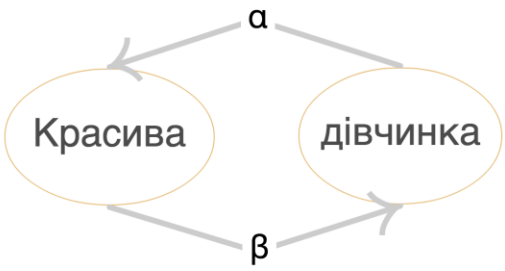


Рис. 2.3 Керуючий простір словосполучення “Красива дівчинка”

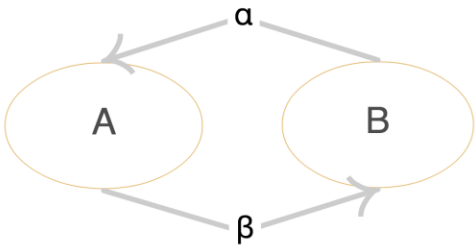


Рис. 2.4 Схематичне зображення кільцевого α-β-зв’язку

2.2 Постановка задачі побудови керуючого простору

Вхідними даними алгоритму є не розмічений текст природньої мови. Представлена реалізація алгоритму на текстах англійською мовою, проте безпосередньо описаний алгоритм не лімітований цією мовою. Обмеження викликане лише наявністю словників та навчених модулів на розмічених текстах англійською мовою.

Вихідними даними алгоритму є керуючий простір природномовних текстів в деякому представленні. Суть та бажані властивості КП були описані вище, тому для повного опису вихідних даних алгоритму

залишилося дати формальний опис цієї структури даних. Довільний керуючий простір сам по собі гарно формалізований, тому побудова структур даних для нього є тривіальною задачею. Наша задача є більш специфічною, так як ми працюємо з КП синтаксичних структур природної мови, що розширює наші знання про нього. Скориставшись цією інформацією можна спроектувати більш ефективні структури даних. Структури даних КП повинні відображати його зміст. Проведемо декомпозицію КП та формалізуємо його. Керуючий простір складається з елементів та зв'язків між ними.

Розглянемо представлення зв'язків КП. Як було вказано вище, зв'язки бувають 2 типів - *альфа* та *бета*. Окрім цього типовим для КП синтаксичних структур є ще один складений зв'язок - *кільцевий альфа-бета* зв'язок. Тому для зручності представлення даних виділимо його в окремий тип зв'язку. Інших атрибутів в керуючому просторі зв'язок не має.

Зв'язок об'єднує 2 елементи утворюючи новий елемент керуючого простору. Елементи утворені таким чином будемо називати *складеними*. Інформація про тип зв'язку, а також посилання на елементи, що зв'язуються, будуть зберігатися в таких елементах. Крім цього є елементи КП, що вказують на одне слово вхідного речення. Такі елементи будемо називати *базовими*.

Кожний елемент керуючого простору має деяке синтаксичне та семантичне значення. Синтаксичне значення базових елементів визначається частиною мови слова, якому він відповідає. Синтаксичне значення складених елементів визначається типом цього словосполучення - іменникова група, просте речення тощо. Частину мови і тип словосполучення будемо задавати деяким тегом. Набір можливих тегів скінченний, оскільки для кожної мови скінченний набір частин мови та типових синтаксичних структур, які його утворюють. Цей список тегів

описаний в Додаток В. Опис тегів дерева виводу. Представлення синтаксису буде розглянуте більш детально при побудові системи попереднього синтаксичного аналізу.

Семантичне значення елементу визначимо деяким головним словом. Це найпростіший підхід до представлення семантичного значення. Іншими способами представлення семантичного значення може бути посилання на синсет семантичної мережі WordNet. Проте відомі алгоритми вирішення задачі співставлення словам речення синсетів (в англомовній літературі ця задача відома під назвою “word sense disambiguation” чи “semantic indexing”) мають досить низькі показники f-міри на рівні 0.1 для не спеціалізованих текстів [54]. Зважаючи на це, на етапі побудови керуючого простору семантичне значення буде задаватися одним словом, але потім буде проведено розклад результату за допомогою невід’ємної факторизації тензорів, що дасть змогу згрупувати слова за схожістю контекстів в якому вони вживаються, а значить і виділити деякі більш загальні семантичні поняття на основі такого представлення керуючих просторів.

UML діаграма описаної вище структури даних зображена на Рис. 2.5.

Рис. 2.5 UML діаграма структури даних керуючого простору

Таким чином ми побудували структуру даних для представлення будь-якого керуючого простору синтаксичних в формі бінарного дерева(Рис. 2.6 б)). Зафіксуємо також порядок розгортання зв'язків КП. Вищий пріоритет мають альфа-зв'язки, потім бета-зв'язки, і нарешті кільцеві альфа-бета зв'язки мають найменший пріоритет.

Доведемо теореми про еквівалентність цих двох форм представлення КП.

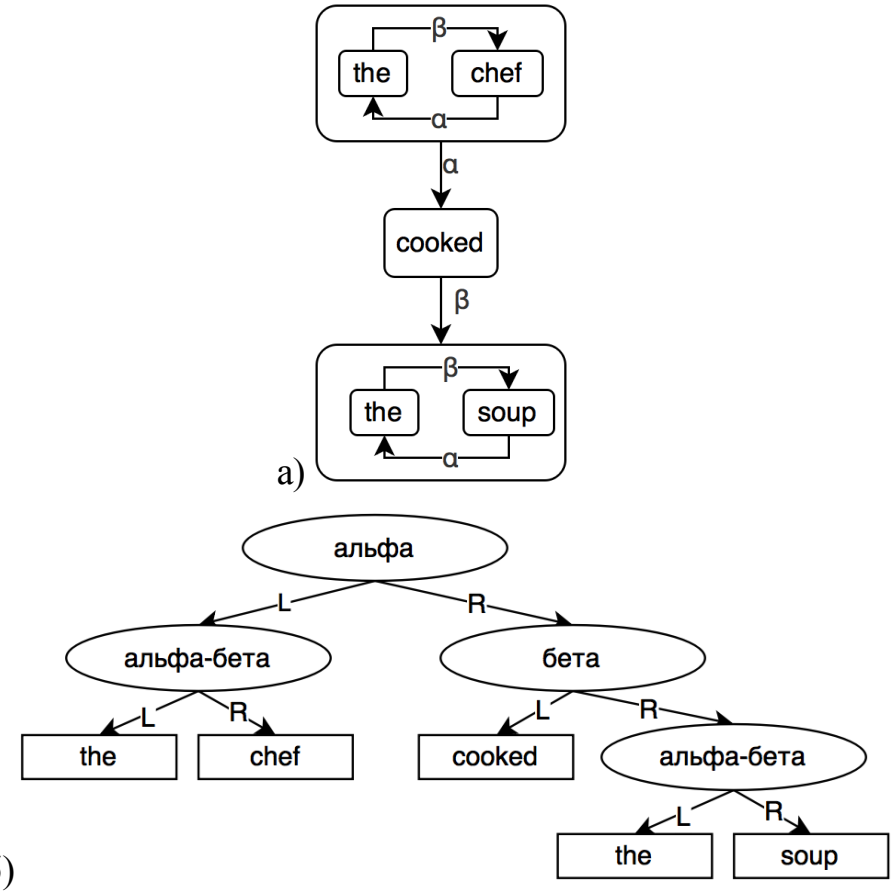


Рис. 2.6 а) Класичне представлення КП синтаксичних структур. б) представлення КП в формі бінарного дерева

Теорема 1. Для будь-якого керуючого простору синтаксичних структур природномовних речень існує єдине представлення його в вигляді бінарного дерева.

Доведення.

Нехай дано КП в класичному представленні А. Побудуємо для нього КП в представленні дерева.

Крок 1. На найвищому рівні ієрархії знайдемо зв’язок з найвищим пріоритетом. Пріоритети в порядку спадання визначаємо наступним чином : альфа-, бета- та кільцевий альфа-бета зв’язок. Такий зв’язок існує та тільки один, оскільки на кожному рівні ієрархії є хоча б один зв’язок, та не більше одного зв’язку кожного з типів.

Крок 2. Розіб’ємо КП А на 2 частини видаливши знайдений зв’язок на два підпросторі A_1 та A_2. Таке розбиття існує та є єдиним.

Крок 3. Створимо вершину дерева КП в бінарному представленні з значенням типу зв’язку. Ліве та праве піддерево для цього дерева будуємо аналогічно для КП A_1 та A_2. Ліве піддерево відповідає КП від якого напрямлений зв’язок для типів зв’язку альфа- та бета-. Для кільцевого альфа-бета зв’язку ліве піддерево визначається КП від якого напрямлений бета-зв’язок пари альфа-бета зв’язку. Такий вибір вершин існує та єдиний.

Для кожного підпростора КП побудованого викликом з пункту 3 також виконується існування та єдиність, оскільки Крок 1-3 гарантують це. Таким чином для кожного КП буде побудоване єдине можливе його представлення в бінарній формі.

Теорема 2. Для будь-якого представлення КП в вигляді бінарного дерева існує єдиний КП в класичному представленні.

Доведення.

Нехай дано КП в бінарному представленні А. Побудуємо для нього КП в класичному представленні.

Крок 1. Розглянемо корінь бінарного дерева. Він має деякий тип як значення вершини та 2 вказівники на піддерева. Корінь бінарного дерева єдиний та вказівники мають фіксований порядок.

Крок 2. Створимо зв’язок КП класичного представлення, що відповідає типу зв’язку вказаному в розглянутій на кроці 1 вершині. Цей зв’язок об’єднує 2 підпростори. Підпростір з якого виходить зв’язок будуємо даним алгоритмом рекурсивно виходячи з лівої синівської

вершини бінарного дерева, підпростір куди напрямлений зв'язок – з правої вершини піддерева. Такий рекурсивний виклик однозначний, та КП що їм побудовані також існують та єдині.

Таким чином запропонований алгоритм будує єдине можливе КП в класичному представленні, що і доводить існування та єдиність такого представлення.

2.3 Попередній синтаксичний аналіз

Більшість систем комп'ютерної лінгвістики містять в собі певні загальні етапи аналізу тексту. Розглянемо більш детально ті з них, що будуть використані для побудови керуючого простору. Так як реалізація ефективних алгоритмів реалізації кожного з етапів попереднього синтаксичного аналізу є темою окремої наукової роботи, тому ми не будемо зосереджуватися на методах їх вирішення. Ці етапи будуть розглянуті для того щоб проаналізувати додаткову синтаксичну інформацію яку можна отримати з не розмічених текстів, та враховувати типові проблеми роботи кожного з модулів, щоб конструювати алгоритм побудови керуючого простору природномовних речень як можна більш стійким до помилок модулів попередньої обробки.

Розгляд кожного з етапів попереднього синтаксичного аналізу буде включати постановку проблеми, можливий алгоритм вирішення її та типові проблеми, що виникають при її вирішенні. В якості програмної реалізації кроків попереднього синтаксичного аналізу було обрано Stanford Parser, тому буде розглянуто приклад виводу цього парсера.

2.3.1 Розбиття тексту на речення.

Самий простий підхід до вирішення - це розбити текст по розділовим знакам крапка ".", знак оклику "!" та знак питання "?". Проте якщо розбити на речення попереднє речення, то воно буде помилково

розбите на 4 частини. Схожі проблеми виникають при прямій мові. Тому задача не є такою тривіальною як вона виглядає с першого погляду.

Розглянемо фрагмент “Аліса в країні чудес” Льюїса Керола розбитий на речення за допомогою Стенфордського парсера:

1. There was nothing so VERY remarkable in that; nor did Alice think it so VERY much out of the way to hear the Rabbit say to itself, `Oh dear!
2. Oh dear!
3. I shall be late!'
4. (when she thought it over afterwards…

Як бачимо, Стенфордський парсер проводить розбиття тексту на речення за допомогою тривіального методу - розбивки по розділовим знакам без врахування контексту.

2.3.2 Розбиття речення на лексеми.

В широкому значенні, лексема — це слово як самостійна смислова одиниця, що розглядається в мовознавстві в усій сукупності своїх форм і значень. В програмуванні, лексема — послідовність машинних символів вихідного коду програми, що мають певне сукупне значення. В комп’ютерній лінгвістиці лексемою здебільшого виступають слова та розділові знаки. В кожній мові є деякі пограничні випадки між словом та розділовим знаком, які в кожній системі визначають по різному. Для української мови в якості прикладу можна навести слова з дефісом, такі слова як “російсько-український”, “боді-арт”. В даному випадку можна розглядати ці слова як єдина лексема, або ж 3 лексеми - 2 слова та знак. Також в деяких текстах символ дефісу та символ подані тією самою позначкою. Для англійськомовних текстів проблеми можуть виникати з апострофом, який вживається для скорочення слів (can’t = can not) та в присвійному відмінку іменників (the cat's tail). З одного боку розбиваючи на лексеми ми ще не знаємо частини мови слів, щоб вміти розрізняти ці випадки. З іншого боку здавалося б логічним що присвійна форма

іменника - це одна лексема, тоді як скорочення декількох слів через апостроф - це декілька лексем, кожна з яких відповідає скороченому слову, а також знак що вказує на скорочення. Так як ці випадки можуть бути визначені по різному, не можна комбінувати різні модулі з різних програмних систем.

Більш простий підхід до розбиття тексту на лексеми - це опис регулярних виразів, жорстких правил по яким проходить розбиття. Складніші алгоритми полягають в навчанні алгоритмів на деякому розміченому корпусі правильно розбитих речень на лексеми.

Розглянемо одне з речень твору “Аліса в країні чудес” Льюїса Керола розбитий на лексеми за допомогою Стенфордського парсера:

“How”, “funny”, “it”, “'ll”, “seem”, “to”, “come”, “out”, “among”, “the”, “people”, “that”, “walk”, “with”, “their”, “heads”, “downward”, “!”.

В цьому випадку парсер коректно розбив скорочення “it’ll” на дві частини “it” та “‘ll” (“will” в повній формі).

2.3.3 Морфологічний аналіз

На вхід даного модуля подається одне речення розбите на лексеми. На виході ми отримуємо співставлення кожної лексеми певному значення з словника морфологічних ознак. В англомовній літературі така задача називається Part-of-speech tagging, або скорочено POS-tagging. Морфологічні ознаки задаються тегом. Так наприклад більшість систем використовує тег NP для іменників, теги що починаються на V для різних форм дієслова (Таблиця 2.1).

В морфологічному аналізі можна виділити 2 підзадачі. Перша полягає в знаходженні для кожної лексеми списку всіх можливих морфологічних ознак, що виражаються даною лексемою. Алгоритмічно задача тривіальна. Проте вона вимагає наявності словник. Побудова такого словника полягає в тривалій кропіткій роботі лінгвістів певної мови. В даній роботі алгоритми були протестовані лише для англомовних

текстів внаслідок відсутності необхідних вільних словників для української мови, а також відповідно відсутності навчених вільних модулей морфологічного аналізу.

Таблиця 2.1. Розшифрування тегів модуля морфологічної розмітки.

Тег	**Опис**
PRP	Особовий займенник (Personal pronoun)
PRP$	Присвійний займенник (Possessive pronoun)
VB	Дієслово, початкова форма (Verb, base form)
VBD	Дієслово, минулий час (Verb, past tense)
DT	Артикль (Determiner)
NN	Іменник, однина (Noun, singular or mass)

Друга підзадача полягає в знаходженні морфологічних ознак виходячи з контексту вживання слова. Так слово англійської мови “cut” може бути дієсловом 1, 2 чи 3 форми, яке буде перекладатися як “відрізати”, “відрізав”, “надрізаний”. Також воно може бути іменником в значенні “надріз”. Множину всіх можливих значень було отримано вирішивши першу підзадачу по словнику. Сучасні алгоритми полягають в побудові класифікатора проаналізувавши деякий контекст - наприклад 3 попередніх слова. Якщо попереднє слово було модальним дієсловом - то cut може бути лише інфінітивом дієслова. Якщо попереднє слово було прикметником - то cut може бути лише іменником. В state-of-the-art системах процес визначення виглядає ще складнішим, оскільки морфологічні ознаки попередніх слів були визначені алгоритмом також не точно, а лише з деякою ймовірністю.

На Рис. 2.7 візуалізовано розбиття одного з речень з Аліси в країні чудес. Текст над лемами речення відповідає за тег словоформи частини мови. Словник відповідності тегів частинам мови наведений в Додатку А. Різним кольором позначені різні частини мови.

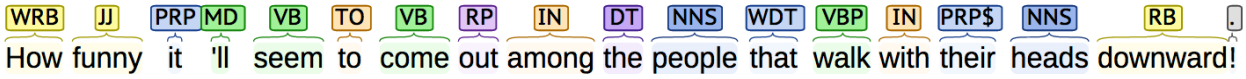


Рис. 2.7. Візуалізація морфологічного аналізу речення засобами Stanford Parser

2.3.4 Виділення іменованих сутностей

Інший етап попередньої синтаксичної обробки полягає в знаходженні власних назв (онімів).

Власні назви з'являються тоді, коли індивідуальне розрізнення набуває суспільної значущості. За денотатами розрізняють такі класи власних назв [97]:

- особові імена людей (антропоніми)
- назви географічних об'єктів (топоніми)
- назви космічних об'єктів (космоніми)
- найменування божеств, міфічних істот (теоніми, міфоніми)
- клички тварин (зооніми)
- назви організацій, виробничих і громадських об'єднань (ця група власних назв позначається терміном ергонімія і включає величезну кількість найменувань — від партій, товариств, заводів, вузів до кінотеатрів, кооперативів, магазинів, фірм тощо)
- назви відрізків часу, подій (це хрононімія, наприклад, Коліївщина, Друга світова війна)
- назви окремих предметів (хрематонімія, що включає власні назви матеріальних предметів — кораблів, ураганів, алмазів тощо та витворів духовної культури — заголовки творів, назви музичних п'єс, творів живопису, кінофільмів та ін.)

Аналіз типів власних назв є важливим для задач знаходження кореферентних зв'язків, оскільки всередині одного типу власні назви можуть мати різне написання, але назви різних типів не можуть бути кореферентними.

З Рис. 2.8 видно, що алгоритм виділення іменованих сутностей Stanford Parser працює коректно не в усіх випадках, так як Dinah в одному випадку було розпізнано іменованою сутністю, а в іншому - ні.

