UNIVERSITATEA DIN
BUCUREȘTI
VIRTUTE ET SAPIENTIA

Aspecte ale lingvisticii computaționale în producerea de cuvinte

Lucrare de Licență

Maria-Smaranda Pandele

Coordonator: Prof. Dr. Liviu Dinu

Facultatea de Matematică și Informatică, UNIBUC

Rezumat

În această lucrare ne propunem să introducem câteva noțiuni despre lingvistica computațională și producerea de cuvinte. Pornind de la tehnici deja existente, examinăm o metodă prin care să putem produce cuvinte cognate ce lipsesc din anumite limbi romanice, cu precadere pe limba română. Algoritmul prezentat se bazează pe modelări statistice ale unor alinieri de perechi de cuvinte și etimonul lor. Limbile folosite sunt română, italiană, franceză, spaniolă, portugheză și latină. Informația redată de aceste limbi asupra modificărilor structurii cuvintelor este combinată folosind agregarări pe baza dinstanței rank.

Cuprins

Cuprins **iii**

1 Introducere **1**
1.1 Reconstrucția limbajelor folosind metode comparative 2
1.2 Metode computaționale 2
1.3 Latina și limbi romanice moderne 3
1.4 Obiective si abordare 3

2 Reconstrucție de cuvinte latinești **5**
2.1 Prezentare generala 5
2.2 Aliniere 6
2.2.1 Needleman-Wunsch 6
2.3 Câmpuri condiționate aleatoare 7
2.3.1 Generalități 8
2.3.2 Aplicate pe problema reconstrucției cuvintelor 9

3 Agregarea rezultatelor folosind distanța rank **11**
3.1 Clasamente și distanța rank 11
3.1.1 Agregări cu distanța rank 12
3.1.2 Reducerea la o problemă de cuplaj perfect de cost minim 13
3.1.3 Calcularea tuturor agregărilor optime 14
3.2 Determinarea tuturor agregărilor producțiilor de cuvinte . . . 17

4 Experimente și rezultate **19**
4.1 Antrenare 19
4.2 Testare 20
4.3 Evaluare 20
4.3.1 Producerea de cuvinte cognate 20
4.3.2 Reconstruirea cuvintelor latinești neatestate 21

5 Concluzii **23**
5.1 Îmbunătățiri 23

Bibliografie **25**

Capitolul 1

Introducere

Limbajul unei țări se află în continuă schimbare datorită mai multor cauze precum cauze sociale, economice, migrația populației, progresele în știință, tehnologie și medicină. De-a lungul timpului, modificările s-au petrecut inevitabil și nu au fost neapărat reglementate de experți în domeniu. Astfel a apărut o nouă ramură a lingvisticii ce studiază sistematic aceste transformări încercând să găsească șabloane, reguli și să pună o ordine asupra schimbărilor lexicale, fonetice, semantice și sintactice. Câteva exemple clasice ar fi determinarea etimologiei unui cuvânt (românescul *genunchi* provine din latinescul *genuclum*) sau determinarea similarității limbajelor.

De cele mai multe ori, popoare ce vorbeau aceeași limbă s-au despărțit și au evoluat diferit, apărând limbaje derivate. Ne putem da seama de gradul de rudenie a acestora prin identificarea formelor **cognate** (grupuri de cuvinte ce au derivat din același etimon). În alte cazuri, limbile au împrumutat cuvinte între ele fie luându-le ca atare (cuvântul japonez *sushi*), fie modificând forma lor (romanescul *cafea* din turcescul *kahve*). Astfel, se pot descoperi chiar relații de natură istorică între limbi și popoarele ce le folosesc. Spre exemplu, există peste 200 de cuvinte ce se regăsesc în toate limbile romanice moderne mai puțin limba română. Este greu de crezut ca o parte din aceste cuvinte nu au existat în lexicul limbii române la un moment dat. Fischer[9] a identificat câteva cauze ce au condus la dispariția acestor cuvinte și înlocuirea acestora prin cuvinte cu alte etimologi: cauze externe, social-economice, schimbarea ocupațiilor romanilor, întreruperea României cu lumea Occidentala, dezvoltarea limbii române departe de nucleul romanic și asa mai departe.

1.1 Reconstrucția limbajelor folosind metode comparative

Lingvisticii istorici se ocupă cu cercetarea schimbărilor limbilor de-a lungul timpului. O preocupare majora a acestora o reprezintă producerea de cuvinte înrudite. Ei folosesc metode comparative prin care analizează modificările limbajelor efectuând comparații între limbaje înrudite pentru a deduce, printr-o inginerie inversă, proprietăți ale limbii strămoș comune.[25] Practic, se încearcă determinarea grupurilor de cuvinte cognate și găsirea unor reguli prin care au fost obținute din limba strămoș. Acestea nu sunt neapărat atestate, cele mai multe dintre ele denumindu-se **proto-limbaje**. În consecința, rezultatele obținute sunt foarte greu de demonstrat pentru ca nu exista dovezi arheologice concrete.

Asemenea metode au dat rezultate satisfăcătoare, reușind să determine structura familiilor de limbi europene. Mai mult, reconstrucția proto-limbajul **Indo-European** considerat cel mai vechi limbaj cunoscut, a fost posibilă prin punerea în corelație cu limbajele derivate din acesta (proto-limbajul German, proto-limbajul Indo-Iranian).[24] În plus, metodele comparative au avut avut succes și în analiza altor familii de limbaje de pe alte continente.

1.2 Metode computaționale

În principal, tehnicile comparative au mai multi pași efectuați manual de lingviști și presupun multe ore de muncă. În perioada în care calculatoarele nu erau încă inventate, căutarea cuvintelor în cărți, dicționare și prelucrarea acestora era o muncă nu numai obositoare și repetitivă dar și dificilă, fiind nevoie de atenție continuă. Însă, odată cu apariția metodelor computaționale, se pune accentul pe determinarea automată cuvintelor înrudite precum în [12] sau [20]. Aceste soluții sunt departe de a înlocui un expert în domeniul lingvisticii și își propun mai mult să vină în ajutorul acestuia pentru a facilita dezvoltarea în profunzime a domeniului.

Metodele computaționale doar automatizează procesul păstrând ideea pornirii de la formele moderne ale cuvintelor pentru a le reconstrui pe cele vechi din care provin. Există numeroase date întreținute activ de către specialiști ce determină conexiuni între cuvinte din mai multe limbi moderne. Spre exemplu, corpusul Europarl pentru limbile oficiale vorbite în Uniunea Europeana sau WordNet (Fellbaum, 1998) pentru limba engleza. Asemenea resurse vaste sunt esențiale în aplicarea unor tehnici comparative automate. Pe baza lor se pot determina și aplica reguli de producție a cuvintelor într-un mod formal, fără prea mari intervenții din partea lingvisticilor pentru asa zisele *excepții*.

