



UNIVERSITATEA DIN
BUCUREȘTI
— VIRTUTE ET SAPIENTIA

Aspecte ale lingvisticii compuționale 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 lingvistică 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 precădere pe limba română.

Cuprins

Cuprins	iii
1 Introducere	1
1.1 Reconstrucția limbajelor folosind metode comparative	2
1.2 Metode computazionale	2
1.3 Latina și limbi romanice moderne	3
1.4 Obiective și abordare	3
2 Reconstrucție de cuvinte latinești	5
2.1 Prezentare generală	5
2.2 Aliniere	6
2.2.1 Needleman-Wunsch	6
2.3 Campuri conditionate aleatoare	6
2.3.1 Generalități	6
2.3.2 Aplicate pe problema reconstrucției cuvintelor	6
3 Agregarea rezultatelor folosind distanța rank	7
3.1 Clasamente și distanța rank	7
3.1.1 Agregări cu distanța rank	8
3.1.2 Reducerea la o problemă de cuplaj perfect de cost minim	9
3.1.3 Calcularea tuturor agregărilor optime	10
3.2 Determinarea tuturor agregărilor producțiilor de cuvinte . . .	13
4 Rezultate	15
4.1 Evaluare autoamata	15
4.2 Evaluare manuală	15
5 Concluzii	17
Bibliografie	19

Capitolul 1

Introducere

Limbajul unei tari se afla in continua schimbare datorita mai multor cauze precum cauze sociale, economice, migratia populatiei, progresele in stiinta, tehnologie si medicina. De-a lungul timpului, modificarile s-au petrecut inevitabil si nu au fost neaparat reglementate de experti in domeniu. Astfel a aparut o noua ramura a lingvisticii ce studiaza sistematic aceste transformari incercand sa gaseasca sabloane, reguli si sa puna o ordine asupra schimbarilor lexicale, fonetice, semantice si sintactice. Cateva exemple clasice ar fi determinarea etimologiei unui cuvint (romanescul *genunchi* provine din latinescul *genuclum*) sau determinarea similaritatii limbajelor.

De cele mai multe ori, popoare ce vorbeau aceeaasi limba s-au despartit si au evoluat diferit, aparand limbaje derivate. Ne putem da seama de gradul de rudenie a acestora prin identificarea formelor **cognate** (grupuri de cuvinte ce au derivat din acelasi etimon). In alte cazuri, limbile au imprumutat cuvinte intre ele fie luandu-le ca atare (cuvantul japonez *sushi*), fie modificand formele (romanescul *cafea* din turcescul *kahve*). Astfel, se pot descoperi chiar relatii de natura istorica intre limbi si popoarele ce le folosesc. Spre exemplu, exista peste 200 de cuvinte ce se regasesc in toate limbile romanice moderne mai putin limba romana. Este greu de crezut ca o parte din aceste cuvinte nu au existat in lexicul limbii romane la un moment dat. Fischer[7] a identificat cateva cauze ce au condus la disparitia acestor cuvinte si inlocuirea acestora prin cuvinte cu alte etimologii: cauze externe, social-economice, schimbarea ocupatiilor romanilor, intreruperea Romaniei cu lumea Occidentala, dezvoltarea limbii romane departe de nucleul romanic si asa mai departe.

1.1 Reconstructia limbajelor folosind metode comparative

Lingvisticii istorici se ocupa cu cercetarea schimbarilor limbilor de-a lungul timpului. O preocupare majora a acestora o reprezinta producerea de cuvinte inrudite. Ei folosesc metode comparative prin care analizeaza modificarile limbajelor efectuand comparatii intre limbi inrudite pentru a deduce, printr-o inginerie inversa, proprietati ale limbii stramos comune.[20] Practic, se incearca determinarea grupurilor de cuvinte cognate si gasirea unor reguli prin care au fost obtinute din limba stramos. Acestea nu sunt neaparat atestate, cele mai multe dintre ele denumindu-se **proto-limbaje**. In consecinta, rezultatele obtinute sunt foarte greu de demonstrat pentru ca nu exista dovezi arheologice concrete.