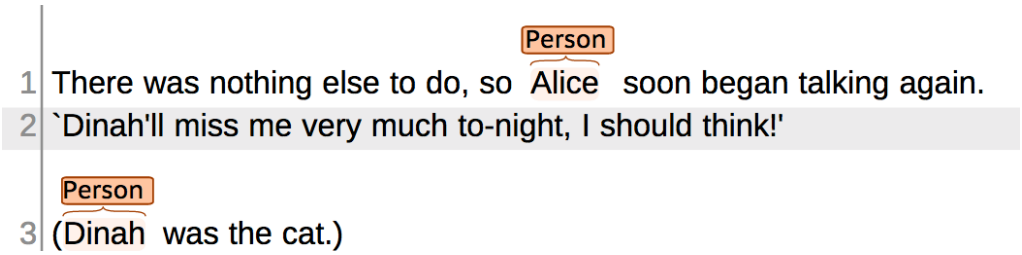


Рис. 2.8 Візуалізація прикладу роботи алгоритму знаходження іменованих сутностей Стенфордського парсера.

2.3.5 Алгоритм виділення іменованих сутностей на основі аналізу сторінок Вікіпедії

Проблема багатьох систем машинного навчання полягає не в недоліках самого алгоритму, а в недостатності розмічених даних, на яких було проведено тренування. А тому такі системи покривають не достатньо велику кількість ситуацій, що можуть зустрічатися при аналізі. У випадку виділення іменованих сутностей ця ситуація досить критична, оскільки появу власної назви, якщо вони не починаються з великої літери, неможливо передбачити не використовуючи знання про історичний розвиток людства. Тому вкрай важливо мати як можна більшу вибірку таких назв. Для таких цілей було обрано використовувати Вікіпедію. Вона покриває багато загальновідомих та специфічних для окремих галузей понять, які постійно поповнюються. Станом на 18 червня 2015 року англомовна частина Вікіпедії мала 4,894,879 статей, загалом по всім мовам 36,522,945 сторінок [81]. Назви статей іменують окремі визначені сутності. Серед них є слова загально значення, але якщо словосполучення

є назвою статті Вікіпедії, це означає що воно має якесь додаткове значення для людини, ніж просто значень окремих слів що в нього входять. Саме такі словосполучення - назви статей Вікіпедії - будуть формувати словник іменованих сутностей.

Процес знаходження виділених з Вікіпедії іменованих сутностей в довільному реченні природномовного тексту вимагає попереднього морфологічного аналізу та попередньої обробки словника іменованих сутностей. В загальному випадку речення розглядається як послідовність слів, і ми намагаємося в ньому знайти підпослідовності слів, що йдуть підряд без розривів, що знаходяться с словнику. При такому підході виникає проблема знаходження іменованих сутностей в відмінках, що відрізняються від називного. Для української мови це вимагає складного розбору речень. З англійською мовою проблема є тривіальною, так як правила узгодження слів є більш простими (наприклад прикметника і іменника). Для узгодження ми будемо визначати головний іменник в групі. Якщо головний іменник імені з назви статті Вікіпедії знаходиться в однині, то допустимо, щоб фраза в реченні була з головних іменником в множині та без артиклю “a” чи “an”.

2.3.6 Побудова дерева виведення

Дана частина попереднього синтаксичного аналізу була формалізована Ноамом Хомським [19]. Розглянемо коротко основні означення.

Формальна граматика або просто граматика в теорії формальних мов — спосіб опису формальної мови, тобто виділення деякої підмножини з множини всіх слів деякого скінченного алфавіту. Розрізняють породжуючі і аналітичні граматики — перші ставлять правила, за допомогою яких можна побудувати будь-яке слово мови, а другі дозволяють по даному слову визначити, входить воно в мову чи ні [102].

Алфавіт — скінченна множина символів. ε — порожній ланцюжок, слово, послідовність. Алфавітом є об'єднання алфавітів, перетин, різниця алфавітів. Нехай T — алфавіт, тоді:

- T^+ — множина усіх можливих послідовностей, що складені з елементів цього алфавіту крім порожньої послідовності ε.
- T^* — множина усіх можливих послідовностей, що складені з елементів цього алфавіту, будь-якої довжини. Отже: $T^* = T^+ \cup \{\varepsilon\}$
- T^k — множина усіх можливих послідовностей, що складені з елементів цього алфавіту, довжини не більше *k*.

Мова в алфавіті *T* це множина ланцюжків скінченної довжини в цьому алфавіті. Зрозуміло, що кожна мова в алфавіті *T* є підмножиною множини *T**.

Формальна граматика - це четвірка $G = \{N, T, P, S\}$, де:

- *T* — алфавіт термінальних символів, терміналів (від англ. terminate - завершитись). Термінальні символи є алфавітом мови.
- *N* — алфавіт нетермінальних символів, нетерміналів. $T \cap N = \emptyset$;

Нетермінали не входять в алфавіт мови.

- *S* — аксіома, спеціально виділений нетермінальний символ з якого починається опис граматики. $S \in N$
- *P* — скінченна підмножина множини $(T \cap N)^+ \times (T \cap N)^*$. Інколи визначають так: $\alpha \in (T \cap N)^+ \times T \times (T \cap N)^*, \beta \in (T \cap N)^*$.

Елемент (α,β) з множини *P* називається правилом виводу і записується у вигляді $\alpha \to \beta$. Таким чином, ліва частина правила не може бути порожньою. Правила з однаковою лівою частиною записують:

$$\alpha \to \beta_1 | \beta_2 | \dots | \beta_n,$$

$\beta_i, i = 1,2,\dots,n$ - називаються альтернативами правила виводу з ланцюжка α.

Синтаксичне дерево виводу слова ω в граматиці *G* — це впорядковане дерево, корінь котрого позначено аксіомою, в проміжних вершинах знаходяться нетермінали, а на кроні — термінали.

В випадку аналізу речень, словами в термінах Хомського виступають речення природномовного тексту. Літерами граматики Хомського виступають слова природномовного речення. Терміналами виступають POSTag'и морфологічних ознак слів. Нетерміналами виступають певні типові групи слів слів природномовних речень. Повний перелік нетерміналів з описом знаходиться в Додатку В.

Розглянемо дерево виводу речення "In another moment down went Alice after it, never once considering how in the world she was to get out again." побудоване Стенфордським парсером:

```
(ROOT
  (S
    (PP (IN In)
      (NP (DT another)))
    (NP (NN moment))
    (ADVP (RB down))
    (VP (VBD went)
      (NP
        (NP (NNP Alice))
        (PP (IN after)
          (NP (PRP it))))
      (, ,)
      (S
        (ADVP (RB never))
```

```
      (VP
        (ADVP (RB once))
        (VBG considering)
        (SBAR
          (WHADVP (WRB how))
          (S
            (PP (IN in)
              (NP (DT the) (NN world)))
            (NP (PRP she))
            (VP (VBD was)
              (S
                (VP (TO to)
                  (VP (VB get)
                    (PRT (RP out))
                    (ADVP (RB again)))))))))
(. .)))
```

В даному прикладі нетерміналом S виділені прості речення в складні складного, прослідковуються іменникові (NP) та дієслівні групи (VP).

2.2.7 Побудова дерева залежностей

Граматика залежностей - одна з формальних моделей, розроблених в рамках структурного синтаксису (поряд з граматикою складових). Являє структуру речень у вигляді ієрархії компонентів, між якими встановлено відношення залежності. Таким чином, структура речень розглядається в термінах головних і залежних вершин.

Сучасна граматика залежностей в значній мірі ґрунтується на ідеях Л. Теньєр [101].

Граматикою залежностей у вузькому сенсі називається теорія синтаксичної структури речень, в якій всі зв'язки в реченні розглядаються

як підрядні, вершиною речення визначається присудок або його знаменникова частина.

Стенфордський парсер генерує класичне дерево залежностей, в якому кожне слово, окрім головного слова речення, є залежним від деякого іншого слова. Класичні дерева будуються без поширення зв'язків на інші слова (no propagation) та без представлення їх в більш компактному виді (no collapsing).

Крім класичної граматика залежностей, Стенфордський парсер вміє будувати покращені Стенфордські залежності (СЗ, Stanford Dependencies, SD). Вони були розроблені з метою спрощення розуміння та ефективності використання людьми. Стенфордські залежності є трійками: ім'я відносини, головне слово і залежне слово. Стенфордські дерева залежностей будуються за принципами поширення зв'язків та побудови компактного їх представлення (propagation and collapsing). (Рис. 2.9)

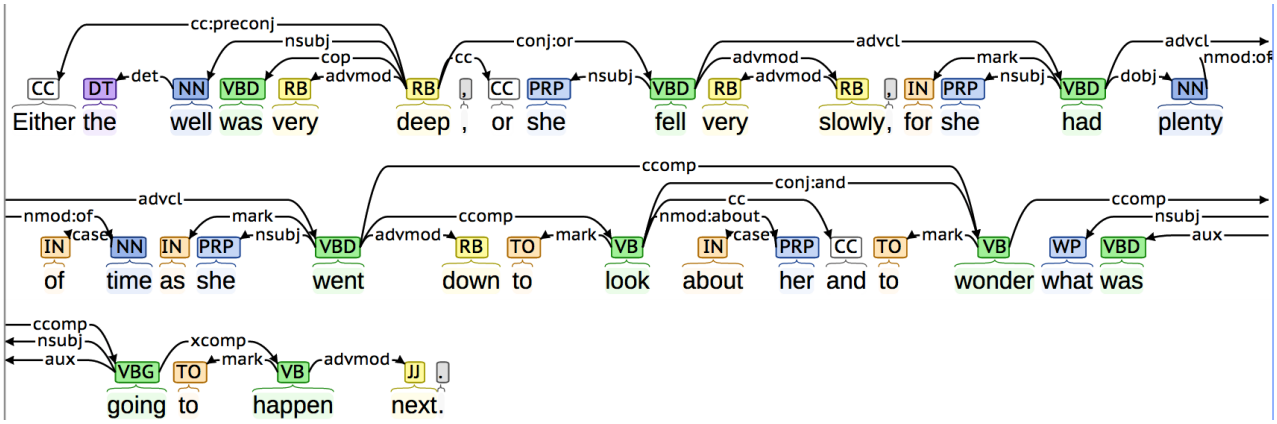


Рис. 2.9 Приклад покращеного стенфордського дерева залежностей

2.4 Алгоритм побудови шестивимірного тензора для задачі пошуку прихованих семантичних зв'язків в корпусах природномовних текстів

Пошуки покращеної моделі мови почалися з розробки шестивимірного представлення мови на основі Стенфордських залежностей. Розглянемо цей проміжний результат більш детально.

2.4.1 Мотивація побудови шестивимірного тензора

Основною робочою структуру розроблюваного алгоритму буде тензор, в якому буде зберігатися інформації про можливі комбінації слів. Розглянемо для прикладу тензор розмірності 2 (Рис. 2.10). На його вісях відкладені підмети і присудки. На перетині знаходиться зважений коефіцієнт їх сумісного входження. Так наприклад входження “піаніст грає” і “радіо грає” буде більше за нуль, оскільки такі комбінації підмета і присудка можуть зустрічатися в текстах. Можна розглянути інший тензор розмірності 2 (Рис. 2.11). На його вісях відкладені присудки та додатки, і в ньому входження пари “грає на піаніно” також буде додатнім.

Тензори розмірності три були введені через неповноту моделей заснованих на тензорах розмірності два. Для наведеного вище випадку, фраза “радіо грає на піаніно” буде коректною, оскільки її складові “радіо грає” та “грає на піаніно” є коректними. Розглянемо тензор розмірності 3 (Рис. 2.12). Його вісі - підмети, присудки та додатки. Трійка “радіо грає на піаніно” буде входити в нього з значенням нуль, оскільки в природномовних текстах такі фрази можуть зустрічатися тільки в казках, а їх ми не будемо брати до уваги. Тоді як “піаніст грає на піаніно” буде входити з вагою більшою за нуль.

Рис. 2.10. Приклад тензору “підмет-присудок”

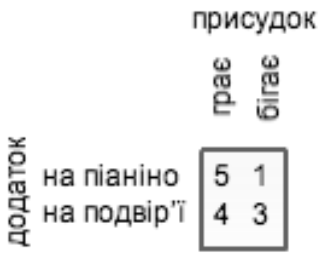


Рис. 2.11 Приклад тензору “присудок-додаток”

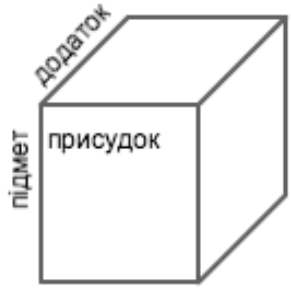


Рис. 2.12 Приклад тензора розмірності 3

Користуючись аналогічними міркуваннями можна користуватися тензорами ще більших розмірностей. Новими вісями можуть бути прямий об’єкт, не прямий об’єкт, модифікатор заперечення, кількості, часу, прийменник тощо.

Розмір Вікіпедії - 100 ГБ (10^{11} символів) тексту з форматування в форматі XML. Власне текст складає близько десятої частини розмітки XML, тому символів тексту складає 10^{10}. Кількість елементів тензора відповідає кількості граматичних основ, тобто кількості простих речень в тексті. Згідно до [24] середня довжина речення англійської мови складає 85 символів. Тому нам необхідно буде зберегти порядку 10^8 елементів тензора.

Іншою проблемою є знаходження корпусу даних, що будуть проаналізовані. Досить популярний для таких задач є корпус природномовних текстів з енциклопедії Вікіпедія. Її популярність пов’язана з доступністю, об’ємом, документованістю формату даних, уніфікованістю структури даних для всіх мов, наявністю додаткових тегів

та перехресних посилань. Враховуючи вищесказане, було використано саме Вікіпедію в якості корпусу.

2.4.2 Алгоритм виділення елементів тензора

Якщо розглянути слова тексту як вершини, а граматичні відношення Стенфордського парсеру - як ребра, то отримаємо граф (Рис. 2.13). Цей граф має вид дерева. Його вершиною буде головне дієслово головного речення. Наступна підзадача, яку треба вирішити - пошук та виділення шестірок в цьому графі. Слід пам'ятати, що прості речення в складеному утворюють незалежні граматичні одиниці, тому кожна граматична основа речення дозволяє отримати по одному елементу тензора, при чому ці елементи не повинні перекриватися по словам. Для цього треба скористатися деревом виводу речення.

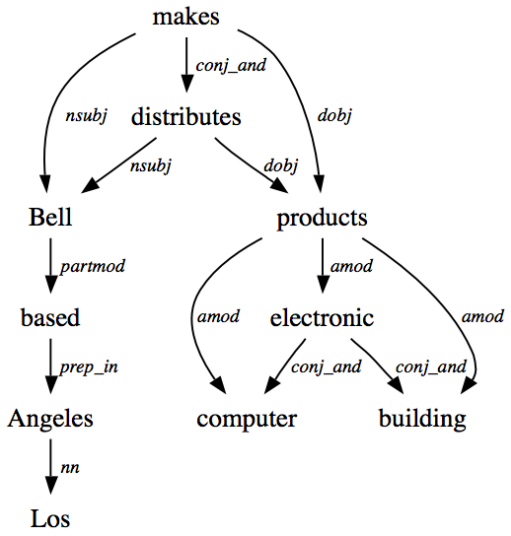


Рис. 2.13. Приклад графу дерева відношень

Для виділення слів розбитим по типам для кожного простого речення, що входить до складного, скористаємося рекурсивним алгоритмом:

```
function recursiveExtract(Node n) : Map<Type, Word>
// n - вершина простого речення
begin
    currentSentence ← ∅
```

```
    for i ← 1 until sizeof(n.leafs)
    begin
        // додаємо нову пару до поточного речення
        currentSentence[n.leaf[i].type] ←
              n.leaf[i].dependentWord;
        // рекурсивний виклик до синівської верши
        newPairs ←recursiveExtract(c.dependentNode)
        // додаємо знайдені елементи поточного
    речення в синівських вершинах
        додати newPairs до currentSentence
    end
    return currentSentence;
end
```

З цієї більш загальної структури даних, нам необхідні лише 6 типів зв'язку. А також необхідно обійти всі прості речення складного речення. Результуючу множину векторів отримаємо наступною операцією:

```
function parseText(String text): Set<String[6]>
begin
    extractedElements ← ∅
    sentences ← split(text)
    for i ← 1 until sizeof(sentences)
    begin
        simple_sentences ← split(sentences[i])
        for j ← 1 until sizeof(simple_sentences)
        begin
              t ← recursiveExtract
                  (simple_sentences[j].rootNode());
```

```
                v ← {t[root], t[nsubj], t[dobj],
                      t[iobj], t[prep], t[xcomp]}
                додати v до extractedElements
        end
    end
end
```

Розбиття тексту на речення, а також розбиття речення на прості речення виконується засобами Stanford Parser.

2.4.3 Збереження даних

Після опрацювання кожної статті Вікіпедії необхідно об'єднати новий результат з вже існуючим, що представляє деяку технічну проблему. Вона полягає в тому, що в пам'яті ці данні зберігати не можна, оскільки кількість елементів може потенційно досягати 2^{64}. Проте виявляється, що тензор буде дуже сильно розрідженим. Оцінимо скільки пам'яті це буде займати. Не всі елементи шестірок будуть заповнені словами. Наприклад речення "Я пишу статтю" має лише підмет, присудок, а також прямий об'єкт, тому фактично є 3 слова в векторі, і 3 порожніх вироджених слова, які також займають місце. За рахунок порожніх місць, представлення тензору як множини входжень кожного елементу буде займати більше місця ніж вихідний корпус.

Скориставшись властивістю природномовних текстів, яка полягає в обмеженості кількості унікальних слів та сталих словосполучень природньої мови, приходимо до першої оптимізації розміру даних. Визначимо словник елементів тензору, в якому зберігаються пари "унікальне слово" та "порядковий номер". Для частини корпусу Вікіпедії Simple English кількість унікальних слів та словосполучень дорівнює 658058, при середній довжині 12 символів. Це означає, що замінивши слово його числовим представленням розміром 4 байти (для кількості слів достатньо 3 байтів, проте більшість систем мають 32-розрядну чи 64-

розрядну архітектуру, а тому більш швидко оперують з 4-х байтними числами), розмір тензору можна зменшити втричі.

З іншого боку, так як метод прихованого семантичного аналізу полягає в пошуці сталих сполучень слів, тому кожний елемент тензору повинен входити в нього декілька разів без змін. А значить об'єм тензору можна зменшити ввівши додаткове значення - кількість входжень елементу.

Описана структура даних найкраще лягає під концепцію баз даних. На Рис. 2.14 зображено ER-модель цієї бази даних.