Este dificil de prezis cum anume a fost modificat un cuvânt pentru a ajunge în forma lui actuală. Deși se presupune că ar exista șabloane și reguli în evoluția unui cuvânt din etimonul sau, sunt și exemple care s-au detașat semnificativ de strămoșul lor: latinescul *umbilicu(lu)s* a ajuns la forma de *buric* în română, *nombril* în franceză și *umbigo* în portugheză.

Detecția automată a cuvintelor cognate poate fi formulată ca o problemă de clasificare prin învățare automată. O serie de atribute s-au dovedit a fi de succes de-a lungul timpului precum *n*-gramele, distanțe de editare, cel mai lung subșir comun etc. Jager si Sofroniev [11] determină perechi cognate folosind un clasificator SVM iar apoi probabilități și distanțe pentru a grupa cuvinte în grupuri cognate. Rama [19] aplica o rețea neurală convoluțională. Ciobanu și Dinu [4] folosesc câmpuri aleatoare condiționate reușind să reconstruiască proto-cuvinte din seturi cognate incomplete.

1.3 Latina și limbi romanice moderne

Română(ro), Italiană(it), Franceză(fr), Spaniolă(es), Portugheză(pt) sunt doar câteva dintre limbile moderne ce au evoluat din latina, în special din dialectul vulgar (latina vorbita de oamenii de rand în Imperiul Roman). Bineînțeles, provenind din aceeași limbă, se pot pune foarte multe probleme de natură lingvistică precum gradul de similaritate dintre acestea, găsirea grupurilor de cuvinte cognate, reconstrucția de cuvinte latinești neatestate și asa mai departe.

Tranziția de la latină la o limbă romanică moderna s-a efectuat prin schimbări majore de vocabular, sintaxă și fonologie. Aceste schimbări sunt mai mult sau mai puțin similare pentru fiecare limbă romanică modernă. În principal, s-a urmărit simplificarea vocabularului prin eliminarea arhaismelor și a excepțiilor în favoarea regulilor clare, reducerea sinonimelor, stabilirea clară a sensurilor cuvintelor (conform [23]).

1.4 Obiective si abordare

Pentru limbile romanice *ro, it, fr, es, pt* există un dicționar etimologic în [22] pornind de a limba latină. Acest set este incomplet și ne propunem sa-l completam folosind o metoda computațională bazata pe lingvistica comparativă. Evaluarea rezultatelor va fi făcută în mare parte manual, aceste cuvinte nefiind atestate. Metoda computațională are la baza ideea prezentată în [4].

Astfel, pornind de la mai multe perechi cognate din limbi romanice moderne, se va aplica o metodă de reconstrucție a cuvintelor bazată pe etichetarea secvențelor și câmpuri aleatoare condiționate. Fiecare limba modernă va fi analizată separat în raport cu limba latină. Apoi, rezultatele din toate

limbile romanice vor fi combinate folosind agregări pe baza distanței rank [6]. Toate acestea vor fi detaliate pe larg în următoarele capitole.

Capitolul 2

Reconstrucție de cuvinte latinești

Pornim de la formele cuvintelor din limbile romanice moderne. Fiind date mai multe perechi de cuvinte cognate vrem să deducem forma latinescului strămoș comun. Aplicăm o metodă similară cu cea folosită în [4]. Ca și în [4], ne bazăm pe faptul că modificările ortografice sunt strâns legate de evoluția cuvintelor. Deci încercam să reconstruim proto-cuvinte din forma ortografică a cuvintelor moderne.

În final, dorim să obținem o listă cu n cele mai bune predicții pe care mai apoi sa le prelucrăm într-o manieră atât automată cât și manuală pentru a obține cele mai bune rezultate.

2.1 Prezentare generala

Dat fiind mai multe seturi de date de cuvinte cognate din limbi romanice moderne, metoda de reconstrucție va încerca să aproximeze forma latinescului de proveniență. Vom folosi: *Română*, *Italiană*, *Franceză*, *Spaniolă*, *Portugheză*, iar seturile de date vor avea forma $(\textit{cuvânt modern}, \textit{cuvânt latinesc})$. Din acestea, modelul va învăța pe baza câmpurilor condiționate aleatoare (*conditional random fields* sau prescurtat CRF) diverse schimbări de ortografie suferite de cuvintele latinești pentru a le forma pe cele moderne. Apoi, vom aplica o tehnica de agregare a acestor rezultate pentru a combina informație din toate limbile. În final, vom folosi aceste schimbări pentru a oferi variante de cuvinte ce completează anumite seturi de date.

Pașii algoritmului pentru o anumită limbă romanică modernă sunt:

1. Pentru fiecare pereche $(\textit{cuvânt modern}, \textit{cuvânt latinesc})$, vom alinia cele două cuvinte pentru a înțelege ce semne ortografice s-au păstrat, schimbat sau elidat.

2. Pregătim antrenarea sistemului CRF: extragem caracteristici din alinierile fiecărei perechi.

3. Rulăm sistemul CRF și obținem liste de n cele mai bune producții sortate în funcție de probabilitatea lor.

2.2 Aliniere

Avem perechi de tipul $(cuvânt\ modern, cuvânt\ latinesc)$ pe care vrem să le aliniem. Nu orice aliniere ne oferă informație validă. Avem nevoie de așa numitele alinieri optime, în care numărul de diferențe dintre cele două cuvinte este minim. Vom aplica algoritmul de aliniere Needleman-Wunsch[18] din bioinformatică, folosit cu succes și în probleme de procesare al limbajului natural.

2.2.1 Needleman-Wunsch

Algoritmul de aliniere Needleman-Wunsch provine din bioinformatică, mai exact din alinierea secvențelor de proteine sau nucleotide. Determinarea alinierilor se face printr-o tehnică de programare dinamică. Așadar, problema inițială va fi împărțită în subprobleme fie deja calculate, fie mai ușor de calculat. De fiecare dată când vom spune alinieri ne vom referi doar la alinieri de tip Needleman-Wunsch.

Avem doua șiruri $a = a_1a_2...a_n$ și $b = b_1b_2...b_m$ de caractere de lungime n respectiv m. Vrem sa aliniem șirul b pentru a se potrivi cu șirul a.

