UNIVERSITATEA DIN BUCUREȘTI
VIRTUTE ET SAPIENTIA

Aspecte computaționale în producerea de cuvinte

Lucrare de Licență

Maria-Smaranda Pandele

22 Iunie, 2019

Coordonator: Prof. Dr. Liviu Dinu

Facultatea de Matematică și Informatică, UNIBUC

Rezumat

This example thesis briefly shows the main features of our thesis style, and how to use it for your purposes.

Cuprins

Capitolul 1

Introducere

Capitolul 2

Reconstrucție de cuvinte

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 [3] întrucat s-au obtinut rezultate bune in alte probleme de natura lingvistica precum determinarea similitudinii silabice a limbilor romanice [1], [4].

In primul rand vom defini ce inseamna o lista ordonata de elemente. In al doilea rand, vom explica distanta rank intre doua clasamente si intre un clasament si o multime de clasamente. Apoi vom prezenta o metoda de aflare a tuturor agregarilor unei multimi de mai multe clasamente folosind distanta rank. In final, vom prelucra multimea de agregari bazat pe un sistem de vot pentru a determina **o singura lista ordonata** de posibile etimoane latinesti.

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 sa definim mai întâi ce înseamnă distanța dintre două clasemente 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 [3]. In intregul capitol vom folosi următoarele notatii:

- $U = \{1, 2, ..., n\}$ 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 si 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.[8] În al doilea rând, definiția funcției *ord* pune accentul pe lungimea clasamentelor întrucăt 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.[5]

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. Raportandu-ne la distanța rank avem[3]:

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 multime 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 si* $T = \{\tau_1, \tau_2, ..., \tau_m\}$ *o multime de clasamente peste universul U. Vrem sa determinam multimea de agregari* $A(T,k)$ *pentru un k fixat.*

Contruim 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 [5]. 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:*

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

Problema de mai sus se aseamănă cu o problemă de cuplaj de dimensiune k de cost minim întrucăt 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 solutie pentru a rezolva problema precedenta este aplicarea algoritmul Ungar prezentat de Khun [6]. Altfel, putem considera matricea W ca fiind o matrice de costuri intr-un graf bipartit G pe care aplicam un algoritm clasic de gasire a cuplajului maxim de cost minim (din care luam doar k perechi). Conform [9] aceasta problema poate fi rezolvata in timp polinomial $\mathcal{O}(n^3)$ construind o retea de flux cu capacitati convenabile si prin gasirea unor drumuri de augmentare minime, din punct de vedere al costului, folosind algoritmul lui Dijkstra[2].

Toate aceste rezolvari determina o singura agregare dar nu si pe toate, adica multimea $A(T, k)$ din definitia 3.3. In continure prezentam o metoda de determinare a tuturor agregarilor bazata pe gasirea tuturor cuplajelor *perfecte* de cost minim dintr-un graf, metoda prezentata in [7]. Algoritmul ruleaza intr-un timp polinomial. Particularizam problemele si algoritmii din articolul *A generalization of the assignment problem, and its application to the rank aggregation problem* [7] pentru Problema 3.6.

3.1.3 Calcularea tuturor agregărilor optime

Reamintim faptul ca dorim sa calculam multimea de agregari $A(T, k)$, stiind costul plasarii fiecarui element pe fiecare pozitie, memorat in matricea W^k calculata la (3.4). Reformulam problema in elemente de teoria grafurilor. Astfel, asociem Problemei 3.6 un graf $G = (V, E, c, w)$, unde V reprezintă multimea de noduri, E este multimea de muchii iar $c\colon E \to \mathbb{N}$ si $w\colon E \to \mathbb{N}$ reprezinta capacitatea unei muchii respectiv costul acesteia. Legaturile intre Problema 3.6 si graful G sunt:

- $V = \{src, dst\} \cup U \cup \{1, 2, ..., k, k+1\}$
- $E = \{(src, x_i) | x_i \in U\} \cup \{(x_i, j) | x_i \in U$ si $j = 1, ..., k\} \cup \{(j, dst) | j = 1, ..., k\}$
- $c(muchie) = 1, \forall muchie = (x, j) \in E, j \neq k+1$
- $c(muchie) = \infty, \forall muchie = (x, k+1) \in E$
- functia 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 pozitia $k+1$ va semnifica excluderea acestuia din agregare ce afecteaza distanta rank dintre un clasament ce nu contine elementul x si multimea intreaga T. Se poate calcula usor in acest graf un flux maxim de cost minim folosind algoritmi clasici[9]). Notam prin $solve(W)$ un asemenea algoritm. Fie solutia $M = \{(x,j) | x \in U \text{ si } j = 1,...,k\}$. Urmatorul pas este aflarea unei solutii M' diferite de M.