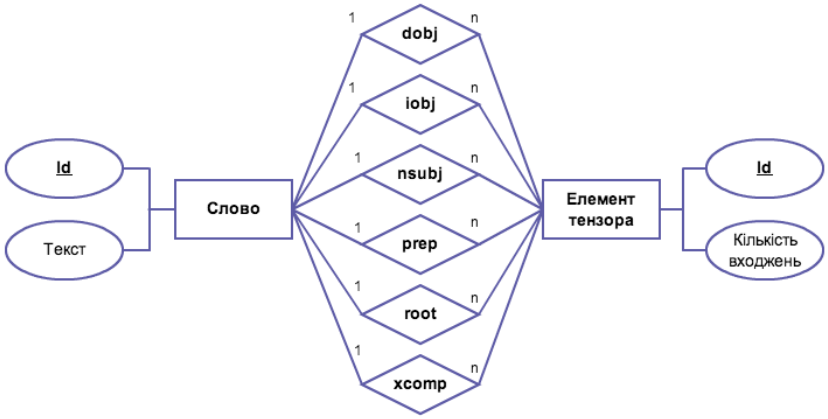


Рис. 2.14. ER-модель тензора в нотації Чена

2.4.4 Невід'ємна факторизація тензорів

На попередніх етапах було отримано тензор, яким вже можна користуватися для аналізу текстів. Проте він має 2 недоліки. Перший з цих недоліків - це його великий розмір. Другий недолік полягає в тому, що фраза, що зустрічається в аналізованому тексті, повинна була знаходитися в корпусі майже без змін, а тому в корпусі повинні міститися всі можливі комбінації слів в реченні.

Обидва недоліки можна в деякій мірі вирішити за допомогою невід'ємної тензорної факторизації. Невід'ємна факторизація тензорів полягає в розкладанні тензора на добуток тензорів меншого порядку. Один

з алгоритмів факторизації розглянутий в [33]. В рамках даної статі буде розглянуто обґрунтування застосування цього алгоритму для виділення сталих семантичних зв'язків.

Розглянемо цей підхід на короткому прикладі з 3-вимірним тензором. Нехай в нього входять наступні елементи:

(дівчинка, збирати, гриби) -> 1

(хлопчик, збирати, гриби) -> 1

(хлопчик, збирати, ягоди) -> 1

(біолог, досліджувати, гриби) -> 1

В такому представленні прихованих семантичних зв'язків, запит до системи "Чи може дівчинка збирати ягоди?" дасть негативну відповідь. Проте це не відповідає дійсності, бо з контексту видно, що і дівчинка і хлопчик з словом збирати входять в схожі речення.

Розкладемо цей тензор на 3 матриці - *A*, *B* і *C*.

Добуток матриць визначимо наступним чином:

$$T_f(x,y,z) = \sum_{i=1}^{n} A_{x,i} * B_{y,i} * C_{z,i}$$

Цільова задача факторизації:

$$T_f = \underset{A,B,C}{\operatorname{argmin}} \left\| T_f - T \right\|$$

де *n* - кількість фактор вимірів, тобто кількість груп слів що вживаються в різних значеннях. Чим більше кількість фактор вимірів, тим меншою буде різниця $\left\| T_f - T \right\|$, а тому факторизований тензор буде більш схожий на вихідний. Чим менше кількість фактор вимірів, тим більшим буде "розмиття" даних, а тому система зможе передбачати наявність більшої кількості зв'язків, яких не було в вихідному тексті.

A =

Слово \ Фактор	1	2
хлопчик	0.75	0
дівчинка	0.75	0
біолог	0	1

B =

Слово \ Фактор	1	2
досліджувати	0	1
збирати	1	0

C =

Слово \ Фактор	1	2
гриби	1	0
ягоди	1	0

Таким чином, оцінка того, що (дівчинка, збирає, ягоди) є коректною фразою дорівнює 0.75. Проте оцінки трьох інших фраз також впали до 0.75, але залишилися додатними і досить високими.

Тензор розмірності m буде представлений в добуток m матриць з n категорій наступним чином:

$$T(v) = \sum_{i=1}^{n} \prod_{j=1}^{m} M_{v_i,i}^{j}$$

Цільова функція визначається аналогічно до тривимірного варіанту:

$$T_f = \underset{M^i_{j,k}, i=\overline{1,m}, j-\text{слово}, k=\overline{1,6}}{\operatorname{argmin}} \left\| T_f - T \right\|$$

2.5 Алгоритм побудови керуючого простору природномовних речень

Для побудови керуючого простору природномовних речень скористуємося інформацією представленою деревом виводу та деревом залежностей. Дерево залежностей дає інформацію про тип зв'язків між словами, який ми зведемо до представлення альфа-бета-зв'язків керуючого простору. Дерево виводу дає інформацію про ієрархічність побудови більш складних груп слів починаю з безпосередньо слів.

2.5.1 Конвертація типу зв'язку дерева залежностей в тип зв'язку керуючого простору

Почнемо з простого випадку. Нехай в нас є фрагмент речення з двох слів, між якими є зв'язок по дереву залежностей. Припустимо, що кожному з типу зв'язків дерева залежностей відповідає один з типів зв'язку керуючого простору - альфа-, бета-, та кільцевий альфа-бета-зв'язок. Крім типу зв'язку також має значення напрямок зв'язку. В деяких випадках напрямок від головного до залежного слова в ДЗ збігається з напрямком в КП, в деяких - ні. Ми можемо скласти словник перетворень типів.

Для прикладу проаналізуємо як співвідносяться тип зв'язку nsubj ДЗ до типу зв'язку КП. nsubj вказує на іменниковий суб'єкт. Тобто іменниковою фраза, що є синтаксичним суб'єктом граматичної основи. Такий тип зв'язку є класичним зв'язком генерації. В реченні "Clinton defeated Dole", дерево залежностей містить зв'язок типу nsubj, що направлене від defeated до Clinton (позначається nsubj(defeated, Clinton). Це відповідає альфа-зв'язку що направлений від Clinton до defeated (позначимо alpha(Clinton, defeated).

Іншим типовим зв’язком є “Прямий об’єкт”, що позначається як dobj. Прямий об’єкт є залежною від дієслівної групи іменниковою групою. Такий тип зв’язку називається зв’язком поширення КП. При цьому напрямок збігається. В реченні “She gave me a raise” в ДЗ є зв’язок dobj(gave, raise), що конвертується в зв’язок beta(gave, raise).

Таблиця 2.2. Фрагмент таблиці співвідношення типів граматичного зв’язку Stanford Parser до зв’язків КП

Мітка Stanford Parser	**Тип зв’язку КП**	**Напрямок**
attr	альфа	зворотній
csubj	альфа	прямий
acomp	бета	прямий
agent	бета	прямий
amod	кільцевий альфа-бета	зворотній
poss	кільцевий альфа-бета	прямий

Повний варіант словника перетворень наведений в Додатку Б.

В коді проекту дана функціональність представлена класом DepencyToSpaceType. Клас реалізує патерн проектування Одинак (Singleton). При першому запиті сутності даного класу статичною функцією DepencyToSpaceType getInstance(), об’єкт створюється зчитавши з файлу "data/spaceByDeps.map" словник. При всіх наступних запитах функція повертає створений при першому запиті та збережений в статичну змінну об’єкт. Розбір файлу словника відбувається за допомогою використання регулярних виразів. Кожний рядок файлу має наступний формат :

<Тег типу зв’язку ДЗ>\s<Тег типу зв’язку КП>[*]

де <Тег типу зв’язку ДЗ> - строкового типу,

\s - довільний “порожній символ”, такий як прогалик, символ табуляції тощо,

<Тег типу зв’язку КП> - один з “a”, “b”, “ab”. Альфа-, бета-, та кільцевий альфа-бета-зв’язок відповідно,

[*] - за наявності вказує на зміну напрямку зв’язку в КП на протилежний порівняно з ДЗ.

Основна функціональність класу представлена функцією public Type getTypeFor(String stanfordType), що повертає розширений опис типу зв’язку КП для заданого строчкою типу ДЗ Стенфордського парсера.

Лема 1. Алгоритм конвертації працює за час *O(1)* та потребує *O(N)* додаткової пам’яті, де *N* - кількість типів зв’язків дерева залежностей.

Доведення.

Нехай є *N* різних типів зв’язку Стенфордського парсера. Кожному з них відповідає унікальний тег, що має розмір до 10 символів. Функція на вхід приймає значення цієї строки, та на виході повертає один з *N* результатів, що відповідає саме цьому типу зв’язка дерева залежностей.

Для реалізації такої функціональності скористаємося хеш-таблицею, що за хешем строкового літералу повертає розширений опис типу зв’язку КП. Обчислення хешу строки фіксовано розміру виконується за *O(1)*. Запит до хеш таблиці за обчисленим хешем також в середньому відбувається з *O(1)*. Ці дії відбуваються послідовно, тому оцінки складності додаються:

$$O(1) + O(1) = O(1)$$

Отже асимптотична складність роботи алгоритму конвертації буде *O(1)*.

Кожен запис в хеш-таблиці містить деякий обсяг *O(1)* пам'яті для хеш-значення фіксованого розміру, *O(1)* для збереження до 10 символів тегу, *O(1)* для опису типу зв'язку КП та *O(1)* службової інформації мови програмування. Всього таких записів в хеш таблиці є *N*. Крім того для підтримки структури даних хеш-хеш таблиці також необхідно *O(1)* пам'яті. Тому маємо:

$$O(1) + \big(O(1) + O(1) + O(1) + O(1)\big) * N = O(N)$$

Тобто оцінка пам'яті роботи алгоритму є *O(N)*.

2.5.2 Створення елементу керуючого простору для двох слів

Нехай дано 2 слова, їх POSTag'и, тип зв'язку ДЗ між ними, та тег найменшого піддерева дерева виводу, що їх містить. Алгоритм побудови КП для них буде тривіальним.

Спочатку знайдемо тип зв'язку КП між ними скориставшись екземпляром класу DepencyToSpaceType з попереднього розділу. Головним словом даного КП вважаємо слово, від якого напрямлений зв'язок КП. Тегом даного КП вважаємо тег піддерева дерева виводу що його містить.

Розглянемо частковий випадок алгоритму вибору головного слова для КП. Якщо одне з слів є прийменником, то головне слово має включати прийменник з цим словом через підкреслення. Це необхідно, оскільки ця службова частина мови в англійській мові входить в фразові дієслова та виступає деяким аналогом відмінку для іменника. З вищим пріоритетом будемо намагатися об'єднати слова, що пов'язані типом prt дерева залежностей. Згідно документації цей тег відповідає зв'язку, що зв'язує дієслово з службовою частиною в фразових дієсловах. Інший метод об'єднання полягає в аналізі POSTag'ів. Нас цікавить один з "IN", "ON", "OF", "AFTER", "AT".

2.5.3 Створення складного елементу керуючого простору для піддерева дерева виведення

Нехай для деякого вузла дерева виведення дано його прямі нащадки - піддерева з відповідними вже побудованими керуючими просторами. Будемо вважати що всі ці КП знаходяться на одному рівні ієрархії виходячи з дерева виводу. Рівні ієрархії для них будемо визначати пріоритетом тегів дерева залежностей. Найвищий пріоритет мають конкретні та специфічні типи зв'язку, що впливають лише на маленькі групи слів (наприклад зв'язок типу прикметник іменник). Найменший пріоритет мають абстрактні зв'язки, що зв'язують великі групи слів (Наприклад сполучні слова між простими реченнями). Якщо на одному рівні ієрархії є два однакових зв'язки, то розглядаємо зв'язки в порядку справа наліво, так як людина читає текст зліва на право, і по мірі закінчення речення йде уточнення його суті. Таким чином ми визначили порядок розгляду зв'язків, а значить порядок групування слів та існуючих КП в більші КП. В [86] наведено приклади неоднозначного такого трактування речень. Описаний порядок розгляду зв'язків був вибраний з метою обирати найбільш частовживаний варіант трактування речення.

Загальний алгоритм об'єднання одного рівня ієрархії КП виглядає наступним чином : поки не об'єднали все в один елемент КП, беремо зв'язок за найбільшим пріоритетом між різними не об'єднаними КП даного рівня, та заміняємо цих два входження КП на даному рівні на один об'єднаний. Об'єднання двох КП в один відбувається аналогічно об'єднанню двох слів в один КП. Різниця полягає в тому, що головним словом нового КП вважається головне слово КП, з якого направлений зв'язок КП. Тегом новоутвореного КП вважаємо тег батьківського вузла до всіх елементів рівня що розглядається.

Спрощений алгоритм виглядає наступним чином:

```
function  об'єднатиДекількаКПВОдин
```

```
        (Базовий_елемент кп[]) : Базовий_елемент
begin
    while sizeof(кп)>1
    begin
        for пріоритет ← максПріоритет
                step -1 until 1
        begin
            n ← sizeof(кп)
            for i ← n step -1 until 2
            begin
                if існує зв'язок в граматичних
                відносинах з пріоритетом = пріоритет
                між словами кп[i] і кп[i-1]
                then замінити кп[i] та кп[i-1]
                простором об'єднати2КП(кп[i], кп[i-
                1], тип граматичного зв'язку)
            end
        end
    end
    return кп[0]
end
```

Спрощена версія алгоритму працює не оптимально. В циклі "для пріоритет від максПріоритет до 1" по мірі об'єднання зв'язків з більш високим пріоритетом, алгоритм не буде виконувати ніякої корисної роботи, проте буде намагатися знайти зв'язки з більш високим пріоритетом. Розглянемо оптимізований алгоритм, що вирішує дану проблему:

```
function об'єднатиДекількаКПВОдин
    (Базовий_елемент кп[]) : Базовий_елемент
```

```
begin
    while sizeof(кп) > 1
    begin
        найбільшийПріоритет ← 0
        параЗНайвищимПріорітетом ← -1
        n ← sizeof(кп)
        for i ← n step -1 until 2
        begin
            if існує зв'язок зв в граматичних
            відносинах між словами кп[i] і кп[i-1]
            and зв.пріоритет > найбільшийПріоритет
            then
            begin
                найбільшийПріоритет ← зв.пріоритет
                параЗНайвищимПріорітетом ← i
            end
        end
        if параЗНайвищимПріорітетом != -1 then
        begin
            i ← параЗНайвищимПріорітетом
            замінити кп[i] та кп[i-1] простором
            об'єднати2КП(кп[i], кп[i-1], зв.тип
            граматичного зв'язку)
        end
        else
            замінити кп[n] та кп[n-1] простором
    об'єднати2КП(кп[i], кп[i-1], зв.тип граматичного
    зв'язку)
```

```
    end
    return кп[0]
end
```

Лема 2. Покращений варіант алгоритму об'єднання працює за час $O(N^2)$, та потребує $O(M)$ додаткової пам'яті, де N-кількість КП, що необхідно об'єднати, M - кількість типів зв'язків КП.

Доведення.

Розглянемо зовнішній цикл спрощеного алгоритму "доки кількість елементів в кп>1". До початку виконання цього циклу кількість елементів КП дорівнює N. На кожній ітерації алгоритму кількість КП зменшиться в точності на 1. Тому тіло зовнішнього циклу буде виконуватися N раз.

Розглянемо внутрішній цикл спрощеного алгоритму "для і від n до 2". Його тіло виконається в точності $N-1$ раз. Тіло цього циклу містить оператори лише порівняння та присвоювання, а тому виконується за час O(1).

Операція "замінити кп[і] та кп[і-1]" виконується за час $O(N)$, оскільки вимагає покрокового зсуву вліво даних.

Функція об'єднати2КП виконується за час $O(1)$ згідно до Леми 1.

В цілому маємо:

$$N * \big((N-1) * O(1) + O(N) * O(1)\big) = N * O(N) = O(N^2)$$

Тобто оцінка складності алгоритму об'єднання дорівнює $O(N^2)$.

В алгоритмі використані лише примітивні типи даних – лічильник та змінні для збереження поточного стану найбільшого пріоритету. Крім того було використано функцію об'єднати2КП, що згідно до Леми 1 потребує $O(M)$ додаткової пам'яті. Тому оцінка необхідної пам'яті для роботи алгоритму:

$$O(1) + O(M) = O(M)$$

2.5.4 Рекурсивний обхід дерева виводу для побудови керуючого простору

В попередніх розділах були наведені всі необхідні функції для побудови основного рекурсивного обходу дерева виводу для побудови КП.

Листки дерева виводу містять окремі слова. Це прості елементи КП, що не містять зв'язків. Головним словом такого елемента вважаємо саме слово. Тегом елемента вважаємо POSTag морфологічного аналізатора, визначений автоматично для даного слова.

В іншому випадку, візьмемо повний список синів цього вузла та видалимо з нього розділові знаки. В нашій реалізації КП розділові знаки не будуть входити в результат окремим елементом. Нагадаємо що в оригінальній статті [86], розділові знаки входили окремим елементом та були поєднані кільцевим альфа-бета-зв'язком з реченнями. Розділові знаки не були враховані в даній розробці для зменшення кількості зв'язків та спрощення самого КП.

Якщо після відсіювання розділових знаків залишився лише один син, то результат даного запуску рекурсивної функції буде збігатися з результатом запуску цієї ж функції для єдиного синівського вузла без змін.

Якщо ж таких вузлів декілька, то спочатку визначимо КП для кожного з піддерев цією ж рекурсивною функцією. Потім об'єднаємо отримані КП одного рівня ієрархії дерева виводу за допомогою функції описаної в попередньому розділі.

Нижче запропонований спрощений варіант алгоритму:

```
function побудуватиКП(Дерево p) : Базовий_елемент
begin
    if p – листок
```

```
then return Базовий_елемент(синтаксичний тег
отриманим з морфологічної розмітки, слово-
термінал для цього листку)
else
begin
    сини ← список синів вершини p
    for син ← сини
        кп ← union(кп,  {побудуватиКП(син)})
    return
        об'єднатиДекількаКПВОдин(керуючіПростори)
end
end
```

Теорема 3 (про обчислювальну складність). Алгоритм побудови КП працює за час $O(N * K)$ та потребує $O(M+K*log_K(N))$ пам'яті, при умові, що кожен вузол має K синів, N - сумарна кількість терміналів в дереві, M - кількість типів дерева залежностей. $K > 1$

Доведення.

Для обчислення складності роботи рекурсивних алгоритмів часто використовується Майстер-теорема.

Основна теорема про рекурентні співвідношення або Майстер-теорема дозволяє обчислити асимптотичну складність для рекурентних співвідношень наступного виду:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

Де n — розмір задачі,

a — кількість підзадач на кожному поступі рекурсії,

n/b — розмір кожної з підпроблем. (Тут мається на увазі, що всі підзадачі однакового розміру.)

$f(n)$ — обсяг роботи поза рекурсивними викликами, який включає обсяг задачі розділення й обсяг злиття розв'язків підпроблем.

Для доведення складності рекурсивного алгоритму спочатку обчислимо рекурентні співвідношення оцінок асимптотичної складності для етапів роботи алгоритм.