Există 3 tipuri de operații într-o aliniere la o anumită poziție i:

1. **Potrivire**, caracterele aliniate se potrivesc: $a_i = b_i$

2. **Nepotrivire**, caracterele aliniate nu se potrivesc: $a_i \neq b_i$

3. **Spațiu**, caracterul din primul șir nu se aliniază cu niciun caracter din al doilea șir sau invers

Se observă că în cazul operației de tipul 3, caracterele din dreapta poziției curente i se vor deplasa cu o poziție.

Fiecare dintre operațiile de mai sus are un anumit cost. În problema noastră de aliniere a unui cuvânt latinesc cu un cuvânt modern, vom considera costul unei operații de **Potrivire** ca fiind 0, iar costul unei operații de **Nepotrivire** sau **Spațiu** ca fiind 1. Astfel, alinierea optimă va fi cea cu costul minim. În procesul de potrivire nu vom lua în considerare diacriticele. Literele ce conțin astfel de semne vor fi considerate la fel cu literele de bază (spre exemplu, caracterul *è* poate fi potrivit cu *e*).

Problema se rezolvă folosind tehnica programării dinamice. Definim două matrici bidimensionale cu $n+1$ și $m+1$ coloane numerotate de 0 la n respectiv de la 0 la m:

$$D_{i,j} = \text{costul minim pentru a alinia prefixul } a_1a_2...a_i \text{ din șirul a}$$
$$\text{cu prefixul } b_1b_2...b_j \text{ din șirul b}$$
$$P_{i,j} = \text{ultima operație efectuată } \textbf{Potrivire} \text{ , } \textbf{Nepotrivire} \text{ sau } \textbf{Spațiu}$$

Considerăm faptul că prefixul vid va fi reprezentat de poziția fie cu linia 0 (în cazul în care prefixul vid provine din șirul a), fie cu coloana 0 (în cazul în care prefixul vid provine din șirul b).

Relația de recurență este:

$$D_{0,j} = j, \forall j = 0, ..., m$$
$$D_{i,0} = i, \forall i = 0, ..., n$$
$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} & , a_i = b_j \textbf{ Potrivire} \\ D_{i-1,j-1} + 1 & , a_i \neq b_j \textbf{ Nepotrivire} \\ D_{i-1,j} & , \textbf{ Spațiu} \\ D_{i,j-1} & , \textbf{ Spațiu} \end{cases}$$

Scorul alinierii se va afla pe poziția $D_{n,m}$. Pentru a reconstitui alinierea, la fiecare pas din recurență trebuie să reținem în matricea P ultima operație efectuată. Acum, pornim cu doi indici $i = n$ și $j = m$. Dacă pentru a rezolva subproblema determinată de prefixele $a_1a_2...a_i$ și $b_1b_2...b_j$ am folosit ca și ultimă operație:

- **Potrivire**: aliniem caracterul de pe poziția i din a cu cel de pe poziția j din b; decrementăm indicii i și j
- **Spațiu**: deducem dacă spațiul este în șirul a sau b și refacem indicii corespunzător (dacă spațiul provine din șirul a atunci decrementăm indicele j, iar dacă spațiul provine din șirul b decrementăm indicele i
- **Nepotrivire**: aliniem caracterele diferite de pe pozițiile i din a și j din b; decrementăm indicii i și j

2.3 Câmpuri condiționate aleatoare

În continuare vrem să învățăm care sunt schimbările ortografice produse în evoluția unui cuvânt modern știind etimonul său. În secțiunea precedentă am aliniat cuvintele pentru a determina ce modificări au suferit caracterele. Folosim un algoritm de învățare automată pentru a studia șabloane de schimbări ortografice dintre fiecare limbă modernă și limba latină.

Propunem o metodă bazată de câmpuri condiționate aleatoare, întrucât acestea au dat rezultate satisfăcătoare în generarea transliterațiilor [10] și în producția de cuvinte cognate [2].

Vom explica mai întâi câteva noțiuni despre câmpurile condiționate aleatoare iar apoi le vom aplica pe problema reconstrucției de cuvinte.

2.3.1 Generalități

Câmpurilor condiționate aleatoare (*conditional random fields* sau prescurtat CRF) sunt o metodă de modelare statistică pentru a face predicții structurate.

Lafferty, McCallum si Pereira[14] sunt primii care explică această structură. Ei introduc un nou cadru pentru construirea modelelor probabilistice pentru segmentarea și etichetarea datelor secvențiale. Până în acel moment, astfel de probleme erau rezolvate prin modele Markov ascunse și gramatici stocastice.

Conform [14], fie X o variabilă aleatoare peste secvențe de date ce trebuie etichetate și Y o variabilă aleatoare peste etichetele corespunzătoare. Construim un model de probabilități condiționate $P(Y|X)$ din perechile de secvențe de date și etichete.

Definiție 2.1 *Fie $G = (V, E)$ un graf astfel încât $Y = (Y_i)_{i \in V}$ (nodurile grafului reprezintă indecșii etichetelor Y). (X, Y) se numește câmp condiționat aleator dacă variabila aleatoare Y_v condiționată de X respectă proprietatea Markov raportat la graful G:*

$$P(Y_v|X, Y_w, w \neq v) = P(Y_v|X, Y_w, w \sim v)$$

unde $w \sim v$ înseamnă că există muchia (w, v) în E.

Observăm că CRF-ul este global condiționat de variabila X. Graful G este nedirecționat, construit diferit în funcția de atributele secvențelor de etichetat dar de cele mai multe ori are forma unui lanț sau arbore.

Pentru atribute, vom defini o funcție pentru a reprezenta caracteristici ale secvențelor de date. Valoarea acestor funcții se calculează în funcție de problema pe care dorim să o rezolvăm. Le vom nota cu f_k, g_k etc. Spre exemplu $g_k(v, y|S, x)$ va fi adevărat dacă cuvântul X începe cu o majusculă, iar eticheta Y este "substantiv propiu".[14] $y|S$ reprezintă setul de componente ale lui Y asociate cu nodurile subgrafului G. Funcțiile de atribute trebuie calculate înaintea aplicării sistemului CRF.

Atribuim fiecărui atribut o anumită pondere λ_i (pentru atributele în raport cu muchile grafului G) și μ_i (pentru atributele reportate la nodurile grafului G). Acestea sunt valorile pe care algoritmul trebuie să le învețe pentru a face mai apoi predicțiile.

Exista doua structuri de grafuri pe care se pretează să aplicăm sistemul CRF: cele lanț și cele arborescente.