Propoziție 3.7 *Doua cuplaje M si M′ sunt diferite daca exista cel putin o pereche* (x,y) *care se afla in M si nu se afla in M′.*

Astfel, propunem urmatorul algoritm, adaptat din [7], prin care cautam o a doua solutie M' fixand cate o muchie candidat (x,y) prin care M' sa difere de M. Setand costul muchiei (x,y) pe o valoare infinita, avem garantia ca aceasta nu va fi luata in considerare in constructia lui M'.

Algoritm 1 anotherSolution

Input: W, M
Output: M'

$s \leftarrow \sum_{(u,v)\in M} w_{uv}$
for all $(x,y) \in M$ **do**
  $temp \leftarrow w_{xy}$
  $w_{xy} \leftarrow \infty$
  $M' \leftarrow solve(W)$
  if $M' \neq \emptyset$ si $\sum_{(u,v)\in M'} w_{uv} = s$ **then**
    return M'
  else
    $w_{xy} \leftarrow temp$
  end if
end for
return $\emptyset$

Algoritmul returneaza fie multimea vida, fie o solutie M' astfel incat exista o pereche $(x,y) \in M \setminus M'$. Se poate imparti problema initiala in doua subprobleme disjuncte P_1 si P_2:

- P_1: multimea tuturor cuplajelor ce contin muchia (x,y)
  In acest caz fortam pastrarea perechii in solutie prin setarea tuturor celorlalte valori de pe linia x, coloana y pe o valoare infinita in matricea W: $w_{iy} = w_{xj}, \forall i = 1,..,n,\ i \neq x$ si $j = 1,...,n,\ j \neq y$
- P_2: multimea tuturor cuplajelor ce **nu** contin muchia (x,y)
  In acest caz perechea (x,y) nu va mai fi niciodata aleasa intr-o solutie daca costul acesteia este infinit: $w_{xy} = \infty$

Evident, exista deja cate o solutie calculata pentru cele 2 subprobleme si 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 construieste traversand arborele in adancime si pastrand toate solutile parțiale calculate la fiecare pas. Nu se va genera aceeasi solutie de mai multe ori prin faptul ca problemele P_1 si P_2 sunt complet disjuncte.

Algoritm 2 dfsAgregare

Input: S, M, W

$s \leftarrow \sum_{(u,v)\in M} w_{uv}$
$M' \leftarrow another_solution(W, M)$
if $M' \neq \emptyset$ **then**
  return
else
  $S \leftarrow M'$
  $(x, y) \in M \setminus M'$
  $w_{xy} \leftarrow \infty$
  $dfsAgregare(S, M', W)$
  $w_{iy} \leftarrow w_{xj}, \forall i = 1, .., n,\ i \neq x$ si $j = 1, ..., n,\ j \neq y$
  $dfsAgregare(S, M' \setminus (x, y), W)$
end if

Algoritm 3 Calculeaza toate cuplajele perfecte de cost minim

Input: W
Output: S

$S \leftarrow \emptyset$
$M \leftarrow solve(W)$
$S \leftarrow S \cup M$
$dfsAgregare(S, M, W)$
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[6] ori un algoritm clasic de determinare a fluxului maxim de cost minim intr-un graf bipartit[9].

Intuitiv, putem afirma ca Algoritmul 3 ruelaza intr-un timp polinomial. Complexitatea acestuia a fost demonstrata in [7] ca fiind $\mathcal{O}((2x+1)k^3 n \log(n+k))$.

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:

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

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

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

Calculam apoi matricea de costuri W^5 conform formulei de la 3.4 pentru a afla cum este afectat costul final daca un anumit cuvant $x_i \in U$ este plasat pe pozitia j. Aceasta poziitie poate sa fie mai mare decat 5 dar conform formulei construite, nu se va face nicio distinctie intre a plasa un cuvant pe pozitia 6 spre exemplu, sau pozitia 7, considerandu-se ca respectivul element nu va apartine in agregarea finala de lungime 5.

Este limpede ca se poate aplica acum Algoritmul 3 prezentat in sectiunea precedenta pentru a afla multimea $S = \{\sigma_1, \sigma_2, ..., \sigma_t\}$ de agregari posibile. Propunem combinarea solutiilor din S printr-un sistem de vot. Fiecare cuvant va primi un scor bazat pe pozitiile pe care se afla in 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 in sfarsit sa construim lista finala, introdusa vag la inceputul capitolului, de cuvinte ordonate in functie de probabiliatea de a fi etimonul latinesc al unor anumite cuvinte moderne din *ro, it, fr, es, pt*. Pe prima pozitie se va afla cuvantul produs cu scorul cel mai mic, adica cel ce se afla pe primele pozitii in clasamentele S, pe a doua pozitie, cel cu urmatorul cel mai mic scor, si asa mai departe. In cazul in care exista mai multe cuvinte cu acelasi scor, le vom pastra pe toate pe aceeasi pozitie si 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

Rezultate

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] 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.

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

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

[5] 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.

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

[7] 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.

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

[9] 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.