Виклик функції побудуватиКП для листка відбувається за час *O(1)*, тому:

$$T(1,K) = O(1)$$

Для не листка буде викликано рекурсивно функцію побудуватиКП K разів для в K разів меншого об'єму даних. Крім цього, на кожному етапі буде виконано виклик функції об'єднатиДекількаКПВОдин, що згідно до Леми 2 виконуються за час $O(K^2)$. Повна рекурентна формула буде мати наступний вигляд:

$$T(N,K) = K * T\left(\frac{N}{K}\right) + O(K^2)$$

Проте обсяг роботи поза рекурсивними викликами в випадку розглянутого алгоритму не є формулою від *N*, а тому Майстер-теорему застосувати в цьому випадку не можна.

Якщо розглядати складність кожного виклику функції без рекурсивних викликів, то час проведений в кожному вузлі дерева викликів буде залежати лише від *K*, і не буде залежати від *N*:

$$O(K + K^2)$$

Тоді як кількість вузлів дерева викликів буде залежати від *K* та *N*. На останньому кроці буде N вузлів що будуть обчислені за $O(1)$.

$$N * O(1) = O(N)$$

На передостанньому етапі буде в *K* разів менше вузлів, тобто $\frac{N}{K}$, що будуть обчислені за час $O(K + K^2)$. Кроком раніше - ще в *K* разів менше вузлів, що будуть обчислені за час час $O(K + K^2)$. Всього таких кроків буде $log_K(\mathrm{N})$ - глибина дерева рекурсивних викликів.

$$\underbrace{\frac{N}{K} + \frac{N}{K^2} + \frac{N}{K^3} + \cdots + K + 1}_{log_K(\mathrm{N})\ \text{доданків}} = N * \left(\frac{1}{K} + \left(\frac{1}{K}\right)^2 + \cdots + \left(\frac{1}{K}\right)^{log_K(\mathrm{N})}\right) =$$

$$= \Big|\text{обчислюємо суму } log_K(\mathrm{N}) \text{ перших членів геометричної прогресії з знамен}$$

$$= \frac{1}{K}, \text{з першим членом } b_1 = \frac{1}{K}\Big| =$$

$$= N * \left(b_1 * \frac{1 - q^n}{1 - q}\right) = N * \left(\frac{1}{K} * \frac{1 - \left(\frac{1}{K}\right)^{log_K(\mathrm{N})}}{1 - \frac{1}{K}}\right) =$$

$$= N * \left(\frac{1}{K} * \frac{1 - \frac{1}{N}}{\frac{K-1}{K}}\right) = N * \left(\frac{K}{K} * \frac{\left(\frac{N-1}{N}\right)}{K - 1}\right) = \frac{N-1}{K-1}$$

Разом

$$O\left(\frac{N-1}{K-1}\right) * O(K + K^2) + O(N) = O\left(\frac{N * K^2}{K-1}\right) =$$

$$= O(N * K), K > 1$$

Так як даний алгоритм викликає функцію об’єднатиДекількаКПВОдин, то згідно до Леми 2, він буде додатково виділяти *O(M)* пам’яті. Крім того на кожному кроці буде виділятися *K*

блоки для тимчасових змінних, що будуть виділені на стеці. Довжина стеку буде $log_K(\mathrm{N})$. Тому загальна оцінка пам'яті дорівнює

$$O(M + K * log_K N)$$

Теорема 4 (повна коректність). Алгоритм будує керуючий простір за скінченну кількість кроків, що відповідає вхідним дереву виводу та дереву залежностей.

Доведення.

Для доведення коректності рекурсивного алгоритму необхідно довести наступні твердження:

1. *Коректність використання рекурсивного звернення*. Доведемо, що результат, який вираховується в будь-якій рекурсивній гілці функції, буде вірним за умови, що відповідні рекурсивні виклики, у свою чергу, повернуть вірний результат.

На даному рівні будуть побудовані всі КП для синівський вершин ДВ. За умовою рекурсивного припущення, для них були побудовані коректні КП. Треба довести, що ці КП будуть коректно об'єднані за допомогою виклику функції об'єднатиДекількаКПВОдин. Припустимо, що це не так. Це значить, що один з синівських КП був відсутній або поєднаний некоректним зв'язком.

А) Нехай існує КП *A'*, що був побудований для синівської вершини, та відсутній в побудованому КП *A* для батьківської вершини. Значить існує крок циклу while №i, такий що масив кп функції об'єднатиДекількаКПВОдин його не містить. Таке неможливо на кроці 0, оскільки це суперечить умові, що він був побудований для синівської вершини. Тому для ітерації №i, існує ітерація №i-1, така що цей КП входить в змінну кп. На цій ітерації є 2 кроки, що змінюють змінну кп. Обидві заміняють 2 КП їх об'єднанням. А значить на етапі об'єднання було втрачено один з КП. Але це неможливо, оскільки алгоритм

об’єднання створює новий складений елемент КП, з існуючими КП в якості піддерев. Ми прийшли до протиріччя, з припущення що існує КП для синівської вершини, що не входить до батьківської.

Б) Нехай існує пара КП *A’* та *A’’*, такі що поєднані некоректним типом зв’язку *b*. Це означає що існує ітерація №і, на які *A’* та *A’’* були об’єднані, такі що *b* – не відповідає типу зв’язку, що має найвищий пріоритет. Це в свою чергу означає, цикл for працює не коректно, і не обирає пару слів з найвищим пріоритетом. З цього слідує, що існує пара (кп[і], кп[і-1]), такі що їх пріоритет більший за пріоритет *A’* та *A’’*. А це неможливо, оскільки для них би виконувалася умова “зв.пріоритет > найбільшийПріоритет”, і вони були б обрані як *A’* та *A’’*. Тому припущення хибне.

Так як обидва варіанти, що слідують з припущення, що КП будуть некоректно об’єднані за допомогою виклику функції об’єднатиДекількаКПВОдин є хибними, то і саме припущення є хибним. А значить рекурсивні значення використовуються коректно.

2. *Коректність всіх термінальних гілок*. Доведемо, що всі термінальні гілки повертають вірні значення. Доведення елементарне, оскільки в випадку термінальної гілки обчислень проведено не буде. Буде повернуто базовий елемент КП, що містить синтаксичний тег отриманим з морфологічної розмітки та слово-термінал для цього листку. Такий КП буде коректним та відповідати терміналу ДВ.

3. *Досяжність термінальній гілки для будь-якого коректного набору параметрів після скінченого числа рекурсивних викликів*. Нехай існує термінальна гілка *А*, що недосяжна за скінчену кількість кроків рекурсивних викликів. Значить існує нетермінал *Б*, з якої вона виводиться, для якого не було викликано рекурсивно функцію з деякої вершини *В*. Але це не можливо, оскільки для вершини *В* робляться виклики для всіх

синівських вершин, включаючи *Б.* Тому припущення хибне, і будь-якого термінальна гілка досяжна за скінчене число рекурсивних викликів

2.5.5 Підвищення точності алгоритму побудови керуючого простору за допомогою використання інформації про виділені іменовані сутності

Ідея покращення полягає в використанні іменованих сутностей в якості головних слів, замість одиноких слів що входять в них.

Нехай в нас є КП деякого речення та виділені іменовані сутності в представлені інтервалу номерів слів речення які об'єднують їх в одну сутність. Підхід що був випробуваний, полягає в знаходженні для кожної іменованої сутності цього речення мінімального КП що містить всі слова цієї іменованої сутності, та визначення головним словом цього КП цілою фразою. Проте виявилося, що для іменованих сутностей дерево виводу будується не коректно, і внаслідок такі піддерева містили інші слова, що не входять в цю іменовану сутність. Найбільш правильним рішенням було б модифікувати алгоритм побудови дерева виводу, щоб цей алгоритм вмів використовувати інформацію про іменовані сутності в якості додаткових тегів морфології , але це досить не тривіальна задача. Тому в роботі було модифіковано результат роботи дерева виведення наступним чином:

1. Всі слова іменованої сутності були видалені з існуючого дерева виведення разом з тегами
2. Створено нове піддерева дерева виводу, що містить одну батьківську вершину з тегом NP - іменникова група, та всі слова іменованої сутності в якості синів-листків цієї вершини.
3. Для сформованого піддерева було створено КП.
4. Головне слово КП - місце вставки нового піддерева в існуюче дерево виводу.

Модифіковане дерево виводу не є класичним деревом виводу, так як не має багатьох його властивостей. Для потреб побудови керуючого простору цим можна знехтувати. Модифіковане дерево виводу з точки

зору побудови КП запропонованим вище алгоритмом отримує важливу властивість - всі іменовані сутності будуть розташовані на одному рівні дерева виведення, а тому в згенерованому КП мінімальний підпростір що містить слова іменованої сутності не буде містити інших слів.

Результат роботи алгоритму представлений на Рис. 2.15.

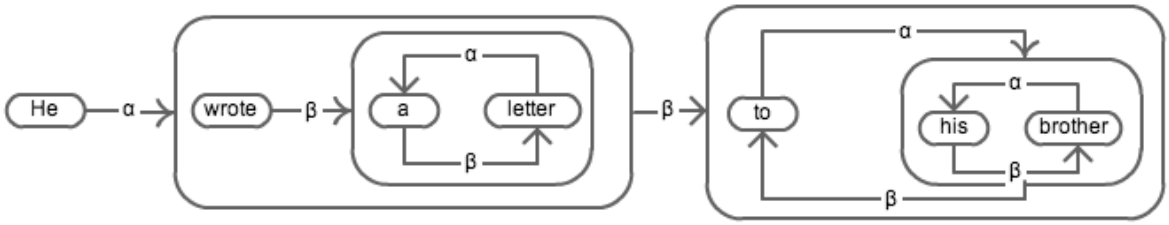


Рис. 2.15. Діаграма автоматично згенерованого КП

2.6 Виділення типових елементів природномовних речень

В попередніх розділах було описано процес побудови керуючого простору з довільного нерозміченого тексту англійською мовою. Наступна задача - проаналізувати великий текстовий корпус, з метою виділення типових елементів природномовних речень. Цей процес базується на ідеях латентного семантичного аналізу (ЛСА) та невід'ємної тензорної факторизації (НТФ). Ми будемо будувати керуючі простори для великого текстового корпусу, виділяти з них більш прості елементи, та використовувати НТФ для виділення закономірностей.

2.6.1 Збереження тензора в реляційній базі даних

Для збереження тензора необхідна ефективна структура даних, що підтримує швидку операцію додавання нового елемента для розрідженого тензора. Ми не можемо зберігати цю структуру даних в оперативній пам'яті, оскільки по-перше тензор в загальному випадку займе більше місця, ніж наявно оперативної пам'яті в сучасному ПК. По-друге в разі аварійного завершення програми всі дані будуть втрачені. Виходячи з цих обставин, було вирішено використовувати СКБД (Система керування базами даних).

На Рис. 2.16 зображено модель “сутність-зв’язок” (ER-модель), що представляє структури даних необхідні для збереження розрідженого тензора. Таблиця Word зберігає в собі посимвольне представлення всіх використаних слів з їх прив’язкою до цілочисельного ключа *id*. Таблиця Binary relation зберігає в собі двоелементні структури керуючого простору. Ключ цієї таблиці є складним, і складається з 3 елементів - ідентифікатор першого слова *word1*, ідентифікатор другого слова *word2* та тип зв’язку між ними *type*. В КП природномовних речень типами двоелементних структур може бути кільцевий альфа-бета-зв’язок, одинокі альфа- та бета-зв’язки. Поле Count зберігає інформацію про кількість входжень даного елемента. Це поле має бути додатнім, оскільки ми не зберігаємо не заповнені елементи тензора. Окремою таблицею “AB relation” представлені послідовні альфа-бета-зв’язки керуючого простору. Ключем виступають ідентифікатори трьох слів, які зв’язані даним типом зв’язку. Поле *Count* зберігає кількість входжень елемента КП аналогічно до таблиці “Binary relation”.

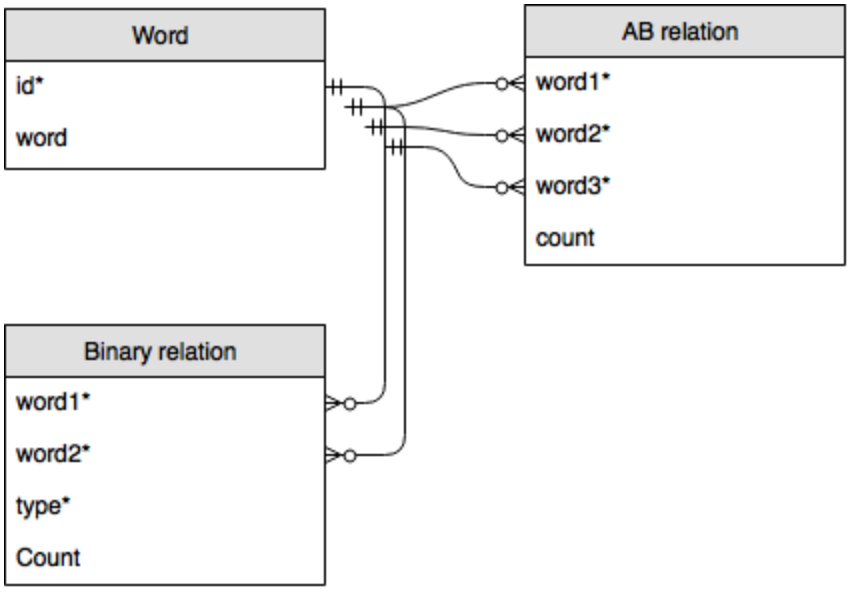


Рис. 2.16 ER-модель представлення тензора

Доведемо, що дана ER-модель знаходиться в нормальній форма Бойса—Кодда.

Перша нормальна форма (1НФ) утворює ґрунт для структурованої схеми бази даних:

- Кожна з трьох таблиць має основний ключ: мінімальний набір колонок, які ідентифікують запис.
- Відсутні повторення груп (категорії даних, що можуть зустрічатись різну кількість разів в різних записах). Це було отримано правильним визначенням неключових атрибутів.
- Атомарність: кожен атрибут має лише одне значення, а не множину значень.

Друга нормальна форма (2НФ) вимагає, аби дані, що зберігаються в таблицях із композитним ключем не залежали лише від частини ключа:

- Дані, що повторно з'являються в декількох рядках виносяться в окремі таблиці. В нашому випадку це слова. Вони були винесені в таблицю “Word”.

Третя нормальна форма (3НФ, 3NF) вимагає, аби дані в таблиці залежали винятково від основного ключа:

- Будь-яке поле, що залежить від основного ключа та від будь-якого іншого поля, має виноситись в окрему таблицю. Такими полями є Count та word. Вони в всіх трьох таблицях залежать лише від основного ключа.

3НФ не збігається з НФБК лише тоді, коли одночасно виконуються такі 3 умови:

1. Відношення має 2 або більше потенційних ключів.
2. Ці потенційні ключі складені (містять більш ніж один атрибут)
3. Ці потенційні ключі перекриваються, тобто мають щонайменше один спільний атрибут.

Так як принаймні перша властивість не виконується, то дана БД знаходиться в 3 НФБК.

2.6.2 Архітектура системи обробки великих текстових корпусів

Обробка корпусів буде виконуватися за допомогою послідовності фільтрів (трубопровід, пайплайн, pipeline). У розробці програмного забезпечення, пайплайн складається з ланцюжка з процесорних елементів (процесів, потоків, співпрограм, функцій і т.д.), розташованих так, що вихід кожного елемента надходить на вхід наступного. Зазвичай для балансування навантаження на елементи пайплайну, дані між ними буферизуються. Це підвищує швидкодію системи. Інформація, яка проходить в цьому пайплайні представляє собою потік записів, байтів або бітів. Елементи пайплайну часто називають фільтрами.

Вузько кажучи, пайплайн є лінійним і одностороннім, хоча іноді термін застосовується до більш загальних потоків. Наприклад, потоки по одно направленому дереву і іншим направлених топологіях, таких як ациклічний граф, поводять себе аналогічно до лінійних пайплайнів. Відсутність циклів робить такі структури обробки простими.

Першим фільтром в ланцюжку обробки в проекті буде FileInputStream. Цей фільтр ініціалізується ім’ям файлу. В процесі роботи він буде вичитувати дані з цього файлу за необхідності та передавати процесорам що йдуть після нього в пайплайні. Виходом фільтру є масив байт запрошеного розміру.

Останнім фільтром в ланцюжку буде фільтр абстрактного інтерфейсу PageHandler. Його реалізація має обробляти одну не велику незалежну частину корпусу - будь то сторінка Вікіпедії, чи стаття журналу.

```
public interface PageHandler {
    void handle(String page, String title);
}
```

Тут page - текст сторінки, що буде аналізований,

title - заголовок сторінки за наявності.

Реалізація алгоритму побудови керуючого простору буде викликатися в об’єкті класу, що реалізує даний інтерфейс. Там же буде знаходитися код, який буде зберігати керуючі простори в реляційну базу даних.

Також корисною буде можливість не обробляти вже оброблені сторінки при завершенні програми в середині обробки даних. Ця функціональність реалізована за допомогою декоратора OnlyUnprocessedPageHandler. Декоратор (Decorator, Wrapper) — структурний шаблон проектування, призначений для динамічного підключення додаткових можливостей до об'єкта. Шаблон Декоратор надає гнучку альтернативу методу визначення підкласів з метою розширення функціональності. Клас OnlyUnprocessedPageHandler є нащадком PageHandler. Конструктор OnlyUnprocessedPageHandler отримує та зберігає в об’єкті посилання на об’єкт класу PageHandler, функціонал якого буде декорований. Функція handle спочатку зробить виклик до однойменної функції handle збереженого внутрішнього об’єкту, а потім збереже в текстовий файл заголовок обробленої статі. При наступному запуску, цей файл буде існувати, а тому функція handle декорованого об’єкту не буде викликана до тих пір, поки не надійде остання успішно оброблена стаття.

Проміжні фільтри пайплайну будуть різними для різних джерел даних.

Вікіпедія є вільною енциклопедією, а тому весь її вміст можна завантажити одним архівом xml файлу з офіційного сайту. Вся обробка відбувається на основі послідовної обробки різними фільтрами. Одразу після фільтру вичитки файлу в пайплайні розташований фільтр розпаковки BZip2CompressorInputStream з вільного пакету org.apache.commons [8]. Він має спільного нащадка з FileInputStream - InputStream, а тому функція

читання має однакову сигнатуру - він також повертає масив байт вказаної довжини.

Для розбору XML існує 2 основних підходи - SAX (Simple API for XML) та DOM (Document Object Model). Більшість програмістів XML технологій вважають, що обробка XML документів відповідно парадигмі SAX, в цілому, швидша, аніж при використанні DOM. Це пояснюється тим, що потік SAX потребує набагато меншого обсягу пам'яті у порівнянні із побудовою повного дерева DOM.

SAX аналізатори реалізують з використанням підходу передачі повідомлень (event-driven), коли програмісту необхідно описати обробники подій, які викликаються аналізаторами під час обробки XML документа.