Asemenea metode au dat rezolvate satisfacatoare, reusind sa determine structura familiilor de limbi europene. Mai mult, reconstructia proto-limbajul **Indo-European** considerat cel mai vechi limbaj cunoscut, a fost posibila prin punerea in corelatie cu limbajele derivate din acesta (proto-limbajul German, proto-limbajul Indo-Iranian).[19] In plus, metodele comparative au avut succes si in analiza altor familii de limbaje de pe alte continente.

1.2 Metode computationale

In principal, tehnicile comparative au mai multi pasi efectuati manual de lingvistici si presupun multe ore de munca. In perioada in care calculatoarele nu erau inca inventate, cautarea cuvintelor in carti, dictionare si prelucrarea acestora era o munca nu numai obositoare si repetitiva dar si dificila, fiind nevoie de atentie continua. Insa, odata cu aparitia metodelor computationale, se pune accentul pe determinarea automata a cuvintelor inrudite precum in [9] sau [15]. Aceste solutii sunt departe de a inlocui un expert in domeniul lingvisticii si isi propun mai mult sa vina in ajutorul acestuia pentru a facilita dezvoltarea in profunzime a domeniului.

Metodele computationale doar automatizeaza procesul pastrand ideea pornirii de la formele moderne ale cuvintelor pentru a le reconstrui pe cele vechi din care provin. Exista numeroase date intretinute activ de catre specialisti ce determina conexiuni intre cuvinte din mai multe limbi moderne. Spre exemplu, corpusul Europarl pentru limbile oficiale vorbite in Uniunea Europeana sau WordNet (Fellbaum, 1998) pentru limba engleza. Asemenea resurse vaste sunt esentiale in aplicarea unor tehnici comparative automate. Pe baza lor se pot determina si aplica reguli de productie a cuvintelor intr-un mod formal, fara prea mari interventii din partea lingvisticilor pentru asa zisele *exceptii*.

Este dificil de prezis cum anume a fost modificat un cuvânt pentru a ajunge

in forma lui actuala. Desi se presupune ca ar exista sabloane si reguli in evolutia unui cuvant din etimonul sau, sunt si exemple care s-au detasat semnificativ de stramosul lor: latinescul *umbilicu(lu)s* a ajuns la foma de *buric* in romana, *nombril* in Franceza si *umbigo* in Portugheza.

Detectia automata a cuvintelor cognate poate fi formulata ca o problema de clasificare prin invatare autoamata. O serie de attribute s-au dovedit a fi de succes de-a lungul timpului precum n -gramele, distante de editare, cel mai lung subsir comun etc. Jager si Sofroniev [8] determina perechi cognate folosind un clasificator SVM iar apoi probabilitati si distante pentru a grupa cuvinte in grupuri cognate. Rama [14] aplica o retea neurala convolutiionala. Ciobanu si Dinu [2] folosesc campuri aleatoare conditionate reusind sa reconstruiasca proto-cuvinte din seturi cognate incomplete.

1.3 Latina si limbi romanice moderne

Romana(ro), Italiana(it), Franceza(fr), Spaniola(es), Portugheza(pt) sunt doar cateva dintre limbile moderne ce au evoluat din latina, in special din dialectul vulgar (latina vorbita de oamenii de rand in Imperiul Roman). Bineinteles, provenind din aceeasi limba, se pot pune foarte multe probleme de natura lingvistica precum gradul de similaritate dintre acestea, gasirea grupurilor de cuvinte cognate, reconstructia de cuvinte latinești neatestatate si asa mai departe.

Tranzitia de la latina la o limba romanica moderna s-a efectuat prin schimbari majore de vocabular, sintaxa si fonologie. Aceste schimbari sunt mai mult sau mai putin similare pentru fiecare limba romanica moderna. In principal, s-a urmarit simplificarea vocabulurului prin eliminarea arhaismelor si a exceptiilor in favoarea regulilor clare, reducerea sinonimelor, stabilirea clara a sensurilor cuvintelor (conform [18]).