Formula probabilităților condiționate pentru structura de arbore este:

$$P(Y|X) \propto \exp(\sum_{e \in E,k} \lambda_k f_k(e, y|e, x) + \sum_{v \in V,k} \mu_k g_k(v, y|v, x))$$

Pentru a determina formula probabilităților condiționate pentru structura de lanț vom nota $start = Y_0$ și $stop = Y_{n+1}$ Probabilitățile condiționate le vom calcula într-o matrice M astfel:

$$M_i(y', y|x) = exp(\Lambda_i(y', y|x))$$
$$\Lambda_i(y', y|x) = \sum_k \lambda_k f_k(e_i, Y|e_i = (y', y), x) + \sum_k \mu_k g_k(v_i, Y|v_i = y, x).$$

unde, e_i este muchia cu etichetele (Y_{i-1}, Y_i) și v_i este nodul corespondent lui Y_i.

Pentru a estima parametrii λ_i și μ_i se folosesc algoritmi iterativi pentru a maximiza rezultatele pe un set de date de antrenare. Algoritmii sunt prezentați în [14].

Acum pute scrie formula pentru probabilitățile condiționate pentru un graf de tip lanț:

$$P(Y|X) \propto \frac{\prod_{i=1}^{n+1} M_i(y_{i-1}, y_i|x)}{(\prod_{i=1}^{n+1} M_i(x))_{start,stop}}$$

Precum și în alte modele statistice ce folosesc probabilități condiționate (spre exemplu: Naive Bayes), predicția se face prin:

$$\hat{Y} = argmax_Y(P(Y|X))$$

Putem lua chiar și primele n cele mai bune probabilități pentru face o analiză în profunzime.

2.3.2 Aplicate pe problema reconstrucției cuvintelor

Ne întoarcem la problema de predicție a cuvintelor latinești pornind de la cuvinte moderne folosind sisteme CRF. Secvențele de date vor fi cuvintele moderne. Atributele vor fi n-grame din cuvintele de intrare, extrase din ferestre de dimensiune w.

Etichetele se calculează folosind algoritmul de aliniere, utilizând caracterele ce se potrivesc în cuvântul modern și cuvântul latinesc. Dar, pentru că cele două cuvinte nu sunt complet identice, în cazul unei inserări vom pune caracterul adăugat la eticheta precedentă (nu putem asocia aceasta literă cu niciuna din cuvântul de intrare). În cazul unui spațiu, apărut în cuvântul latinesc se produce practic un fenomen de elidare. Asociem caracterului respectiv din cuvântul sursă, o etichetă nouă (spre exemplu $-$) pentru a semnifica dispariția acestuia.

Sistemul are nevoie de margini la capetele cuvintelor, în cazul în care avem inserții produse la începutul sau sfârșitul cuvântului, adică o eticheta "precedentă" cu care să asociem aceste litere noi. Așadar, fiecare cuvânt va fi extins prin adăugarea a două caractere **B** și **E** la începutul și sfârșitul acestuia.[4] Orice literă nouă adăugata la început sau la sfârșit, va fi asociată cu aceste 2 litere speciale.

Folosim 5 astfel de sisteme CRF pentru fiecare limba modernă *română*, *italiană*, *franceză*, *spaniolă*, *portugheză* pusa în raport cu *limba latină*. Fiecare sistem va calcula liste de cele mai bune n cuvinte latinești sortate în funcție de probabilitate. Pentru a profita de toate limbile moderne propunem o metodă de combinare a rezultatelor sistemelor CRF bazată pe agregări folosind distanța rank.

Capitolul 3

Agregarea rezultatelor folosind distanța rank

Am văzut cum putem obține producții de cuvinte combinând câte o singură limbă romanică cu limba latină. Pentru a îmbunătății rezultatele vrem să folosim informația din mai multe limbi romanice moderne. Astfel, fiecare clasificator întoarce o listă ordonată de cuvinte latinești, pe prima poziție aflându-se etimonul latinesc cu cea mai mare probabilitate. Prin agregarea acestora cu o anumită metrică vom obține o lista sortata cu mai probabile cuvinte latinești. Metrica folosita este distanta rank [6] întrucât s-au obținut rezultate bune în alte probleme de natură lingvistică precum determinarea similitudinii silabice a limbilor romanice [1], [7].

În primul rând vom defini ce înseamna o listă ordonată de elemente. În al doilea rând, vom explica distanța rank între două clasamente și între un clasament și o mulțime de clasamente. Apoi vom prezenta o metodă de aflare a tuturor agregărilor unei mulțimi de mai multe clasamente folosind distanța rank. În final, vom prelucra mulțimea de agregări bazat pe un sistem de vot pentru a determina **o singură listă ordonată** de posibile etimoane latinești.

3.1 Clasamente și distanța rank

Un *clasament* este o listă ordonată de obiecte după un anumit criteriu, pe prima poziție aflându-se cel cu cea mai mare importanță. În unele situații se pune problema găsirii unui clasament cât mai apropiat de o mulțime de mai multe clasamente. Pentru a rezolva această problemă trebuie să definim mai întâi ce înseamnă distanța dintre două clasamente sau dintre un singur clasament și o mulțime de clasamente.

Există mai multe metrici folosite cu succes în diverse aplicații: distanța *Kedall tau*, *Spearman footrule*, *Levenshtein*, dar noi vom folosi distanța *rank* introdusa

in articolul [6]. În întregul capitol vom folosi următoarele notații:

- $U = \{1, 2, ..., n\}$ o mulțime finită de obiecte numită univers
- $\tau = (x_1 > x_2 > ... > x_d)$ un clasament peste universul U
- $>$ o relație de ordine strictă reprezentând criteriul de ordonare
- $\tau(x)$ = poziția elementului $x \in U$ în clasamentul τ dacă $x \in \tau$, numerotând pozițiile de la 1 începând cu cel mai important obiect din clasament

Dacă un clasament conține toate elementele din univers, atunci el se va numi *clasament total*. Asemănător, dacă conține doar o submulțime de obiecte din univers, atunci îl vom numi *clasament parțial*.