SAX було розроблено зусиллями спільноти списку розсилки xml-dev, без формальних комітетів, але він був швидко визнаний компаніями, які спеціалізуються на засобах обробки XML документів. Першим головним розробником та інженером супроводу був Давід Маггінсон [15].

На мові Java було використано реалізацію SAX класом javax.xml.parsers.SAXParser. Реалізація обробки подій реалізована простим скінченим автоматом з 3 станами - “заголовок”, “текст”, “‘інше”. При виході з стану “текст” накопичена стаття відсилається в PageHandler.

2.7 Профілювання та оцінка результатів

Опишемо процедуру тестування, за допомогою якої можна отримати якісні показники роботи алгоритму на великих корпусах текстів. Керуючий простір складається з елементів та зв’язків між ними, тому нехай елементарною структурою, правильність якої ми будемо визначати, буде тип зв’язку та 2 компоненти, які він зв’язує. Окремо будемо збирати статистику правильних (true positive, tp), неправильних (false positive, fp), а також не знайдених залежностей (false negative, fn) по кожному з типів

зв’язків керуючого простору. В окремий тип зв’язку будемо обчислювати статистику по кільцевим альфа-бета зв’язкам, оскільки в керуючому просторі вони мають додаткову семантику. Також отримаємо загальну статистику по всім типам зв’язків.

Знайшовши кількість цих зв’язків можна визначити міру точності (precision), повноту, та F1-міру за наступними формулами:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F = 2\frac{precision \cdot recall}{precision + recall}$$

Результати тестування на деякій підмножині документів енциклопедії Вікіпедія представлені в таблиці 2.3.

Таблиця 2.3. Результати тестування якості роботи алгоритму побудови керуючого простору

	Міра точності (Precision)	**Покриття (Recall)**	**F1-міра**
Кільцеві альфа-бета зв’язки	77.0	91.9	83.8
Альфа зв’язки	85.7	82.3	84.0
Бета зв’язки	96.3	71.2	81.9
Разом	78.8	89.3	83.2

Крім показників якості роботи алгоритму, також проаналізуємо показники швидкості. Для цього зручно скористатися інструментом профілювання середовища NetBeans. Даний інструмент автоматично додає

до кожної функції проекту код, що записує час проведений в цій функції. Таким чином можна знайти ділянки коду, що виконуються найбільше часу. З Рис. 2.17. видно, що в середньому алгоритм побудови керуючого простору займає більше ніж на порядок менше часу ніж попередній синтаксичний аналіз. Обробка одного рівня дерева виведення займає близько 400 мілісекунд.

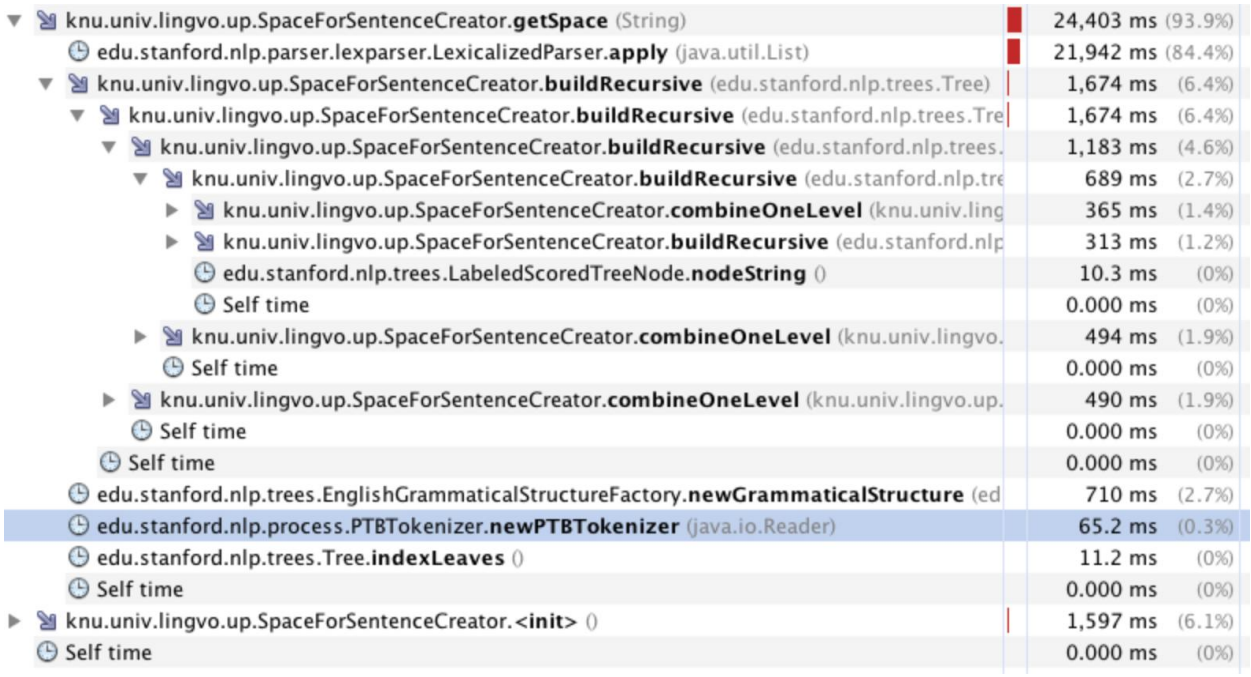


Рис. 2.17 Результати профілювання додатку для побудови керуючих просторів

2.8 Висновки

В розділі розвивається ідея невід'ємної факторизації тензорів. Пошуки кращої системи розпочалися з розширення вимірності тензору, що дозволило виділити більш складні структури природномовних речень. В оригінальній роботі, що описує тензорну модель було використано 3 виміри – підмет, присудок та додаток. В запропонованому в розділі шестивимірному тензорі також використовується прямий об'єкт, не прямий об'єкт та прийменник.

З збільшенням розмірності тензору експоненційно росте об'єм пам'яті необхідної для їх збереження. Тому було вирішення надалі модель

розвивати за допомогою більш компактного представлення інформації. Для цього було використано керуючі простори синтаксичних структур природномовних текстів. Для роботи з цим представленням текстів, було розроблено алгоритми їх побудови на основі дерев виводу та дерев залежностей. Для цих алгоритмів було визначено обчислювальну складність та коректність. Для представлення керуючих просторів було розроблено деревовидну рекурсивну структуру даних та доведено існування бієктивного відображення конвертації класичного представлення КП в деревовидне.

В заключення розділу було проведено тестування роботи системи в термінах точності та покриття, а також профілювання додатку засобами середовища NetBeans.

Таким чином було побудовано тензорну модель керуючих просторів, яка буде використана в наступному розділі для вирішення задачі знаходження кореферентних зв’язків.

Розділ 3. Алгоритми визначення кореферентних зв'язків за допомогою статистичної інформації типових структур керуючих просторів

Система знаходження кореферентних зв'язків складається з двох головних етапів: виявлення сутностей та виявлення залежностей між ними. На першому етапі знаходяться сутності разом с релевантною інформацією про них (наприклад рід та число). На другому етапі відбувається власне знаходження кореферентних зв'язків. Алгоритм фіксує певний порядок обходу пар сутностей та об'єднання їх в одну групу, чи навпаки внесення в список пар сутностей, що гарантовано не є кореферентними. Рішення щодо наявності кореферентного зв'язку для пари сутностей виноситься на основі роботи набору решіт (Рис 3.1), що відсортовані він найбільш точних, до менш точних. Таким чином більш точні методи мають більший пріоритет. Менш точні методи підвищують кількість знайдених залежностей, при цьому допускають помилки. Наприклад найперше найбільш точне решето вимагає посимвольного збігання двох сутностей. Важливим є факт, що знаходження сутностей розроблювалося з метою підвищення повноти (recall), тоді як для другого етапу знаходження кореферентних зв'язків точність (precision) є більш важливою. Відомо, що така архітектура системи знаходження кореферентних зв'язків показала кращі результати на Conll'2011. Тому саме вона була обрана для даної роботи.

3.1 Пошук сутностей

Задача пошуку сутностей полягає в знаходженні слів та словосполучень, між якими ми надалі будемо знаходити кореферентні

зв’язки. Так як головним результатом є кореферентність, то алгоритму пошуку сутностей намагається знаходити якнайбільше таких слів та словосполучень, знехтуючи можливими помилково знайденими сутностями. Чому ми так робимо? Припустимо ми не змогли знайти деяку сутність. В результаті ми не знайдемо кореферентний зв’язок в який вона входить. Це в свою чергу призведе до зниження якісних показників роботи алгоритму знаходження кореферентностей. Припустимо інший варіант, ми знайшли деяке словосполучення, яке не є сутністю. Більш за все таке словосполучення не будо пов’язано зв’язком кореферентностей ні з яким іншим словом. А тому це не вплине на результат знаходження кореферентних зв’язків.

Рис. 3.1 Архітектура знаходження кореферентностей заснована на ситах (фільтрах)

Для пошуку сутностей спочатку виконуються попередній синтаксичний аналіз описаний в 2 розділі. З дерева виводу обираються всі іменникові фрази (мають тег NP та NNP). Також до переліку знайдених сутностей додаються іменовані сутності (власні назви). Наступним кроком ми відсіюємо ті з них, для яких виконується хоча б одне з наступних правил:

1. Видаляємо сутність, якщо є більша сутність з тим самим головним словом. Наприклад, старенький дідусь буде видалено якщо є сутність старенький дідусь в синьому пальто.

2. Видаляємо такі сутності як проценти, гроші, порядкові номера, кількості, та деякі інші сутності що містять числівники.

3. Видаляємо сутність, що виражає частину цілого або комбінацію. Наприклад “сума двох чисел”.

4. Видаляємо займенник it, що слугує як допоміжна стала частина таких фраз як “It is possible”. Це відбувається за словником всіх можливих фраз.

5. Видаляємо етнохрононіми - назви мешканців певної місцевості, співвіднесена з топонімом. Наприклад, Київ — киянин, Одеса — одесець (форма «одесит» – неправильна, скалькована з офіційної мови Російської імперії), Крим — кримчанин.

6. Видаляємо слова що належать до стоп-списку. Наприклад “there”, “ltd”.

3.2 Зведення задачі знаходження кореферентних зв’язків до тернарного класифікатора пари сутностей

Після того як ми виділили сутності в тексті, ми їх спочатку сортуємо за порядковим номером речення, в яке вони входять. Сутності, що входять в одне і те ж речення, сортуються за допомогою обходу дерева виводу алгоритмом пошуку в ширину. Таким чином надається перевага найбільш великим групам слів, що йдуть в тексті найпершими.

В кожному новоутвореному кластері сутностей ми будемо порівнювати тільки перші з них. Для рішення було декілька причин:

1. Перші входження сутностей в більшості випадків краще визначені, що дозволяє більш точно визначити їх зміст та побудувати простір ознак для машинного навчання.

2. Менше порівнянь антецедентів означає меншу ймовірність зробити помилкове рішення про кореферентність.

Наприклад, нехай дано наступний список сутностей $\{m_{1,1}, m_{2,2}, m_{2,3}, m_{1,4}, m_{1,5}, m_{2,6}\}$, де перший індекс означає порядковий номер сутності після сортування, другий - номер кластера до якого його було раніше віднесено. В цьому списку ми будемо визначати кореферентність тільки однієї пари сутностей - $m_{1,1}$ та $m_{2,2}$.

На наступному кроці ми видалимо з порівняння перші сутності в кластері з невизначеними займенниками (такими як деякі, інші, тощо) чи невизначеними артиклями (a, an), якщо вони не мали жодного антецеденту для найбільш точних решіт посимвольного порівняння. Для кожного нової сутності m_i буде проведено порівняння з попередніми сутностями m_{i-1}, …, m_0.

Спочатку ми застосовуємо до всіх пар перше решето, потім друге і так далі. Кожного разу коли ми знаходимо пару кореферентних сутностей, ми починаємо процес спочатку, оскільки властивості сутностей в кластерах могли збагатитися новою інформацією. Всі сутності одного кластеру вказують на один об’єкт, а тому розподіляють між собою синтаксичні атрибути (число, рід, істота чи не істота, тощо). Таким чином ми можемо робити більш точний вибір. Наприклад, якщо є кластер що містить дві сутності : “група студентів” для якого число - однина, та “п’ять студентів” - число множина. Такий кластер буде містити інформацію про число - однина чи множина разом. Це дозволить об’єднувати даний кластер як з сутностями однини, так і з сутностями множини.

3.3 Порівняння піддерев

Розглянемо принципи синтаксичного та семантичного паралелізму.

Згідно синтаксичного паралелізму, кореферентні сутності відіграють схожу синтаксичну роль в реченнях та проявляють схожі синтаксичні властивості.

Наприклад:

Що шукає він у країні далекій?

Що кинув він в краю рідному?

(М. Лермонтов)

Згідно семантичного паралелізму, кореферентні сутності проявляють схожі семантичні властивості.

Негативний приклад: Якщо покласти лід в вогонь, то він розтане.

Сутність “вогонь” не може проявляти властивість “розтавати”, а тому “він” не кореферентне “вогонь”.

Позитивний приклад: Атож, тогочасною Англією правила королева Вікторія, і правила вона довше за будь-кого з британських королів - аж шістдесят чотири роки: від 1837-го по 1901-й, так що ціла та доба згодом дістала в історії назву "вікторіанської". Льюїс Керолл, “Аліса в країні чудес”.

Обидві сутності “королева Вікторія” та “вона” ініціюють відношення “правила”, що разом з іншими ознаками може свідчити про кореферентність.

На основі комбінації цих двох принципів було розроблено алгоритм перевірки ізоморфізму дерев, що враховує синтаксичну та семантичну складову.

```
function подібність(Базовий_елемент а,
Базовий_елемент б, int глибина) : double
begin
   if а – листок or б - листок
   then return 0
   else return оцінити(а, б) * вага[глибина]
        + подібність (а.ліве, б.ліве, глибина + 1)
        + подібність (а.праве, б.праве, глибина + 1)
end
```

Вважаємо, що листки або прості елементи КП не ізоморфні між собою. Ізоморфізм складених елементів КП визначається як оцінка ізоморфізму даного рівня дерев визначена функцією *оцінити* плюс сума ізоморфізмів відповідних піддерев керуючого простору. Так як зв'язки керуючого простору напрямлені, то достатньо порівняти лише ліве піддерево з лівим, та праве з правим. Визначимо функцію порівняння одно рівня піддерев:

```
function оцінити(Базовий_елемент а, Базовий_елемент
б): double
begin
    if а.тип != б.тип
    then return 0
    факторА = argmax(фактор) тензор(а.тип)(
а.ліве.головнеСлово, а.праве.головнеСлово, фактор)
    факторБ = argmax(фактор) тензор(а.тип)(
б.ліве.головнеСлово, б.праве.головнеСлово, фактор)
    return факторА == факторБ ? 1 : 0
end
```

Тобто, якщо тип зв'язку не збігається, то оцінка ізоморфізму дорівнює нулю. В іншому випадку визначаємо фактор множини до яких входять головні слова лівого та правого піддерева кожного з дерев. Якщо ці фактор множини збігаються, то оцінка ізоморфізму дорівнює 1, інакше - 0.

Синтаксична складова методу полягає в порівнянні структури КП. Тоді як входження слів до тієї ж самої фактор-множини тензору означає близьке за змістом вживання цих слів.

Теорема 5. Алгоритм оцінки семантико-синтаксичного паралелізму працює за час $O(\min(N_1, N_2) * K)$ та використовує $O(log_2 \min(N_1, N_2)$

пам’яті, де N_1 та N_2 - кількість елементів в КП, що аналізуються, K - кількість фактор-множин, обрана при факторизації тензора.

Доведення.

Рекурсивних алгоритм обходить в точності по одному разу ті елементи дерева, що зустрічаються в обох деревах. Таких елементів не більше $\min(N_1,N_2)$. Для кожної пари елементів дерев викликається функція оцінити. Функція оцінити викликає функцію argmax по всім існуючим фактор множинам, яких в нашому випадку K. Тому часова оцінка складності роботи алгоритму визначається як:

$$O\big(\min(\mathrm{N1},\mathrm{N2})\big) * O(K) = O(\min(\mathrm{N1},\mathrm{N2}) * \mathrm{K})$$

Пам’ять в даному рекурсивному алгоритмі виділяється лише для збереження стеку рекурсивних викликів та пропорційна глибині дерев керуючих просторів:

$$O(log_2 \min(N_1, N_2))$$

3.4 Оцінка наявності кореферентного зв’язку за допомогою виявлення семантичного паралелізму керуючих просторів

Розглянемо спочатку алгоритм оцінки семантичного паралелізму.

Вхід : 2 сутності та відповідні ним керуючі простори синтаксичних структур (Рис. 3.2, а, б)

Вихід : два числа - оцінки сумісності.

Підставимо першу сутність в другий керуючий простір замість першої сутності. Отримаємо елемент КП зображений на Рис. 3.2 б). Зробимо запит до тензору розгорнутих альфа-бета зв’язків для даного елементу. Результатом запиту буде перше число, що характеризує оцінку семантичної сумісності першої сутності в контексті другої сутності. Аналогічно отримаємо друге число підставивши другу сутність в КП першої сутності.

На Рис. 3.2 б) було розглянуто розгорнутий альфа-бета зв’язок (вогонь, розтанув, -), що є не сумісною трійкою, а тому запит до тензора поверне близьке до нуля значення. Порівнявши це значення з деяким порогом ми прийдемо до висновку, що дана пара сутностей є не кореферентними.

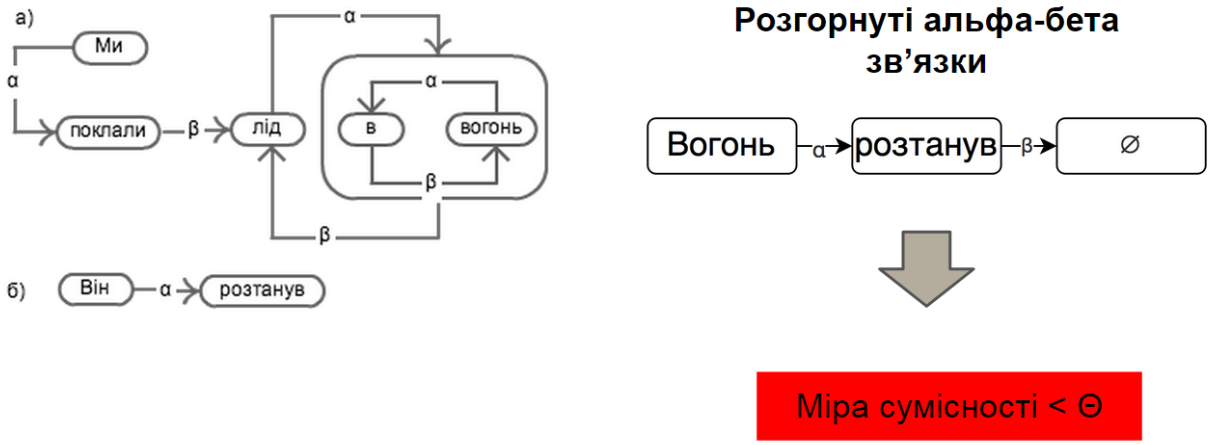


Рис. 3.2. Ліворуч а) керуючий простір речення “Ми поклали лід в вогонь”, б) КП речення “Він розтанув”. Праворуч запит до тензора розгорнутих альфа-бета зв’язків.