1.4 Obiective si abordare

Pentru limbile romanice *ro*, *it*, *fr*, *es*, *pt* exista un dictionar etimologic in [17] pornind de la limba latina. Acest set este incomplet si ne propunem sa-l completam folosind o metoda computationala bazata pe lingvistica comparativa. Evaluarea rezultatelor va fi facuta in mare parte manual, aceste cuvinte nefiind atestate. Metoda computationala are la baza ideea prezentata in [2].

Astfel, pornind de la mai multe perechi cognate din limbi romanice moderne, se va aplica o metoda de reconstructie a cuvintelor bazata pe etichetarea secventelor si campuri aleatoare conditionate. Fiecare limba moderna va fi analizata separat in raport cu limba latina. Apoi, rezultatele din toate limbile romanice vor fi combinate folosind agregari pe baza distantei rank [4]. Toate acestea vor fi detaliate pe larg in urmatoarele 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 strămos comun. Aplicăm o metodă similară cu cea folosită în [2]. Ca și în [2], ne bazăm pe faptul că modificările ortografice sunt strâns legate de evoluția cuvintelor. Deci încercăm 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 să le prelucram într-o manieră atât automată cât și manuală pentru a obține cele mai bune rezultate.

2.1 Prezentare generală

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: *Romana, Italiana, Franceza, Spaniola, Portugheza*, iar seturile de date vor avea forma (*cuvant modern, cuvant latinesc*). Din acestea, modelul va învăța pe baza campurilor conditionate 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 tehnică de agregare a acestor rezultate pentru a combina informație din toate limbile. În final, vom folosi mai apoi aceste schimbări pentru a oferi variante de cuvinte ce completează anumite seturi de date.

Pășii algoritmului pentru o anumită limbă romanică modernă sunt:

1. Pentru fiecare pereche (*cuvant modern, cuvant 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 alinierea fiecărei perechi.
3. Rulam sistemul CRF și obținem liste de n cele mai bune producții sortate în funcție de probabilitatea lor.

2.2 Aliniere

Avem perechi de tipul (*cuvant modern, cuvant 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[13] din bioinformatică, folosit cu succes și în probleme de procesare a 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. Asadar, problema inițială va fi împartită 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 două siruri $a = a_1a_2...a_n$ și $b = b_1b_2...b_m$ de caractere de lungime n respectiv m . Vrem să aliniem sirul b pentru a se potrivi cu sirul a .

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

1. **Potrivire**, caracterele alinate se potrivesc: $a_i = b_i$
2. **Nepotrivire**, caracterele alinate nu se potrivesc: $a_i \neq b_i$
3. **Spatiu**, caracterul din primul sir nu se aliniaza cu niciun caracter din al doilea sir sau invers;

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

2.3 Campuri conditionate aleatoare

2.3.1 Generalități

2.3.2 Aplicate pe problema reconstrucției cuvintelor

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 sortată cu mai probabile cuvinte latinești. Metrica folosită este distanța rank [4] întrucât s-au obținut rezultate bune în alte probleme de natură lingvistică precum determinarea similitudinii silabice a limbilor romanice [1], [5].

În primul rând vom defini ce înseamnă 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* introdusă

in articolul [4]. In intregul capitol vom folosi următoarele notatii:

- $U = \{1, 2, \dots, n\}$ o mulțime finită de obiecte numită univers
- $\tau = (x_1 > x_2 > \dots > x_d)$ 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:

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

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

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

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 să fie cu atât mai mare cu cât diferă mai mult începutul lor.[12] 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.[6]

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[4]:

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

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

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

Problemă 3.4 Fie U un set de obiecte și $T = \{\tau_1, \tau_2, \dots, \tau_m\}$ o mulțime de clasamente peste universul U . Vrem să determinăm mulțimea de agregari $A(T, k)$ 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 $T = \{\tau_1, \tau_2, \dots, \tau_m\}$ fixată indus de plasarea elementului $x_i \in U$ pe poziția j în σ [6]. 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 .

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

Propoziție 3.5 Distanța de la un clasament $\sigma = (\sigma_1 > \sigma_2 > \dots > \sigma_k)$ 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

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

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 , $W = (w_{i,j})_{1 \leq i,j \leq n}$ vrem să determinăm următoarea mulțime:

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

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, \dots, i_n) reprezintă o permutare a elementelor din U .