Notăm ordinea elementului x în τ astfel:

$$ord(\tau, x) = \begin{cases} |n + 1 - \tau(x)| & , x \in \tau \\ 0 & , x \in U \setminus \tau \end{cases} \tag{3.1}$$

Definiție 3.1 *Fie τ și σ două clasamente parțiale peste același univers U. Atunci distanța rank va fi*

$$\Delta(\tau, \sigma) = \sum_{x \in \tau \cup \sigma} |ord(\tau, x) - ord(\sigma, x)| \tag{3.2}$$

Se observă faptul că, în calculul distanței rank, se ia în considerare ordinea definită mai sus și nu poziția. În primul rând, cum primele poziții sunt cele mai importante, distanța dintre două clasamente trebuie sa fie cu atât mai mare cu cât diferă mai mult începutul lor.[16] În al doilea rând, definiția funcției *ord* pune accentul pe lungimea clasamentelor întrucât putem presupune că un clasament mai lung a fost obținut în urma comparării mai multor obiecte din univers. Deci ordinea elementelor este mai solidă. Spre exemplu, dacă două clasamente de lungimi diferite au același element pe prima poziție, există totuși o diferență a ordinii obiectului în cele două liste, diferență ce contribuie la calculul distanței rank totale.[8]

3.1.1 Agregări cu distanța rank

O *agregare de clasamente* reprezintă un singur clasament σ astfel încât o anumită metrică de la acesta la mulțimea de liste de agregat T este minimă. Raportându-ne la distanța rank avem[6]:

Definiție 3.2 *Fie un set de clasamente $T = \{\tau_1, \tau_2, ..., \tau_m\}$ dintr-un univers U și $\sigma = (\sigma_1 > \sigma_2 > ... > \sigma_k)$ un clasament astfel încât $\sigma_i \in U, \forall 1 \leqslant i \leqslant k$. Definim distanța rank de la σ la T astfel:*

$$\Delta(\sigma, T) = \sum_{i=1}^{m} \Delta(\sigma, \tau_i) \tag{3.3}$$

Definiție 3.3 *Se numește mulțime de agregari de lungime k a mulțimii T folosind distanța rank, setul* $A(T,k) = \{\sigma = (\sigma_1 > \sigma_2 > ... > \sigma_k) | \sigma_i \in U, \forall 1 \leqslant i \leqslant k$, *si* $\Delta(\sigma, T)$ *este minim posibila* $\}$

Problemă 3.4 *Fie U un set de obiecte și* $T = \{\tau_1, \tau_2, ..., \tau_m\}$ *o mulțime de clasamente peste universul U. Vrem să determinăm mulțimea de agregări* $A(T,k)$ *pentru un k fixat.*

Construim următoarele matrici bidimensionale $W^k(i,j)$ cu n linii și n coloane. Fiecare celulă din fiecare matrice reprezintă costul total din distanța rank de la un clasament σ, de lungime l, către o mulțime $T = \{\tau_1, \tau_2, ..., \tau_m\}$ fixată indus de plasarea elementului $x_i \in U$ pe poziția j în σ [8]. Se observă faptul că un clasament peste universul U definit mai sus poate avea lungimea maxim n. Rezultă că numărul de coloane al matricilor W^t este egal cu n.

$$W^k(i,j) = \begin{cases} \sum_{p=1}^{m} |ord(p,i) - k + j| & , j \leqslant k \\ \sum_{p=1}^{m} |ord(p,i)| & , j > k \end{cases} \tag{3.4}$$

Propoziție 3.5 *Distanța de la un clasament* $\sigma = (\sigma_1 > \sigma_2 > ...\sigma_k)$ *la mulțimea T este*

$$\Delta(\sigma, T) = \sum_{x_i \in U \cap \sigma} W^k(i, \sigma(x_i)) + \sum_{x_i \in U \setminus \sigma} W^k(i, k+1)$$

unde n reprezintă numărul de obiecte din univers, iar $k < n$.

Se observă faptul că, în cazul în care σ conține toate elementele din U, deci cazul $k = n$, formula se devine

$$\Delta(\sigma, T) = \sum_{x_i \in U \cap \sigma} W^k(i, \sigma(x_i)) \tag{3.5}$$

3.1.2 Reducerea la o problemă de cuplaj perfect de cost minim

Fiecare matrice W^l din secțiunea precedentă este calculată în mod independent de celelalte Deci putem determina doar o singură matrice pentru o anumită lungime fixată l. Astfel, problema se reduce la găsirea unui clasament σ ce minimizează formula (3.5). Formal:

Problemă 3.6 *Fiind dată o matrice pătratică W,* $W = (w_{i,j})_{1 \leqslant i,j \leqslant n}$ *vrem să determinăm următoarea mulțime:*

$$S = \{(i_1, i_2, ..., i_k) | (i_p \neq i_j, \forall p \neq j), (i_j \in U) \text{ } \textit{și} \text{ } \sum_{j=1}^{n} w_{i_j,j} \text{ } \textit{este minim}\}$$

Problema de mai sus se aseamănă cu o problemă de cuplaj de dimensiune k de cost minim întrucât vrem să formăm perechi între obiectele dintr-un univers și pozițiile unui clasament de tip agregare, iar fiecare combinație are un anumit cost. Practic $(i_1, i_2, ..., i_n)$ reprezintă o permutare a elementelor din U.

O soluție pentru a rezolva problema precedentă este aplicarea algoritmul Ungar prezentat de Khun [13]. Altfel, putem considera matricea W ca fiind o matrice de costuri într-un graf bipartit G pe care aplicăm un algoritm clasic de găsire a cuplajului maxim de cost minim (din care luam doar k perechi). Conform [21] această problemă poate fi rezolvată în timp polinomial $\mathcal{O}(n^3)$ construind o rețea de flux cu capacități convenabile și prin găsirea unor drumuri de augmentare minime, din punct de vedere al costului, folosind algoritmul lui Dijkstra[5].

Toate aceste rezolvări determină o singură agregare dar nu și pe toate, adică mulțimea $A(T, k)$ din definiția 3.3. În continuare prezentăm o metoda de determinare a tuturor agregărilor bazată pe găsirea tuturor cuplajelor *perfecte* de cost minim dintr-un graf, metoda prezentată în [15]. Algoritmul rulează într-un timp polinomial. Particularizăm problemele și algoritmii din articolul *A generalization of the assignment problem, and its application to the rank aggregation problem* [15] pentru Problema 3.6.

3.1.3 Calcularea tuturor agregărilor optime

Reamintim faptul că dorim să calculăm mulțimea de agregări $A(T, k)$, știind costul plasării fiecărui element pe fiecare poziție, memorat în matricea W^k calculată la (3.4). Reformulăm problema în elemente de teoria grafurilor. Astfel, asociem Problemei 3.6 un graf $G = (V, E, c, w)$, unde V reprezintă mulțimea de noduri, E este mulțimea de muchii iar $c\colon E \to \mathbb{N}$ și $w\colon E \to \mathbb{N}$ reprezintă capacitatea unei muchii respectiv costul acesteia. Legăturile între Problema 3.6 și graful G sunt:

- $V = \{src, dst\} \cup U \cup \{1, 2, ..., k, k+1\}$
- $E = \{(src, x_i) | x_i \in U\} \cup \{(x_i, j) | x_i \in U \text{ si } j = 1, ..., k\} \cup \{(j, dst) | j = 1, ..., k\}$
- $c(muchie) = 1, \forall muchie = (x, j) \in E, j \neq k+1$
- $c(muchie) = \infty, \forall muchie = (x, k+1) \in E$
- funcția w astfel:
  - $w((src, x_i)) = 0, \forall x_i \in U$
  - $w((x_i, j)) = W^k(i, j), \forall x_i \in U, j = 1, ..., k+1$
  - $w((j, dst)) = 0, j = 1, ..., k+1$

Potrivirea unui element x cu poziția $k+1$ va semnifica excluderea acestuia din agregare ce afectează distanța rank dintre un clasament ce nu conține elementul x și mulțimea întreagă T. Se poate calcula ușor în acest graf un flux maxim de cost minim folosind algoritmi clasici[21]). Notăm prin $solve(W)$ un asemenea algoritm. Fie soluția $M = \{(x, j) | x \in U$ și $j = 1, ..., k\}$. Următorul pas este aflarea unei soluții M' diferite de M.