Теорема 6. Алгоритм оцінки семантичного паралелізму працює за час O(*K*), де *K* - кількість фактор-множин, обрана при факторизації тензора.

Доведення. Нагадаємо формулу розкладу тензора в поелементній формі:

$$y_{itq} = \sum_{j=1}^{J} a_{ij} b_{tj} c_{qj} + e_{itq}$$

Запит до тензора розкладених зв’язків обчислює значення оцінки семантичної сумісності за допомогою наступної формули за відомими

значеннями матриць *A*, *B* та *C*:

$$y'_{itq} = \sum_{j=1}^{K} a_{ij} b_{tj} c_{qj}$$

А тому часова асимптотична складність роботи алгоритму заснованого на такому запиті буде лінійною по *K* :

$$O(K)$$

3.5 Визначення слів індикаторів як фактор-множин розкладених тензорів

В системі описаній в [56] запропоновано використовувати деякий список слів, як таких що підвищують ймовірність, що сутність що йде в тексті поряд має референт. В цій роботі було запропоновано використовувати словник таких слів. Проте це не завжди зручно. По-перше, його треба зіставляти для кожної нової мови, що буде аналізуватися. По-друге, людина має бути спеціалістом з мовознавства, щоб найбільш точно підібрати такий список. Тому постає необхідність покращити цей метод, за допомогою автоматичної побудови словника слів індикаторів.

Для побудови словника скористуємось фактор-множинами на які буде розкладено тензор. Розглянемо два підпростори зображені на Рис. 3.3. Тут X - кандидат в антецеденти для якого будемо визначати вагу методом слів індикаторів; Ω - будь-яке слово тексту. Вага даним методом визначається як ступінь ізоморфізму даного підпростору до існуючого в тексті. Тобто визначається, як оцінка семантико-синтаксичної близькості слова discuss до конкретного слова аналізованого тексту в залежності від типу зв’язку, в який входить антецедент.

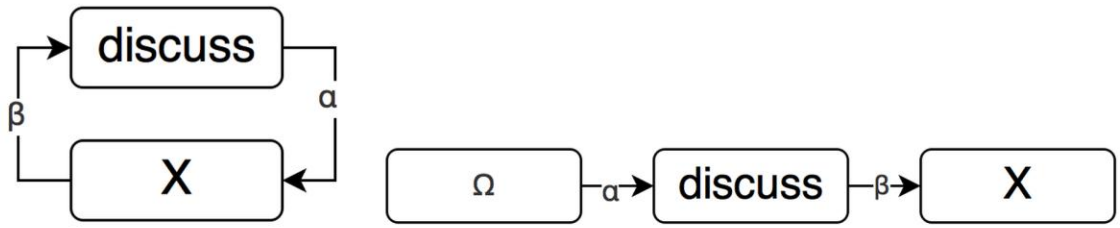


Рис. 3.3. Розгорнуті та кільцеві альфа-бета підпростори-індикатори

Теорема 7. Алгоритм оцінки наявності кореферентного зв'язку за допомогою фактор-множин працює за час *O(K)*, де *K* - кількість фактор-множин використаний в невід'ємній факторизації.

Доведення.

Складність формування запиту (знаходження двійки та трійки) відбувається за час *O(1)*. Доведення часової складності запиту до тензора аналогічно доведенню Теореми 6.

3.6 Особливості тензорної факторизації розгорнутих та кільцевих альфа-бета зв'язків

В результаті обробки великого текстового корпусу ми записали в реляційну базу даних 3 двовимірних тензори - альфа-, бета-, та кільцеві альфа-бета-зв'язки, та один тривимірний - лінійні альфа-бета-зв'язки. Наступний крок полягає в невід'ємній факторизації цих тензорів та виділення певних закономірностей. Саме цю задачу виконує не від'ємна факторизація тензорів.

В останніх розділах було описано, що розріджений тензор було записано як входження окремих його елементів в СУБД. Невід'ємна тензорна факторизація - це складний математичний алгоритм, який найбільш ефективно вирішувати в математичному пакеті. Вирішення цієї задачі вже було написано для середовища MatLab, що заснований на поблочному координатному спуску (Block Coordinate Update, BCU) [73]. Дана реалізація алгоритму коректно працює з великими тензорами. Для

тензорів розміром більшими за 4000 за одним з вимірів весь тензор не зберігається в пам’яті.

Спочатку треба експортувати дані з СУБД MySQL і імпортувати в середовище MatLab. Експорт з СУБД проведемо в файл CSV-формату (comma-separated values, розділені комою значення). Для цього в MySQL є команда експорту результату команди select в файл :

```
SELECT word1, word2, type, Count FROM “Binary
relation”
WHERE type=’2’
INTO OUTFILE 'cir_ab_relation.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Таким чином ми отримали текстовий файл що містить інформацію про кільцеві альфа-бета-зв’язки (тип дорівнює 2). Аналогічно можна отримати інформацію про одинокі альфа-зв’язки (тип дорівнює 0) та одинокі бета-зв’язки (тип дорівнює 1).

Для запиту інформації про розгорнуті альфа-бета-зв’язки скористаємося наступним запитом:

```
SELECT word1, word2, word3, Count FROM “AB relation”
INTO OUTFILE 'seq_ab_relation.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Наступний кроком буде імпорт даних в тензор в середовищі MatLab. Для імпорту цього формату існує команда:

```
M = csvread(filename)
```

На практиці її довелося реалізувати вручну для економії пам’яті, вичитуючи рядок за рядком файл та вставляючи значення в розріджений

тензор MatLab. Для збереження розрідженого тензору скористаємося MATLAB Tensor Toolbox Version 2.6 [50]. Необхідна нам структура даних цього пакету має назву sptensor. Неоптимізований варіант ініціалізації виглядає наступним чином:

```
MSparse = sptensor(max(0, MDense));
```

Нарешті розклад тензору на R факторів виглядає наступним чином:

```
opts.maxit = 1000; % максимальна кількість ітерацій
opts.tol = 1e-4; % порогове значення похибки для
зупинки
t0 = tic;
[A,Out] = ncp(MSparse,R,opts);
time = toc(t0);
```

Вибір кількості факторів *R* проводиться згідно аналогічних міркувань до запропонованих в розділі про факторизацію шестивимірного тензора. Для бінарних зв'язків було обрано 50 факторів, для тернарного - 80.

3.7 Модифікація алгоритму опорних векторів для великих об'ємів негативних прикладів

Метод опорних векторів - це набір схожих алгоритмів виду «навчання із вчителем». Ці алгоритми зазвичай використовуються для задач класифікації та регресійного аналізу. Метод належить до розряду лінійних класифікаторів. Також може розглядатись як особливий випадок регуляризації за А. М. Тихоновим. Особливою властивістю методу опорних векторів є безперервне зменшення емпіричної помилки класифікації та збільшення проміжку. Тому цей метод також відомий як метод класифікатора з максимальним проміжком. Основна ідея методу опорних векторів – перевід вихідних векторів у простір більш високої розмірності та пошук роздільної гіперплощини з максимальним проміжком у цьому просторі. Дві паралельні гіперплощини будуються по

обидва боки гіперплощини, що розділяє наші класи. Роздільною гіперплощиною буде та, що максимізує відстань до двох паралельних гіперплощин (Рис 3.4). Алгоритм працює у припущенні, що чим більша різниця або відстань між цими паралельними гіперплощинами, тим меншою буде середня помилка класифікатора [95].

Припустимо, що точки мають такий вигляд:

$$\{(x_1, c_1), (x_2, c_2), \dots, (x_n, c_n)\},$$

де c_i приймає значення 1 або −1 залежно від того, якому класу належить точка x_i. Кожна точка x_i — це p-мірний дійсний вектор, що зазвичай нормалізується значеннями [0,1] або [-1,1]. Маємо простір:

$$D = \{(x_i, y_i) | x_i \in \mathbb{R}^p, y_i \in \{-1,1\}\}$$

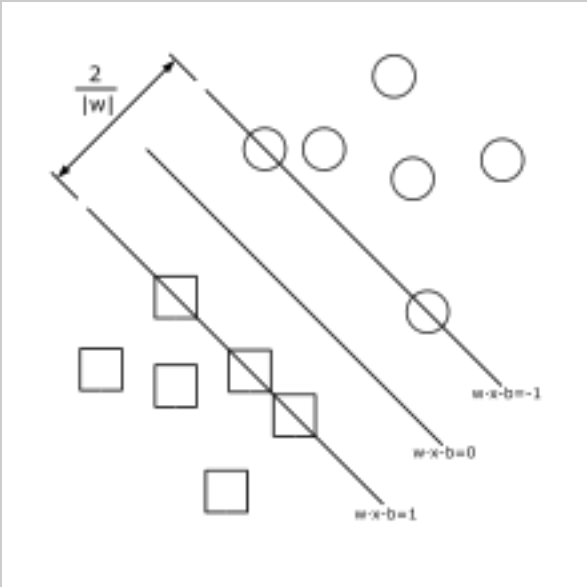


Рис. 3.4. Оптимальна січна гіперплощина для метода опорних векторів, побудована на точках з двох класів. Найближчі точки до паралельних гіперплощин називаються опорними векторами

Якщо точки не будуть нормалізовані, то точка з більшими відхиленнями від середніх значень координат точок матиме сильний вплив на класифікатор. Можемо розглядати це як навчальну колекцію, в якій для

кожного елементу уже задано клас, до якого він належить. Потрібно, аби алгоритм методу опорних векторів класифікував їх таким же чином. Для цього будується роздільна гіперплощина, що матиме такий вигляд:

$$w \cdot x - b = 0$$

Вектор *w* є перпендикуляром до роздільної гіперплощини. Параметр *b* залежить від найкоротшої відстані гіперплощини до початку координат. Якщо параметр *b* дорівнює нулю, то це означає, що гіперплощина проходить через початок координат. А це – обмежує рішення. З метою знайти оптимальне розділення, маємо звернути увагу на опорні вектори та гіперплощини, що є паралельними до оптимальної, та найближчими до опорних векторів двох класів. Можна показати, що ці паралельні гіперплощини можуть бути описані такими рівняннями (з точністю до нормування):

$$w \cdot x - b = 1$$

та

$$w \cdot x - b = -1$$

Якщо навчальна вибірка лінійно роздільна, то ми можемо вибрати гіперплощини таким чином, аби між ними не лежала жодна точка навчальної вибірки, і після цього – максимізувати відстань між гіперплощинами. Можна легко знайти ширину полоси між ними, яка дорівнює $\frac{2}{\|w\|}$ [94]. Тепер потрібно мінімізувати $\|w\|$. Аби виключити всі точки із полоси, маємо переконатись для всіх і, що

$$\begin{cases} w \cdot x - b \geq 1, y_i = 1 \\ w \cdot x - b \leq -1, y_i = -1 \end{cases}$$

Пряма задача оптимізації:

$$\underset{w,b}{argmin}\frac{1}{2}\|w\|^2$$

$$y_i(w \cdot x - b) \geq 1$$

Для переходу до двоїстої задачі оптимізації скористаємося правилами:

$$\|w\|^2 = w^T \cdot w$$

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

Таким чином двоїста задача оптимізації:

$$L = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

$$\alpha_i \geq 0$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

У випадку лінійної не роздільності класів використовують метод опорних векторів з м’яким зазором. В данному випадку пряма задача оптимізації визначається як:

$$\underset{w,b}{argmin}\left\{\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i\right\}$$

$$y_i(w \cdot x - b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

Перейдемо до двоїстої задачі:

$$L = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

$$0 \leq \alpha_i \leq C, i = 1, \ldots, n$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

Надалі в роботі буде використовуватися метод опорних векторів з м'яким зазором та з лінійним ядром.

Характерної властивістю тренувальної вибірки для навчання класифікатора кореферентностей, є те що негативних прикладів є значно більше ніж позитивних. І це зрозуміло : пар не кореферентних сутностей більш ніж кореферентних. Для тренувальної вибірки конференції CoNLL позитивних прикладів було близько 20 тисяч, тоді як негативних майже мільйон. Внаслідок цього, негативні приклади не вміщуються в оперативній пам'яті. Тому було вирішено змінити алгоритм опорних векторів таким чином, щоб кожного разу ми розглядали не всю множину негативних прикладів, а деяку її частину, що знаходиться близько до шуканої гіперплощини. Для цього алгоритм тренування буде постійно змінювати тренувальну вибірку. Це так зване online-навчання.

1. Ініціалізуємо класифікатор деяким початковим значенням.
2. Проаналізуємо наступне речення вхідної тренувальної вибірки, знайдемо пари сутностей для який навчений на даний момент класифікатор видає не правильний результат, та додамо їх до тренувальної вибірки простору ознак.
3. Виконаємо ітерацію оптимізації методом градієнтного спуску для класифікатора на поточному наборі даних
4. Якщо кількість негативних прикладів є більшою за NegativeMax (в нашому випадку = 50000), то виконуємо виконуємо ітерації градієнтного

навчання доки алгоритм не збіжиться. Умовами збіжності можуть бути : різниця між результатом цільових функцій прямої на спряженої задач не стануть менше порога E чи не буде виконано N ітерацій. Після цього всі негативні приклади, що не є опорними векторами методу SVM видаляємо з тренувальної вибірки.

5. Якщо в вхідній тренувальній вибірці залишились неопрацьовані речення, то повернутися до пункту 2.

6. Виконати навчання на існуючих тестових даних з жорсткими умовами збіжності алгоритму.

3.8 Простір ознак машинного навчання

На Рис. 3.5. зображено загальну схему навчання класифікатора для вирішення кореферентностей. Спочатку ми обробляємо розмічену тренувальну вибірку конференції CoNLL. В результаті ми отримуємо набір позитивних прикладів - кореферентних пар сутностей та негативних прикладів - не кореферентних пар сутностей. Для кожного з цих прикладів ми обчислюємо деяких вектор ознак, який буде описаний нижче. Цей вектор задає точку в просторі ознак машинного навчання. Далі застосовується алгоритм навчання методу опорних векторів, що проводить гіперплощину між негативними та позитивними прикладами. Надалі при класифікації для кожної нової пари сутностей ми будемо будувати простір ознак та визначати як далеко від побудованої гіперплощини вона знаходиться, та на основі цієї відстані та деяких порогів визначати чи пара сутностей кореферентна, не кореферентна чи не відомо.

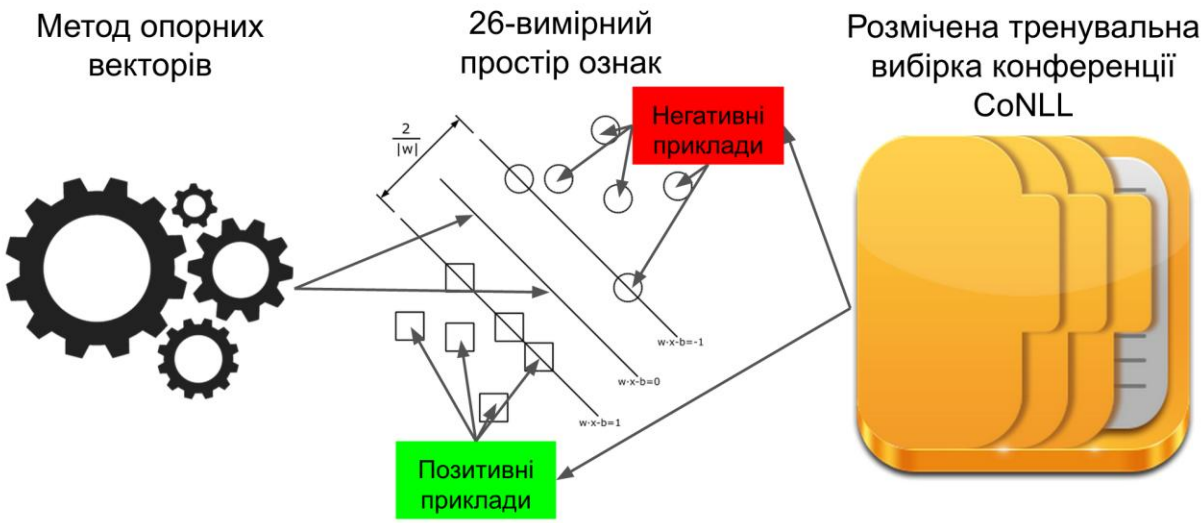


Рис. 3.5. Схема навчання класифікатора методу опорних векторів

Розглянемо детально алгоритм побудови 27-вимірного простору ознак.

1. Антецедент є підметом речення. Визначається за допомогою дерева залежностей за зв’язком nsubj.
2. Антецедент є власною назвою. Визначається за допомогою алгоритму пошуку іменованих сутностей.

3-11. Бінарні ознаки чоловічого, жіночого, середнього роду, множини анафори та антецеденту. Визначається за допомогою модулю морфологічного аналізу.

12-13. Кількість входжень антецедента в текст/дане речення. Аналізуються побудовані кластери кореферентних сутностей.

14. Кількість входжень антецедента до анафоричних зв’язків даного тексту. Аналізуються побудовані кластери кореферентних сутностей та тег морфологічного значення.
15. Кількість власних назв між анафорою та антецедентом. Проходим в циклі від наступного слова після першого антецедента до попереднього слова до другого антецедента, при цьому збільшуємо лічильник кількості іменованих сутностей кожного разу, коли слово є іменованою сутністю.

16-18. Кількість іменників/займенників/речень між анафорою та антецедентом. Проходим в циклі від наступного слова після першого антецедента до попереднього слова до другого антецедента, при цьому збільшуємо відповідні лічильники кількості сутностей кожного разу, коли слово було помічено тегом іменника чи займенника.

19. Порядковий номер серед іменників антецедента в реченні. Визначаємо за допомогою модулю морфологічного аналізу.

20. Порядковий номер серед займенників анафори в реченні. Визначаємо за допомогою модулю морфологічного аналізу.

21. Кількість анафор, для яких було раніше визначена відповідність даному антецеденту на проміжку між антецедентом та анафорою. Визначаємо за побудованим кластером кореферентних сутностей.

22. Кількість анафор, для яких було раніше визначена не відповідність даному антецеденту на проміжку між антецедентом та анафорою. Визначаємо за допомогою допоміжної структури, що містить інформацію про пари не кореферентних сутностей.