O soluție pentru a rezolva problema precedentă este aplicarea algoritmului Ungar prezentat de Khun [10]. 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 luăm doar k perechi). Conform [16] 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[3].

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 metodă de determinare a tuturor agregărilor bazată pe găsirea tuturor cuplajelor perfecte de cost minim dintr-un graf, metoda prezentată în [11]. 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* [11] 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 $c: E \rightarrow \mathbb{N}$ și $w: E \rightarrow \mathbb{N}$ reprezintă capacitatea unei muchii respectiv costul acesteia. Legăturile între Problema 3.6 și graful G sunt:

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

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 întreaga T . Se poate calcula ușor în acest graf un flux maxim de cost minim folosind algoritmi clasici[16]). Notăm prin $solve(W)$ un asemenea algoritm. Fie soluția $M = \{(x, j) | x \in U \text{ și } j = 1, \dots, k\}$. Următorul pas este aflarea unei soluții M' diferite de M .

Propoziție 3.7 *Două 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 [11], prin care căutăm o a doua soluție M' fixând câte o muchie candidat (x, y) prin care M' să difere de M . Setând costul muchiei (x, y) pe o valoare infinită, avem garanția că 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:    $temp \leftarrow w_{xy}$ 
4:    $w_{xy} \leftarrow \infty$ 
5:    $M' \leftarrow solve(W)$ 
6:   if  $M' \neq \emptyset$  și  $\sum_{(u,v) \in M'} w_{uv} = s$  then
7:     return  $M'$ 
8:   else
9:      $w_{xy} \leftarrow temp$ 
10:  end if
11: end for
12: return  $\emptyset$ 

```

Algoritmul returnează fie mulțimea vidă, fie o soluție M' astfel încât există 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 infinită în matricea W : $w_{iy} = w_{xj}, \forall i = 1, \dots, n, i \neq x \text{ și } j = 1, \dots, n, j \neq y$
- 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 câte o soluție calculată pentru cele 2 subprobleme și

3. AGREGAREA REZULTATELOR FOLOSIND DISTANȚA RANK

anume $M \in P_1$ si $M' \in P_2$. Prim urmare, se poate aplica Algoritmul 1 in mod recursiv pentru fiecare dintre aceste subprobleme pentru a determina intreaga multime de solutii. Aceasta abordare conduce la construirea unei structuri de cautare arborescente in care radacina reprezinta problema initiala 3.6, iar fiecare nod intern constituie o impartire pe subprobleme dupa o pereche (x, y) . Solutia finala se construiesc traversand arborele in adancime si pastrand toate solutiile parțiale calculate la fiecare pas. Nu se va genera aceeași soluție de mai multe ori prin faptul ca problemele P_1 si P_2 sunt complet disjuncte.

Algorithm 2 dfsAgregare

Input: S, M, W