Propoziție 3.7 *Doua cuplaje M și M' sunt diferite dacă există cel puțin o pereche (x, y) care se află în M și nu se află în M'.*

Astfel, propunem următorul algoritm, adaptat din [15], prin care căutăm o a doua soluție M' fixând câte o muchie candidat (x, y) prin care M' sa difere de M. Setând costul muchiei (x, y) pe o valoare infinită, avem garanția ca aceasta nu va fi luată în considerare în construcția lui M'.

Algoritm 1 anotherSolution

Input: W, M
Output: M'

1: $s \leftarrow \sum_{(u,v)\in M} w_{uv}$
2: **for all** $(x, y) \in M$ **do**
3: $\quad temp \leftarrow w_{xy}$
4: $\quad w_{xy} \leftarrow \infty$
5: $\quad M' \leftarrow solve(W)$
6: $\quad$ **if** $M' \neq \emptyset$ si $\sum_{(u,v)\in M'} w_{uv} = s$ **then**
7: $\quad\quad$ **return** M'
8: $\quad$ **else**
9: $\quad\quad w_{xy} \leftarrow temp$
10: $\quad$ **end if**
11: **end for**
12: **return** $\emptyset$

Algoritmul returnează fie mulțimea vidă, fie o soluție M' astfel încât exista o pereche $(x, y) \in M \setminus M'$. Se poate împărți problema inițială în două subprobleme disjuncte P_1 și P_2:

- P_1: mulțimea tuturor cuplajelor ce conțin muchia (x, y)
  În acest caz forțăm păstrarea perechii în soluție prin setarea tuturor celorlalte valori de pe linia x, coloana y pe o valoare infinita în matricea W: $w_{iy} = w_{xj}, \forall i = 1, .., n, i \neq x$ și $j = 1, ..., n, j \neq y$
- P_2: mulțimea tuturor cuplajelor ce **nu** conțin muchia (x, y)
  În acest caz perechea (x, y) nu va mai fi niciodată aleasă într-o soluție dacă costul acesteia este infinit: $w_{xy} = \infty$

Evident, există deja cate o soluție calculată pentru cele 2 subprobleme și

anume $M \in P_1$ și $M' \in P_2$. Prim urmare, se poate aplica Algoritmul 1 în mod recursiv pentru fiecare dintre aceste subprobleme pentru a determina întreaga mulțime de soluții. Această abordare conduce la construirea unei structuri de căutare arborescente în care rădăcina reprezintă problema inițială 3.6, iar fiecare nod intern constituie o împărțire pe subprobleme după o pereche (x, y). Soluția finală se construiește traversând arborele în adâncime și păstrând toate soluțiile parțiale calculate la fiecare pas. Nu se va genera aceeași soluție de mai multe ori prin faptul că problemele P_1 și P_2 sunt complet disjuncte.

Algoritm 2 dfsAgregare

Input: S, M, W

1: $s \leftarrow \sum_{(u,v)\in M} w_{uv}$
2: $M' \leftarrow another_solution(W, M)$
3: **if** $M' \neq \emptyset$ **then**
4: **return**
5: **else**
6: $S \leftarrow M'$
7: $(x, y) \in M \setminus M'$
8: $w_{xy} \leftarrow \infty$
9: $dfsAgregare(S, M', W)$
10: $w_{iy} \leftarrow w_{xj}, \forall i = 1, .., n, i \neq x$ si $j = 1, ..., n, j \neq y$
11: $dfsAgregare(S, M' \setminus (x, y), W)$
12: **end if**

Algoritm 3 Calculează toate cuplajele perfecte de cost minim

Input: W
Output: S

1: $S \leftarrow \emptyset$
2: $M \leftarrow solve(W)$
3: $S \leftarrow S \cup M$
4: $dfsAgregare(S, M, W)$
5: **return** S

Algoritmul 3 determină toate cuplajele perfecte de cost minim pornind de la o matrice de costuri W. Calculând matricea W după formula (3.4), atunci mulțimea S determinată de algoritm este chiar soluția căutată în Problema 3.6. Câteva aspecte legate de complexitatea metodei prezentate:
Fie $x = |S|$, numărul total de soluții pentru o anumita problema.

- Pentru fiecare soluție nou calculată, se construiesc două noi alte probleme. În total se vor rezolva maxim $2 * x + 1$ subprobleme.

- O subproblemă necesită găsirea unui cuplaj de cost minim ce se poate calcula într-un timp polinomial folosind metoda Ungară[13] ori un algoritm clasic de determinare a fluxului maxim de cost minim într-un graf bipartit[21].

Intuitiv, putem afirma ca Algoritmul 3 rulează într-un timp polinomial. Complexitatea acestuia a fost demonstrată în [15] ca fiind $\mathcal{O}((2x+1)k^3 n \log(n+k))$.

Subproblemele sunt complet independente de complementarele lor. Prin urmare, rezolvările acestora se pot rula în paralel pe mai multe fire de execuție. În experimentele rulate am ales să pornim un nou fir de execuție pentru fiecare subproblemă de tipul P_2 până la un anumit nivel din arborele de căutare determinat în funcție de procesorul folosit. Subproblema de tipul P_1 a rămas să ruleze pe firul de execuție principal.