23. Оцінка синтаксично-семантичного паралелізму по КП. Визначаємо за наведеним вище алгоритмом оцінки ізоморфізму дерев.

24-25. Оцінка семантичного паралелізму по КП. Визначаємо за сформульованих вище алгоритмом оцінки входження елементу КП до КП речення.

26-28. Слова індикатори. Визначаємо для сформульованим вище принципом роботи слів індикаторів на тензорах керуючих просторів синтаксичних структур природномовних речень.

3.9 Оцінка результатів роботи алгоритму пошуку кореферентностей

Для оцінки результатів скористаємося тестовою процедурою, визначеної для конференції CoNLL [22].

Опис тестової вибірки. Мета побудови анотацій та моделювання кореферентностей в OntoNotes полягає в тому, щоб розширити розмітку поверхневого розуміння текстів, що є ціллю OntoNotes. Наприклад, в "Вона мала гарну пропозицію, і це було одноголосно ухвалене", ми розмічаємо випадок IDENT-кореферентності (ідентичні посилання) між "гарною пропозицією" і "це", які потім забезпечують правильне тлумачення предиката "ухвалено".

Імена, іменникові сутності та займенники можуть бути помічені як кореферентні також. Дієслова, які є кореферентними з іменниковими фразами також можуть бути відзначені як IDENT; наприклад, "виріс" і "сильний ріст" будуть пов'язані в наступному випадку: "Продажі легкових автомобілів зросли на 22%. Сильне зростання слідувало стабільному зростанню впродовж декількох років".

Для того, щоб зберегти анотації коректними на більш високих рівнях розуміння, було визначено кореферентні зв’язки лише тільки всередині документа. Анотації не обмежується будь-яким фіксованим списком типів сутностей, сутності що є універсальними, недовизначеними, або абстрактними не були анотовані.

Для поставленої задачі будемо використовувати англомовну частину даних OntoNotes, яка складається з трохи більше одного мільйона слів з новин (≈450k), журнальних статей (≈150k), трансляції новин (≈200k), субтитрів розмов телеведучих (≈200k) і веб-даних (≈200k). Завдання полягає в тому, щоб автоматично визначати кореферентні особи та події, маючи в наявності правильні розв’язки попередніх кроків синтаксичного аналізу (gold) та в повністю автоматичному режимі.

Опис процедури тестування. В даних OntoNotes розрізняється ідентичні кореферентності та аппозитивні кореферентності. Ми будемо оцінювати системи за результатами лише ідентичних кореферентностей, яка пов'язує всі категорії сутностей і подій разом в еквівалентних класах. На відміну від MUC, або ACE, дані OntoNotes не визначають мінімальне перекриття словосполучень при визначенні кореферентних зв'язків, тому для офіційної оцінки ми будемо розглядати входження сутностей правильним, тільки якщо їм відповідає посимвольно той самий рядок. Оскільки ці дані засновані на ручній розмітці синтаксичних структур, ми плануємо використовувати головні слова із ручної розмітки синтаксичних структур разом з розширеннями, для того щоб зробити більш розслаблену оцінку, де сутності будуть вважається правильними, якщо знайдене словосполучення перетинається з розміченим словосполучення сутності і містить головний слово розміченої сутності. Вибір методи оцінки точності визначення кореферентних зв'язків є складною задачею. Жоден з існуючим не позбавлений недоліків. Тому було обчислено декілька різних метрик - MUC, B-Cubed, CEAF і Blanc. Незважаючи на різноманітність показників, необхідно обчислити одну метрику, щоб визначити систему-переможця. Тому було обрано використовувати MUC-метрику.

Метрика обчислюється за допомогою наступної формули:

$$MUC = \frac{\sum_{i=0}^{n\prime} |C\prime_i \cap C_{j(i)}| - 1}{\sum_{i=0}^{n} |C_i| - 1},$$

$$j(i) = argmax(|C'_i \cap C_j|), j = 0,1,\ldots,n$$

Тут C_i - це множина сутностей в *i*-тому кластері ручної розмітки,

C'_i - це множина сутностей в *i*-тому кластері отриманий за допомогою досліджуваної системи,

j(i) - це номер кластера досліджуваної системи, що має найбільший перетин з кластером *i* з ручної розмітки. Вважається, що кластер з найбільшим перетином є відповідником кластера з ручної розмітки.

Результати тестування за допомогою описаної вище процедури відображені в таблиці 3.1.

Таблиця 3.1. Результати тестування системи розв'язку кореферентностей за допомогою процедури CoNLL-2011

Автор	**Міра MUC**
Vozniuk (Семантико-синтаксична тензорна модель)'2015	64.45
Lee (Stanford Deterministic Coreference Resolution System)'2011	61.03
Cai'2011	57.8
Uryupina'2011	57.63
Klenner'2011	49.86
Irwin'2011	27.21

3.10 Висновки

Вирішення задачі пошуку кореферентих зв'язків починається з пошуку сутностей, між якими надалі будемо знаходити зв'язки. В розділі описано алгоритм обходу дерева виводу, що будує список кандидатів на шукані сутності. Потім список кандидатів фільтрується за деякими правилами.

Система Stanford Deterministic Coreference Resolution System показала найкращі результати на тестовій вибірці кореференції ConLL-2011, що є стандартом для порівняння систем знаходження кореферентних зв'язки. В розділі запропоновано розширення даної системи новими фільтрами, що в архітектурі цієї системи мають назву сит. Ці сита засновані на класифікаторі методу опорних векторів над деяким простором ознак. В розділі запропоновано оптимізацію алгоритму навчання виходячи з властивостей тренувальної вибірки саме цієї задачі.

Для побудови простору ознак машинного навчання було розроблено алгоритми оцінки слів індикаторів, семантичного та синтаксичного паралелізму, що засновані на побудованій в попередньому розділі тензорної моделі керуючих корпусів.

В останньому підрозділі було описано процедуру тестування запропоновану на конференції CoNLL-2011 та представлено результати тестування розробленої системи в порівнянні з іншими відомими системами. Міра точності розробленої системи майже на 3.5% перевищує результати попереднього найкращого результату.

Висновки

Основним результатом дисертації є розробка та математичне обґрунтування нових алгоритмів ідентифікації та аналізу кореферентних зв'язків у природномовних текстах, що має істотне значення для розв'язання фундаментальної задачі комп'ютерної лінгвістики - семантичного аналізу текстів. Для цього було застосовано тензорну модель природної мови, керуючі простори синтаксичних структур речень та методи машинного навчання. При виконанні роботи одержано такі наукові результати:

1. В методі опорних векторів удосконалено алгоритм навчання для класифікації кореферентних сутностей. Це дало змогу одержати більш точні результати класифікації для типової задачі знаходження кореферентностей, коли кількість не кореферентних пар слів на декілька порядків перевищує кількість кореферентних пар.
2. Для підвищення точності класифікації було розроблено розширений простір ознак із додаванням семантико-синтаксичних властивостей. Для обчислення параметрів кореферентних пар в розширеному просторі ознак було вперше побудовано алгоритми оцінки синтаксичного та семантичного паралелізму на основі тензорної моделі.
3. Для тензорної моделі мови розроблено алгоритм побудови багатовимірного масиву опису структур речень та потокову архітектуру системи обробки великих текстів. Тестування системи було проведено на наборі текстів сумарним розміром 100Гб.

4. Розроблено новий алгоритм побудови керуючих просторів синтаксичних структур речень, який дозволив отримати зручне та стисле представлення моделі, зменшити розмірність тензора, отримати більш надійний та стійкий опис семантико-синтаксичних зв'язків між словами. Доведено коректність та обчислено складність алгоритму в термінах швидкодії та пам'яті.
5. Для тестування розробленої системи використовувалася введена конференцією CoNLL-2011 вибірка, яка є стандартом для аналізу роботи систем кореферентних зв'язків. В результаті інтеграції розроблених алгоритмів в одну з найкращих систем визначення кореферентних зв'язків Stanford Deterministic Coreference Resolution вдалось покращити за запропонованою на конференції MUC-6 мірою точність визначення на вказаній тестовій вибірці з 61.03% до 64.45%.

Список використаних джерел

1. A model-theoretic coreference scoring scheme / M. Vilain [and others] // Proceedings of the 6th conference on Message understanding. – 1995. – P. 45-52
2. A Multi-Pass Sieve for Coreference Resolution / H. Lee [and others] // Proceedings of the Conference on Empirical Methods in Natural Language Processing. – 2010. – P. 492-501
3. Alphabetical list of part-of-speech tags used in the Penn Treebank Project [Internet resource] / Access mode: http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html (20.10.2015). – Title from the screen
4. Amit B. Algorithms for scoring coreference chains / B. Amit, B. Baldwin // The First International Conference on Language Resources and Evaluation Workshop on Linguistics Coreference. – 1998. – P. 563 – 566
5. Anisimov A. V. Development of a semantic and syntactic model of natural language by means of non-negative matrix and tensor factorization / A.V. Anisimov, T. G. Vozniuk, V. Taranukha // TSD-2014, Lecture Notes in Artificial Intelligence. – Volume 8655. – Springer Verlag. −2014.− P. 324–335
6. Anisimov A. V. Semantic and Syntactic Model of Natural Language Based on Non-negative Matrix and Tensor Factorization / A.V. Anisimov, T. G. Vozniuk, V. Taranukha // PolTAL-2014, Lecture Notes in Artificial Intelligence. – Volume 8686. – Springer Verlag. −2014. − P. 177–184
7. Anisimov A. V. Semantic and Syntactic Model of Natural Language Based on Tensor Factorization / A.V. Anisimov, T. G. Vozniuk, V.

Taranukha // NLDB-2014, Lecture Notes in Computer Science. – Volume 8455. – Springer Verlag. − 2014. − P. 51–54

8. Apache Commons [Internet resource] / The Apache Software Foundation. – Access mode: https://commons.apache.org/. - Title from screen
9. Benescu R. Associative Anaphora Resolution: A Web-Based Approach // Proceedings of the EACL-2003 Workshop on the Computational Treatment of Anaphora. – 2003. – P. 47-52
10. Bengston E. Understanding the value of features for coreference resolution / E. Bengston, D. Roth // Proceedings of the Conference on Empirical Methods in Natural Language Processing. – 2008. – P. 294-303
11. Berger A. A maximum entropy approach to natural language processing / A. Berger, S. Della Pietra, and V. Della Pietra // Computational Linguistics. – 1996. – Volume 22(1). – P. 39–71
12. Berger A. Modifying agent systems for an open, dynamic agent environment / A. Berger, R. R. Kessler // Proceedings of the second international joint conference on Autonomous agents and multiagent systems. – 2003. – P. 936 - 937
13. Bergsma S. Bootstrapping Path-Based Pronoun Resolution / S. Bergsma, D. Lin // Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics. – 2006. – P. 33-40
14. Björkenstam K. N. SUC-CORE: A Balanced Corpus Annotated with Noun Phrase Coreference // Northern European Journal of Language Technology. – 2013. – #3(10). – P. 19-39
15. Brownell D. SAX2 / D. Brownell. – O'Reilly Media. – 2002. – 240 p.
16. Burges C.J.C. A Tutorial on Support Vector Machines for Pattern Recognition // Data Mining and Knowledge Discovery. – 1998. – Volume 2. – P. 121-167

17. Cardie C. Improving machine learning approaches to coreference resolution / Claire Cardie, Vincent Ng // Proceedings of the 40th Annual Meeting on Association for Computational Linguistics. – 2002. – P. 104-111
18. Chomsky N. Syntactic Structures // The Hague: Mouton. – 1957. – 117 p.
19. Chomsky N. Three models for the description of language // IRE Transactions on Information Theory. – 1956. – Volume 2. – P. 113-124
20. Chowdhury F. M. A controlled greedy supervised approach for co-reference resolution on clinical text / Md. Faisal Mahbub Chowdhury, Pierre Zweigenbaum // Journal of Biomedical Informatics. – 2013. – Volume 36(3). – P. 506 – 515
21. Cichocki A. Hierarchical ALS Algorithms for Nonnegative Matrix and 3D Tensor Factorization / Andrzej Cichocki, Rafal Zdunek, Shun-ichi Amari // Lecture Notes in Computer Science. – Volume 4666. – P 169-176
22. CoNLL-2011 Shared Task: Modeling Unrestricted Coreference in OntoNotes / S. Pradhan [and others] // Proceedings of the Fifteenth Conference on Computational Natural Language Learning. – 2011. – P. 1-27
23. Cortes C. Support-vector networks / Cortes C., Vapnik V. // Machine Learning. – 1995. – Volume 20 (3). – P. 273-297
24. Cutts M. Oxford Guide To Plain English / M. Cutts. – Oxford University Press. – 2010. – 272 p.
25. Elsner M. The same-head heuristic for coreference / M. Elsner and E. Charniak // Annual Meeting of the Association of Computational Linguistics. – 2010. – P. 33-37

26. Finkel J. Enforcing transitivity in coreference resolution / J. Finkel and C. Manning. // Annual Meeting of the Association of Computational Linguistics. – 2008. – P. 45-48
27. First-order probabilistic models for coreference resolution / M. Wick [and others] // Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference. – 2007. – P. 81 – 88
28. Fox B. A. Discourse structure and anaphora: written and conversational English. – Cambridge University Press. – 1993. – 184 p.
29. Frank R. The Acquisition of Anaphora by Simple Recurrent Networks / Robert Frank, Donald Mathis, William Badecker // Language Acquisition. – 2013. – Volume 20. – P. 181-227
30. Ge N. A Statistical Approach to Anaphora Resolution / Niyu Ge, John Hale, Eugene Charniak // Proceedings of the Sixth Workshop on Very Large Corpora. – 1998. – P. 161-170
31. George A. Miller WordNet: A Lexical Database for English // Communications of the ACM. – 1995. – Volume 38(11). – P. 39-41
32. Giménez J. SVMTool: A general POS tagger generator based on Support Vector Machines / Jesús Giménez and Lluís Márquez. // Proceedings of the 4th International Conference on Language Resources and Evaluation. – 2004. – P. 43-46
33. Graph Regularized Non-negative Matrix Factorization By Maximizing Correntropy / Le Li [and others] // Journal of Computers. – 2014. – Volume 9(11). – P. 2570-2579
34. Haghighi A. Simple coreference resolution with rich syntactic and semantic features / A. Haghighi, D. Klein // Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing. – Volume 3. – 2009. – P. 1152-1161

35. Harabagiu S. M. Text and knowledge mining for coreference resolution / S. M. Harabagiu, R. C. Bunescu, S. J. Maiorano // Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies. – 2001. – P. 1-8
36. Hobbs J. R. Resolving Pronoun References // Lingua. – 1978. – Vol. 44. – P. 311-338
37. Iida R. On the Issue of Combining Anaphoricity Determination and Antecedent Identification in Anaphora Resolution / R. Iida, Y. Inui // Proceedings of 2005 IEEE International Conference on Natural Language Processing and Knowledge Engineering. – 2005. – P. 244-249.
38. Ji H. Gender and animacy knowledge discovery from web-scale n-grams for unsupervised person mention detection / H. Ji and D. Lin // Proceedings of the 23rd Pacific Asia Conference on Language, Information and Computation. – 2009. – P. 220-229
39. Jonnalagadda S. R. Coreference analysis in clinical notes: a multi-pass sieve with alternate anaphora resolution modules / S. R. Jonnalagadda [and others] // Journal of the American Medical Informatics Association. – 2012. – Volume 19. – P. 867-874
40. Karthikeyan K. Understanding text using Anaphora Resolution / K. Karthikeyan, V. Karthikeyani // Pattern Recognition, Informatics and Mobile Engineering. – 2013. – P. 346-350
41. Kehler A. Probabilistic coreference in information extraction. Proceedings of the Second Conference on Empirical Methods in Natural Language Processing. – 1997. – P. 163-173
42. Kirkpatrick S. Optimization by Simulated Annealing / Gelatt Jr C. D., Vecchi M. P // Science. – 1983. – Volume 220(4598). – P. 671-680
43. Kudo T. Fast Methods for Kernel-Based Text Analysis / Taku Kudo, Yuji Matsumoto // Proceedings of the 41st Annual Meeting on Association for Computational Linguistics. – 2003. – P. 24-31

44. Landauer T. Introduction to Latent Semantic Analysis / Thomas K Landauer, Peter W. Foltz, Darrell Laham // Discourse Processes. – 1998. – Volume 25(2-3). – P. 259-285
45. Lee H. Deterministic Coreference Resolution Based on Entity-Centric, Precision-Ranked Rules / Lee H. [and others] // Computational Linguistics. – 2013. – Volume 39(4). – P. 885-916
46. Luo X. On coreference resolution performance metrics // Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing. – 2005. – P. 25-32
47. Malouf R. A comparison of algorithms for maximum entropy parameter estimation. // Proceedings of the Sixth Workshop on Natural Language Learning. – 2002. – P. 49–55
48. Marneffe M.C. Generating Typed Dependency Parses from Phrase Structure Parses / Marie-Catherine de Marneffe, Bill MacCartney and Christopher D. Manning // Proceedings of the Fifth International Conference on Language Resources and Evaluation. – 2006. – P. 449-454
49. Màrquez L. Coreference resolution: an empirical study based on SemEval-2010 shared Task / Lluís Màrquez, Marta Recasens, Emili Sapena. // Language Resources and Evaluation. – 2013. – Volume 47. – P. 661-694
50. MATLAB Tensor Toolbox Version 2.6 (released Feb. 6, 2015) [Internet resource] // Sandia National Laboratories [site]. – Access mode: http://www.sandia.gov/~tgkolda/TensorToolbox/index-2.6.html(11.11.2015). – Title from screen
51. McCarthy J. F. Using Decision Trees for Coreference Resolution / Joseph F. McCarthy, Wendy G. Lehnert // Proceedings of the 14th international joint conference on Artificial Intelligence. – 1995. – P. 1050-1055