```
1:  $s \leftarrow \sum_{(u,v) \in M} w_{uv}$ 
2:  $M' \leftarrow \text{another\_solution}(W, M)$ 
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:    $\text{dfsAgregare}(S, M', W)$ 
10:   $w_{iy} \leftarrow w_{xj}, \forall i = 1, \dots, n, i \neq x$  si  $j = 1, \dots, n, j \neq y$ 
11:   $\text{dfsAgregare}(S, M' \setminus (x, y), W)$ 
12: end if
```

Algorithm 3 Calculeaza toate cuplajele perfecte de cost minim

Input: W

Output: S

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

Algoritmul 3 determina toate cuplajele perfecte de cost minim pornind de la o matrice de costuri W . Calculand matricea W dupa formula (3.4), atunci multimea S determinata de algoritm este chiar solutia cautata in Problema 3.6. Cateva aspecte legate de complexitatea metodei prezentate:

Fie $x = |S|$, numarul total de solutii pentru o anumita problema.

- Pentru fiecare solutie nou calculata, se construiesc doua noi alte probleme. In total se vor rezolva maxim $2 * x + 1$ subprobleme.

- O subproblema necesita gasirea unui cuplaj de cost minim ce se poate calcula intr-un timp polinomial folosind metoda Ungara[10] ori un algoritm clasic de determinare a fluxului maxim de cost minim intr-un graf bipartit[16].

Intuitiv, putem afirma ca Algoritmul 3 ruelaza intr-un timp polinomial. Complexitatea acestuia a fost demonstrata in [11] ca fiind $\mathcal{O}((2x+1)k^3n \log(n+k))$.

Subproblemele sunt complet independente de complementarele lor. Prin urmare, rezolvarile acestora se pot rula in paralel pe mai multe fire de executie. In experimentele rulate am ales sa pornim un nou fir de executie pentru fiecare subproblema de tipul P_2 pana la un anumit nivel din arborele de cautare determinat in functie de procesorul folosit. Subproblema de tipul P_1 a ramas sa ruleze pe firul de executie principal.

3.2 Determinarea tuturor agregărilor producțiilor de cuvinte

În capitolul precedent am prezentat o metoda de a combina o limbă romanică modernă si 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:

$$R = \{r_1, r_2, \dots, r_k\}, \text{ clasamentele produse din ro, it, fr, es, pt}$$

$$k = |R|, 1 \leq k \leq 5$$

$$U = \bigcup_{i=1}^k r_i \text{ universul de cuvinte}$$

$$n = |U|$$

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

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

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

pe poziția j . Aceasta poziție poate să 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 finală de lungime 5.

Este limpede că se poate aplica acum Algoritmul 3 prezentat în secțiunea precedentă pentru a afla mulțimea $S = \{\sigma_1, \sigma_2, \dots, \sigma_t\}$ 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 .

$$\begin{aligned} scor: U &\rightarrow \mathbb{N} \\ scor(x_i) &= \sum_{j=1}^{|S|} \sigma_j(x_i), \forall x_i \in U \end{aligned}$$

Putem în sfârșit să construim lista finală, introdusă vag la începutul capitoului, 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 așa 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.

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

Capitolul 4

Rezultate

4.1 Evaluare autoamata

4.2 Evaluare manuala

Capitolul 5

Concluzii

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 and L.P. Dinu. Ab Initio: Automatic Latin Proto-word Reconstruction. *COLING*, pages 1604–1614, 2018.
- [3] 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.
- [4] L.P. Dinu. On the classification and aggregation of hierarchies with different constitutive elements. *Fund. Inform.* 55 (1), pages 39–50, 2003.
- [5] L.P. Dinu. Rank Distance with Applications in Similarity of Natural Languages. *Fundamenta Informaticae*, 64(1-4), pages 135–149, 2005.
- [6] 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.
- [7] I. Fischer. *Latina Dunăreană. Introducere în istoria limbii române*. Editura Științifică și Enciclopedică, 1985.
- [8] G. Jager and P. Sofroniev. Automatic cognate classification with a Support Vector Machine. *KONVENS*, 2016.
- [9] G. Kondrak. A New Algorithm for the Alignment of Phonetic Sequences. *NAACL*, pages 288–295, 2000.
- [10] H.W. Kuhn. The Hungarian method for assignment problem. *Naval Res. Logist. Quarterly* 2, pages 83–97, 1955.

- [11] 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.
- [12] S. Marcus. Linguistic structures and generative devices in molecular genetics. *Cahiers Ling. Theor. Appl.*, pages 77–101, 1974.
- [13] 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.
- [14] T. Rama. Siamese convolutional networks for cognate identification. *COLING*, pages 1018–1027, 2016.
- [15] 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.
- [16] 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.
- [17] S.R. Rîpeanu. *Lingvistica romanică: Lexic, Morfologie, Fonetică*. Editura All., 2001.
- [18] M. Sala. *De la Latină la Română*. Editura Univers Enciclopedic, 1998.
- [19] A. Schleicher. *Compendium der vergleichenden Grammatik der indogermanischen Sprachen*. Weimar, 1861.
- [20] M. Weiss. *The Routledge Handbook of Historical Linguistics*. Routledge, 2015.