3.2 Determinarea tuturor agregărilor producțiilor de cuvinte

În capitolul precedent am prezentat o metoda de a combina o limbă romanică modernă și limba latină pentru a automatiza procesul de determinare a etimonului latinesc. Metoda returna primele n cuvinte posibile ordonate de la cel cu probabilitatea cea mai mare la cel cu probabilitatea cea mai mică. Vom considera aceste liste de cuvinte ca fiind clasamente. Pentru fiecare cuvânt latinesc vom agrega clasamentele produse din fiecare limbă romanică modernă *(ro, it, fr, es, pt)*. Se observă faptul că pot exista mai multe astfel de agregări așa că ne propunem să le aflăm pe toate într-un mod eficient din punct de vedere al complexității timp. Alegem să luăm în considerare doar primele 5 cele mai bune cuvinte din fiecare set. Astfel, pentru un singur cuvânt latinesc, vom avea:

$$R = \{r_1, r_2, ..., r_k\}, \text{clasamentele produse din ro, it, fr, es, pt}$$
$$k = |R|, 1 \leqslant k \leqslant 5$$
$$U = \bigcup_{i=1}^{k} r_i \text{ universul de cuvinte}$$
$$n = |U|$$

Definim o matrice bidimensională de k linii și n coloane în care calculăm ordinea fiecărui cuvânt din universul unui cuvânt în fiecare clasament dat:

$$ord[i][j] = \begin{cases} |6 - r_i(x_j)| & , x_j \in r_i \\ 0 & , x_j \in U \setminus r_i \end{cases}$$

Calculăm apoi matricea de costuri W^5 conform formulei de la 3.4 pentru a afla cum este afectat costul final dacă un anumit cuvânt $x_i \in U$ este plasat pe

poziția j. Aceasta poziție poate sa fie mai mare decât 5 dar conform formulei construite, nu se va face nicio distincție între a plasa un cuvânt pe poziția 6 spre exemplu, sau poziția 7, considerându-se că respectivul element nu va aparține în agregarea finala de lungime 5.

Este limpede ca se poate aplica acum Algoritmul 3 prezentat în secțiunea precedentă pentru a afla mulțimea $S = \{\sigma_1, \sigma_2, ..., \sigma_t\}$ de agregări posibile. Propunem combinarea soluțiilor din S printr-un sistem de vot. Fiecare cuvânt va primi un scor bazat pe pozițiile pe care se află în clasamentele din S.

$$scor\colon U \to \mathbb{N}$$
$$scor(x_i) = \sum_{j=1}^{|S|} \sigma_j(x_i), \forall x_i \in U$$

Putem în sfârșit să construim lista finală, introdusă vag la începutul capitolului, de cuvinte ordonate în funcție de probabilitatea de a fi etimonul latinesc al unor anumite cuvinte moderne din *ro, it, fr, es, pt*. Pe prima poziție se va afla cuvântul produs cu scorul cel mai mic, adică cel ce se află pe primele poziții în clasamentele S, pe a doua poziție, cel cu următorul cel mai mic scor, și asa mai departe. În cazul în care există mai multe cuvinte cu același scor, le vom păstra pe toate pe aceeași poziție și le vom filtra manual folosind reguli lingvistice.

$$F = (x_1 > x_2 > x_3 > x_4 > x_5)$$
$$x_1 = \min(scor(x))$$
$$x_2 = \min_{x, x \neq x_1} (scor(x))$$
$$x_3 = \min_{x, x \neq x_1, x_2} (scor(x))$$
$$x_4 = \min_{x, x \neq x_1, x_2, x_3} (scor(x))$$
$$x_5 = \min_{x, x \neq x_1, x_2, x_3, x_4} (scor(x))$$
$$\{x_1, x_2, x_3, x_4, x_5\} \subseteq U$$

Capitolul 4

Experimente și rezultate

În capitolul 2 am prezentat pașii metodei propuse pentru reconstrucția cuvintelor latinești. Pe scurt, vom antrena mai multe sisteme CRF pe un set de date pentru a face mai apoi predicții de cuvinte. Din date vor fi extrase anumite atribute pe baza unor alinieri, conform secțiunii 2.3.2. Predicțiile sunt apoi agregate pe baza distantei rank ca în capitolul 3.

În experimentul rulat ne propunem să reconstruim cuvinte românești neatestate pornind de la un posibil etimon latinesc.

4.1 Antrenare

Pentru antrenarea sistemelor CRF am folosit setul de date propus de Ciobanu și Dinu [3]. Acesta conține 3218 grupuri complete de cuvinte cognate pentru cinci limbi romanice (română, italiană, franceză, spaniolă, portugheză) și etimoanele lor latinești. Împărțim acest set de date în trei părți, pentru antrenare, pentru reglarea parametrilor și pentru testare. Dimensiunea proporțiilor este $3 : 1 : 1$.

Aliniem fiecare pereche de cuvinte cognate din care extragem 3-grame ca și atribute pentru sistemele CRF. Folosim implementarea dată de Mallet toolkit [17]. Parametrii sunt găsiți printr-o căutare de tip *grid search* cu numărul de iterații $\{1, 5, 10, 25, 50, 100\}$ și pentru dimensiunea ferestrei w de $\{1, 2, 3, 4, 5\}$.

Pentru cei mai buni parametri (50 de iterații) obținem acuratețea top-10:

- Spaniolă 61%
- Franceză 62%
- Italiană 62%
- Portugheză 51%

- Latină 65%

4.2 Testare

Aplicam modelele învățate pe o lista de 235 de cuvinte cognate ce exista în limbile romanice din Occident dar nu și în limba română. Acestea au fost extrase dintr-o listă propusă de Rîpeanu în [22]. Agregăm apoi rezultatele folosind metoda din capitolul 3. Procesăm producțiile conform următoarelor două reguli:

1. diftongul *iă* nu există în limba română și îl înlocuim cu *ie*
2. consoanele duble dispar (spre exemplu *ll* devine *l*)

4.3 Evaluare

4.3.1 Producerea de cuvinte cognate

Evaluarea a fost făcută manual întrucât natura problemei face dificilă găsirea unei metode de evaluare automată. Algoritmul prezentat produce cuvinte românești neatestate deci doar un expert în domeniu le-ar putea valida. Am decis să folosim doar italiana și latina pentru a reconstrui cuvinte românești deoarece spaniola, franceza, portugheza sunt mai departe de română din punct de vedere ortografic, iar metoda prezentată se bazează pe analiza schimbărilor grafice.