52. McCord M. Slot grammar: A system for simpler construction of practical natural language grammars // Lecture Notes in Computer Science. – 1990. – P. 118-145
53. Medelyan O. Mining meaning from Wikipedia / Olena Medelyan, David Milne, Catherine Legg, Ian H. Witten //. International Journal of Human-Computer Studies. – 2009. – Volume 67. – P. 716-754
54. Mihalcea R. Semantic Indexing using WordNet Senses / Rada Mihalcea, Dan Moldovan // Proceedings of the ACL-2000 workshop on Recent advances in natural language processing and information retrieval: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics. – 2000. – Volume 11. – P. 35-45
55. Mitkov R. A New, Fully Automatic Version of Mitkov's Knowledge-Poor Pronoun Resolution Method. // Computational Linguistics and Intelligent Text Processing. – 2002. – P. 168-186.
56. Mitkov R. Anaphora resolution (Studies in Language and Linguistics) / Ruslan Mitkov. – Routledge, 2014. – 240 p.
57. Non-negative Matrix and Tensor Factorizations Applications to Exploratory Multi-way Data Analysis and Blind Source Separation / Andrzej Cichocki [and others]. – Wiley, 2009. – 500 p.
58. Parsing With Compositional Vector Grammars / Richard S. [and others] // Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics. – 2013. – P. 141-151
59. Quinlan J. R. C4.5: Programs for Machine Learning / J. R. Quinlan, 1993. – 302 p.
60. Rich E. An architecture for anaphora resolution / Elaine Rich, Susann LuperFoy // Proceedings of the second conference on Applied natural language. – 1988. – P. 18-24
61. Sanyal R. Importance of retrieving noun phrases and named entities from digital library content / Ratna Sanyal, Kushal Keshri, Vidya Nand //

Journal of Zhejiang University SCIENCE C. – 2010. – Volume 11. – P. 844-849

62. Sapena E. A Constraint-Based Hypergraph Partitioning Approach to Coreference Resolution / Emili Sapena, Lluís Padró, Jordi Turmo // Computational Linguistics. – 2013. – Volume 39(4). – P. 847-884
63. Schmolz H. In-Depth Analysis of Anaphora Resolution Requirements / Helene Schmolz, David Coquil, Mario Doller // International Workshop on Database and Expert Systems Applications. – 2012. – P. 174-179
64. Schwartz A. S. A simple algorithm for identifying abbrevation definitions in biomedical text / A.S. Schwartz and M.A. Hearst. // Pacific Symposium on Biocomputing. – 2003. – P. 101-112
65. Shalom L. An Algorithm for Pronominal Anaphora Resolution / Shalom Lappin, Herbert J. Leass // Computational Linguistics. – 1994. – Volume 20 Issue 4. – P. 535-561
66. Soon W. A machine learning approach to coreference resolution of noun phrases / W. Soon, H. Ng, and D. Lim. // Computational Linguistics. – 2001. – Volume 27(4). – P. 521–544
67. Spitkovsky V. I. From baby steps to leapfrog: How “less is more” in unsupervised dependency parsing / V.I. Spitkovsky, H. Alshawi, and D. Jurafsky // Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics. – 2010. – P. 751-759
68. Spitkovsky V. I. A Cross-Lingual Dictionary for English Wikipedia Concepts. / V.I. Spitkovsky, A.X. Chang // Conference on Language Resources and Evaluation. – 2012. – P. 3168-3175
69. Stanford typed dependencies manual [Internet resourse] / Marie-Catherine de Marneffe and Christopher D. Manning // The Stanford Natural Language Processing Group. – 2008. – http://nlp.stanford.edu/software/dependencies manual.pdf (20.10.2015). – Title from the screen

70. Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 shared task / H. Lee [and others] // Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task. – 2011. – P. 28-34
71. Stent A. J. Interaction between dialog structure and coreference resolution / Amanda J. Stent, Srinivas Bangalore // Spoken Language Technology Workshop. – 2010. – P. 342-347
72. Stoyanov V. Conundrums in noun phrase coreference resolution: making sense of the state-of-the-art / Stoyanov V., Gilbert N., Cardie C., and Riloff E. // Proceedings of the Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing. – 2010. – P. 34-46
73. The block-coordinate update method [Internet resource] // UCLA Department of Mathematics [site]. – Access mode: http://www.math.ucla.edu/~wotaoyin/papers/bcu/ (11.11.2015). – Title from screen
74. The Stanford CoreNLP Natural Language Processing Toolkit / C.D. Manning [and others] // Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations. – 2014. – P. 55-60
75. Tim Van de Cruys A Non-negative Tensor Factorization Model for Selectional Preference Induction // Journal of Natural Language Engineering. – 2010. – Volume 16(4). – P. 517-437
76. Tran O. T. Automated reference resolution in legal texts / Oanh Thi Tran, Bach Xuan Ngo, Minh Le Nguyen, Akira Shimazu //Artificial Intelligence and Law. – 2014. – Volume 22. – P. 29-60
77. Van der Sandt Presupposition projection as anaphora resolution // Journal of semantics. – Volume 9 (4). – 1992. – P. 333-377

78. Vapnik V. Statistical Learning Theory / V. Vapnik. – Wiley, 1998. – 768 p.
79. Versley Y. BART: A modular toolkit for coreference resolution / Y. Versley [and others] // Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies. – 2008. – P. 9-12
80. Wang Z. Chinese Pronominal Coreference Resolution Using Decision Tree Plus Filter Rules / Zhiqiang Wang, Lei Li, Ruifan Li, Yixin Zhong // Proceedings of 2005 IEEE International Conference on Natural Language Processing and Knowledge Engineering. – 2005. – P. 587-591
81. Wikipedia: Size of Wikipedia [Internet resource] // Wikipedia : [site]. – Access mode: https://en.wikipedia.org/?title=Wikipedia:Size_of_Wikipedia (11.11.2015). – Title from screen
82. Xu Y. A Block Coordinate Descent Method for Regularized Multiconvex Optimization with Applications to Nonnegative Tensor Factorization and Completion / Y. Xu, Yin W. // Society for Industrial and Applied Mathematics. – 2013. – Volume 6(3). – P. 1758–1789
83. Yang X. Coreference Resolution Using Semantic Relatedness Information from Automatically Discovered Patterns / Xiaofeng Yang, Jian Su // Proceedings of the 45th Annual meeting of the Association for Computational Linguistics. – 2007. – P. 528-535
84. Yangi X. An NP-cluster based approach to coreference resolution / X. Yangi [and others] // Proceedings of the 20th international conference on Computational Linguistics. – #226. – 2004. – P. 101-112
85. Алгоритм автоматизированного разрешенеия анафоры местоимений третьего лица на основе методов машинного обучения [Электронный ресурс] / Толпегин П.В., Ветров Д.П, Кропотов Д.А. //

Диалог. – 2006. – Режим доступа: http://www.dialog-21.ru/digests/dialog2006/materials/html/Tolpegin.htm – Загл. с экрана

86. Анисимов А. В. Управляющее пространство синтаксических структур естественного языка // Кибернетика и системн анализ. – 1990. – №1. – С. 11-17
87. Анисимов А. В. Определение семантических валентностей концептов онтологий с помощью неотрицательной факторизации тензоров больших текстовых корпусов / А. В. Анисимов, А. А. Марченко, Т. Г. Вознюк // Кибернетика и системный анализ. – 2014. – №3. – С. 3-16
88. Анисимов А. В. Управляющие пространства в асинхронных параллельных вычислениях. / Анисимов А.В., Глушков В.М. // Кибернетика. – 1980. – №5. – с.1-9
89. Белецкий А. А. Лексикология и теория языкознания: Ономастика / А. А. Белецкий. – Изд-во Киевского университета. – 1972. – 210 с.
90. Вознюк Т. Г. Алгоритм побудови керуючого простору синтаксичних структур природномовних текстів // Вісник Київського національного університету імені Тараса Шевченка, Серія фізико-математичні науки. – 2014. – №1. – С. 122-127
91. Вознюк Т. Г., Алгоритм побудови шестивимірного тензора для задачі пошуку прихованих семантичних зв’язків в корпусах природномовних // Проблеми програмування. – 2014. – №2-3. – С. 273-278
92. Вознюк Т. Г., Застосування керуючого простору синтаксичних структур природномовних текстів для вирішення проблеми анафори // Вісник Київського національного університету імені Тараса Шевченка, Серія фізико-математичні науки. – 2014. – №2. – С. 101-105

93. Вознюк Т. Г., Побудова класифікатора для вирішення займенникової анафори на основі тензорної моделі // Вісник Київського національного університету імені Тараса Шевченка, Серія фізико-математичні науки. – 2015. – №1. – С. 66-70

94. Воронцов К.В. Лекции по методу опорных векторов [Электронный ресурс] / Вычислительный центр им. А.А. Дородницына: [сайт]. – Режим доступа: http://www.ccas.ru/voron/download/SVM.pdf (11.11.15). – Загл. с экрана

95. Жуков І. А. Динамічна просторово-логічна кластеризації нейронної мережі / І. А. Жуков, Г. М. Кременецький // Інформаційні технології та комп'ютерна інженерія. – 2009. – Випуск 1(14). – С. 39-43

96. Кобзарева Т.Ю. Проблема кореференции в рамках поверхностно-синтаксического анализа русского языка // Компьютерная лингвистика и интеллектуальные технологии. Труды Международной конференции Диалог. – 2003. – С. 278-284

97. Кронгауз М.А. Семантика / М.А. Кронгауз. – РГГУ, 2001. – 400 с.

98. Марченко О.О. Semantic Modification of the Mitkov Algorithm for Anaphora Resolution // Штучний інтелект. – 2012. – № 3. – С. 106-110

99. Марченко О.О. Методи оцінювання семантичної близькості–зв'язності слів природної мови. – Штучний інтелект. – №4. – 2012. – С. 213- 219.

100. Реформаторский А. А. Введение в языкознание. — Просвещение. – 1937. — 542 с.

101. Теньер Л. Основы структурного синтаксиса / Л. Теньер. – Прогресс. – 1988. – 656 с.

102. Хомский Н. Введение в формальный анализ естественных языков / Н. Хомский, Дж. Миллер // Кибернетический сборник. – 1965. – 164 с.

Додаток А. Опис тегів частин мови

Тег	Опис	Приклади
$	долар	$ -$ --$ A$ C$ HK$ M$ NZ$ S$ U.S.$ US$
``	мітка початку цитати	` ``
''	мітка закінчення цитати	' ''
(	початок дужкового виразу	([{
)	кінець дужкового виразу	)] }
,	кома	,
--	тире	--
.	кінець речення	. ! ?
:	двокрапка чи багатокрапка	: ; ...
CC	сполучник	& 'n and both but either et for less minus neither nor or plus so therefore times v. versus vs. whether yet
CD	числівник	mid-1890 nine-thirty forty-two one-tenth ten million 0.5 one forty-seven 1987 twenty '79 zero two 78-degrees

		eighty-four IX '60s .025 fifteen 271,124 dozen quintillion DM2,000 ...
DT	визначник	all an another any both del each either every half la many much nary neither no some such that the them these this those
EX	зовнішнє “there”	there
FW	іншомовне слово	gemeinschaft hund ich jeux habeas Haementeria Herr K'ang-si vous lutihaw alai je jour objets salutaris fille quibusdam pas trop Monte terram fiche oui corporis ...
IN	прийменник	astride among upon whether out inside pro despite on by throughout below within for towards near behind atop around if like until below next into if beside ...
JJ	прикметник або порядковий числівник	third ill-mannered pre-war regrettable oiled calamitous first separable ectoplasmic battery-powered participatory fourth still-to-be-named multilingual multi-disciplinary ...
JJR	вищий ступінь прикметника	bleaker braver breezier briefer brighter brisker broader bumper busier calmer cheaper choosier cleaner clearer closer colder commoner costlier cozier creamier crunchier cuter ...
JJS	найвищий ступінь прикметника	calmest cheapest choicest classiest cleanest clearest closest commonest corniest costliest crassest creepiest crudest cutest darkest deadliest dearest deepest densest dinkiest ...
LS	маркери списку	A A. B B. C C. D E F First G H I J K One SP-44001 SP-44002 SP-44005 SP-44007 Second Third Three Two * a b c d first five four one six three two

MD	модальне допоміжне дієслово	can cannot could couldn't dare may might must need ought shall should shouldn't will would
NN	іменник, загальний, однина	common-carrier cabbage knuckle-duster Casino afghan shed thermostat investment slide humour falloff slick wind hyena override subhumanity machinist ...
NNP	іменник, власний, множина	Motown Venneboerger Czestochwa Ranzer Conchita Trumplane Christos Oceanside Escobar Kreisler Sawyer Cougar Yvette Ervin ODI Darryl CTCA Shannon A.K.C. Meltex Liverpool ...
NNPS	іменник, власний, множина	Americans Americas Amharas Amityvilles Amusements Anarcho-Syndicalists Andalusians Andes Andruses Angels Animals Anthony Antilles Antiques Apache Apaches Apocrypha ...
NNS	іменник, загальний, множина	undergraduates scotches bric-a-brac products bodyguards facets coasts divestitures storehouses designs clubs fragrances averages subjectivists apprehensions muses factory-jobs ...
PDT	квантифікатор	all both half many quite such sure this
POS	маркер родового відмінку	' 's
PRP	особовий займенник	hers herself him himself hisself it itself me myself one oneself ours ourselves ownself self she thee theirs them themselves they thou thy us
PRP$	присвійний займенник	her his mine my our ours their thy your

RB	прислівник	occasionally unabatingly maddeningly adventurously professedly stirringly prominently technologically magisterially predominately swiftly fiscally pitilessly ...
RBR	вищий ступінь прислівника	further gloomier grander graver greater grimmer harder harsher healthier heavier higher however larger later leaner lengthier less-perfectly lesser lonelier longer louder lower more ...
RBS	найвищий ступінь прислівника	best biggest bluntest earliest farthest first furthest hardest heartiest highest largest least less most nearest second tightest worst
RP	частка	aboard about across along apart around aside at away back before behind by crop down ever fast for forth from go high i.e. in into just later low more off on open out over per pie raising start teeth that through under unto up up-pp upon whole with you
SYM	символ	% & ' " ".)). * + ,. < = > @ A[fj] U.S U.S.S.R * ** ***
TO	"to" як маркер інфінітиву	to
UH	вигук	Goodbye Goody Gosh Wow Jeepers Jee-sus Hubba Hey Kee-reist Oops amen huh howdy uh dammit whammo shucks heck anyways whodunnit honey golly man baby diddle hush sonuvabitch ...
VB	базова форма дієслова	ask assemble assess assign assume atone attention avoid bake balkanize bank begin behold believe bend benefit bevel beware bless boil bomb boost brace break bring broil brush build ...
VBD	минулий час	dipped pleaded swiped regummed soaked tidied

	дієслова	convened halted registered cushioned exacted snubbed strode aimed adopted belied figgered speculated wore appreciated contemplated ...
VBG	герундій	telegraphing stirring focusing angering judging stalling lactating hankerin' alleging veering capping approaching traveling besieging encrypting interrupting erasing wincing ...
VBN	дієприкметник минулого часу	multihulled dilapidated aerosolized chaired languished panelized used experimented flourished imitated reunifed factored condensed sheared unsettled primed dubbed desired ...
VBP	теперішній час дієслова	predominate wrap resort sue twist spill cure lengthen brush terminate appear tend stray glisten obtain comprise detest tease attract emphasize mold postpone sever return wag ...
VBZ	теперішній час дієслова, третя особа однини	bases reconstructs marks mixes displeases seals carps weaves snatches slumps stretches authorizes smolders pictures emerges stockpiles seduces fizzes uses bolsters slaps speaks pleads ...
WDT	WH-визначник	that what whatever which whichever
WP	WH-займенник	that what whatever whatsoever which who whom whosoever
WP$	присвійний WH-займенник	whose
WRB	WH-прислівник	how however whence whenever where whereby whereever wherein whereof why

Додаток Б. Таблиця відповідності тегів дерева залежностей типам зв’язку керуючого простору

Тег дерева залежностей	**Опис**	**Тип зв'язку керуючого простору**
abbrev	абревіатура	кільцевий
acomp	прикметниковий додаток	бета
advcl	модифікатор обставини	кільцевий
advmod	модифікатор обставини	кільцевий
agent	агент	бета
amod	прикметниковий модифікатор	зворотній кільцевий
appos	модифікатор додатку	кільцевий
arg	аргумент	кільцевий
aux	допоміжне дієслово	кільцевий
auxpass	пасивна форма допоміжного дієслова	кільцевий
cc	сурядність	кільцевий
ccomp	просте речення з внутрішнім суб’єктом	кільцевий
comp	додаток	кільцевий
conj	сполучення	кільцевий
cop	зв’язка	бета
csubj	суб'єкт - просте речення	альфа
csubjpass	пасивна форма суб'єкта - простого речення	зворотній альфа
dep	залежне слово	кільцевий
det	визначник	зворотній кільцевий
dobj	прямий об'єкт	бета

expl	вставне слово	альфа
goeswith	сателіт	кільцевий
iobj	непрямий об'єкт	кільцевий
mark	маркер	кільцевий
mod	модифікатор	кільцевий
mwe	багатослівний модифікатор виразу	кільцевий
neg	модифікатор заперечення	кільцевий
nn	сполучниковий модифікатор іменника	кільцевий
npadvmod	прислівниковий модифікатор іменника	кільцевий
nsubj	іменниковий суб'єкт (іменник)	зворотній альфа
nsubjpass	пасивна форма іменникового суб'єкту	альфа
num	модифікатор кількості	кільцевий
number	модифікатор елемента числівника	кільцевий
obj	об'єкт	бета
parataxis	паратаксис	кільцевий
pobj	об'єкт прийменника	кільцевий
preconj	слово, що ставиться до сполучника	кільцевий
predet	слово, що ставиться до визначника	кільцевий
prep	прийменниковий модифікатор	кільцевий
punct	сполучення слів знаком пунктуації	кільцевий
quantmod	модифікатор числівника	кільцевий
rcmod	модифікатор відносного прямого речення	кільцевий
ref	посилання	кільцевий
root	корінь речення	кільцевий
sdep	семантична залежність	кільцевий
subj	суб'єкт	кільцевий
tmod	модифікатор часу	кільцевий

vmod	скорочений, не доконаний вербальний модифікатор	кільцевий
xcomp	просте речення з зовнішнім суб’єктом	кільцевий
xsubj	контроль суб'єкту	альфа

Додаток В. Опис тегів дерева виводу

Тег	Опис
Рівень речення	
S	Просте рівноправне речення
SBAR	Просте залежне речення
SBARQ	Запитальне речення прямої мови
SINV	Просте рівноправне речення з непрямим порядком слів в граматичні основі
SQ	Інвертоване "так/ні" просте речення
Рівень словосполучень	
ADJP	Прикметникова група
ADVP	Прислівникова група
CONJP	Сполучникова група
FRAG	Цитата
INTJ	Вигук
LST	Маркер списку
NAC	Not a Constituent; використовується, щоб показати межі деяких модифікаторів іменників (Dr., Col., Rev.) в межах NP
NP	Іменникова група
NX	Використовується всередині складний іменникових фраз, щоб позначити головне слово
PP	Займенникова група
PRN	Дужковий вираз
PRT	Частка
QP	Складений числівник
RPC	Reduced Relative Clause
UCP	Unlike Coordinated Phrase

VP	Дієслівна група
WHADJP	WH-прикметникова група
WHADVP	WH-прислівникова група
WHNP	WH-іменникова група
WHPP	WH-займенникова група
X	Невідомо чи невизначено