Rezultatele obținute prin evaluarea manuală a producțiilor de cuvinte românești pornind de la limba latină și italiană:

Tipul cognat romanesc	Latină	Italiană
Reali	82 (34.8%)	72 (30.6%)
Reali cu intervenție lingvistică	12 (5.1%)	11 (4.6%)
Virtuali	69 (29.3%)	32 (13.6%)
Virtuali cu intervenție lingvistică	28 (11.9%)	11 (5.1%)
Inexistenți	51 (21.7%)	111 (47.2%)

- cognat **real**: cuvânt ce există în limba română datorită unor procese de dezvoltare interne ale limbii sau prin împrumutarea masivă a acestora din alte limbi
- cognat **real cu intervenție lingvistică**: cuvânt ce există în limba română în urma apariției unor noi criterii lingvistice
- cognat **virtual**: cuvânt ce ar fi putut exista în limba română dar care nu a fost atestat

- cognat **virtual cu intervenție lingvistică**: cuvânt ce ar fi putut exista în limba română în urma introducerii unor noi criterii lingvistice dar care nu a fost atestat
- cognat **inexistent**: cele ce nu aparțin niciunei categorii de mai sus

4.3.2 Reconstruirea cuvintelor latinești neatestate

În acest caz, setul de date conține cuvântul latinesc propus dar neatestat. Este vorba de un subset al setului de date propus de Rîpeanu în [22]. Deci am putut rula o evaluare automată a rezultatelor. În urma agregării cu distanța rank am obținut o acuratețe de 23% în top-10. Adică am verificat ca etimonul latinesc să se afle în primele zece producții.

Dar, dacă nu folosim agregarea ci doar limba italiană, se obține o acuratețe top-10 de 38%. Astfel, metoda confirmă mai mult de un sfert din cuvintele neatestate latinești, reconstruite artificial.

Capitolul 5

Concluzii

Această lucrare prezintă o metodă computațională aplicată cu succes în reconstrucția cuvintelor cognate în seturi de date incomplete și în producerea etimonului latinesc în cazul limbii române. Pentru acestea se folosesc cinci limbi romanice moderne: spaniola, italiana, franceza, portugheza și româna. Algoritmul are la bază idei preluate din lingvistica comparativă încercând să automatizeze procesul de detecție al șabloanelor de modificare ortografică în evoluția cuvintelor. Aceasta nu ia în considerare alți factori precum cei sociali, economici, tehnologici etc, factori ce ar fi dificil de modelat matematic.

Astfel, producțiile obținute vin în ajutorul experților în domeniul lingvisticii istorice pentru studiul evoluției și reconstrucției artificiale de limbaj.

5.1 Îmbunătățiri

În primul rând, o idee ce merită a fi cercetată este adăugarea dimensiunii fonetice în analiza cuvintelor cognate din limbile romanice moderne și latina. Așadar, sistemul va fi adaptat pentru a combina deducțiile bazate pe schimbările ortografice cu cele bazate pe transcrierea fonetică.

În al doilea rând, am putea extinde algoritmul pentru a lua în considerare toate limbile romanice moderne precum *catalana*, *siciliana* și dialectele lor. Cel mai probabil, ar trebui grupate în funcție de similaritate pentru a obține cele mai bune rezultate, cum am procedat și în experimentele rulate (italiană, română și latină). Spre exemplu, catalana este mai apropiată de spaniolă deci schimbările produse într-una dintre ele au șanse mai mari sa fie similare decât cele produse în română.

În al treilea rând, merită cercetată adaptabilitatea metodei prezentate în adăugarea dimensiunii semantice a cuvintelor. Putem oare adaugă sensul cuvintelor ca un alt set de atribute în învățarea sistemelor CRF?

Bibliografie

[1] A.Dinu and L.P. Dinu. On the syllabic similarities of romance languages. *LNCS, 3406*, pages 785–788, 2005.

[2] A.M. Ciobanu. Sequence Labeling for Cognate Production. *KES*, pages 1391–1399, 2016.

[3] A.M. Ciobanu and L.P. Dinu. Building a Dataset of Multilingual Cognates for the Romanian Lexicon. *LREC*, pages 1038–1043, 2014.

[4] A.M. Ciobanu and L.P. Dinu. Ab Initio: Automatic Latin Proto-word Reconstruction. *COLING*, pages 1604–1614, 2018.

[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*, chapter 24, pages 504–507. MIT Press and McGraw-Hills, 2 edition, 2001.

[6] L.P. Dinu. On the classification and aggregation of hierarchies with different constitutive elements. *Fund. Inform. 55 (1)*, pages 39–50, 2003.

[7] L.P. Dinu. Rank Distance with Applications in Similarity of Natural Languages. *Fundamenta Informaticae, 64(1-4)*, pages 135–149, 2005.

[8] L.P. Dinu and F. Manea. An efficient approach for the rank aggregation problem. *Theoretical Computer Science vol 359(1-3)*, pages 455–461, 2006.

[9] I. Fischer. *Latina Dunăreană. Introducere în istoria limbii române*. Editura Ştiinţifică şi Enciclopedică, 1985.

[10] S. Ganesh, S. Harsha, P. Pingali, and V. Verma. Statistical Transliteration for Cross Language Information Retrieval Using HMM Alignment Model and CRF. *CLIA*, pages 42–47, 2008.

[11] G. Jager and P. Sofroniev. Automatic cognate classification with a Support Vector Machine. *KONVENS*, 2016.

[12] G. Kondrak. A New Algorithm for the Alignment of Phonetic Sequences. *NAACL*, pages 288–295, 2000.

[13] H.W. Kuhn. The Hungarian method for assignment problem. *Naval Res. Logist. Quarterly 2*, pages 83–97, 1955.

[14] J. Lafferty, A. McCallum, and F. Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *International Conf. on Machine Learning*, pages 282–289, 2001.

[15] F. Manea and C. Ploscaru. A generalization of the assignment problem, and its application to the rank aggregation problem. *Fundamenta Informaticae 81*, pages 459–471, 2007.

[16] S. Marcus. Linguistic structures and generative devices in molecular genetics. *Cahiers Ling. Theor. Appl.*, pages 77–101, 1974.

[17] A. K. McCallum. MALLET: A Machine Learning for Language Toolkit, 2002.

[18] S.B. Needleman and C.D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, pages 443–453, 1970.

[19] T. Rama. Siamese convolutional networks for cognate identification. *COLING*, pages 1018–1027, 2016.

[20] T. Rama, J-M. List, J.Wahle, and G. Jager. Are automatic methods for cognate detection good enough for phylogenetic reconstruction in historical linguistics? *NAACL*, pages 393–400, 2018.

[21] R.Jonker and T.Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing 38*, pages 5(1): 325–340, 1987.

[22] S.R. Rîpeanu. *Lingvistica romanică: Lexic, Morfologie, Fonetică*. Editura All., 2001.

[23] M. Sala. *De la Latină la Română*. Editura Univers Enciclopedic, 1998.

[24] A. Schleicher. *Compendium der vergleichenden Grammatik der indogermanischen Sprachen*. Weimar, 1861.

[25] M. Weiss. *The Routledge Handbook of Historical Linguistics*. Routledge, 2015.