Concordia
algorytm przeszukiwania pamięci tłumaczeń

dr Rafał Jaworski
Uniwersytet im. Adama Mickiewicza w Poznaniu

Seminarium ZIL, IPI PAN, Warszawa

Motywacja badań

- Tłumacze powszechnie kolekcjonują wszystkie wykonane przez siebie tłumaczenia.
- Jeden tłumacz jest w stanie zgromadzić pamięć tłumaczeń wielkości rzędu setek tysięcy par zdań.
- Każde nowe tłumaczone zdanie jest najpierw poszukiwane w pamięci tłumaczeń.

Motywacja badań

- Tłumacząc zdanie:

  „the agreement was concluded on 11th of March 2012”

  możemy natrafić na następujące, już przetłumaczone zdania:

  a) *„the agreement was concluded on 25th of September 2014”*
  b) *„the contract was signed on 11th of March 2012”*
  c) *“the agreement was not concluded”*

- O podobieństwie zdań decydują metryki podobieństwa (tzw. *fuzzy search*).

Motywacja badań

- Pamięci tłumaczeń - większa efektywność pracy tłumacza: 10%-30% do 60%-70% (O'Brien, 1998), (Sommers, 2002).

- Większa powtarzalność tłumaczeń.

- Trudność w zgromadzeniu pamięci tłumaczeń zawierającej dostatecznie podobne zdania.

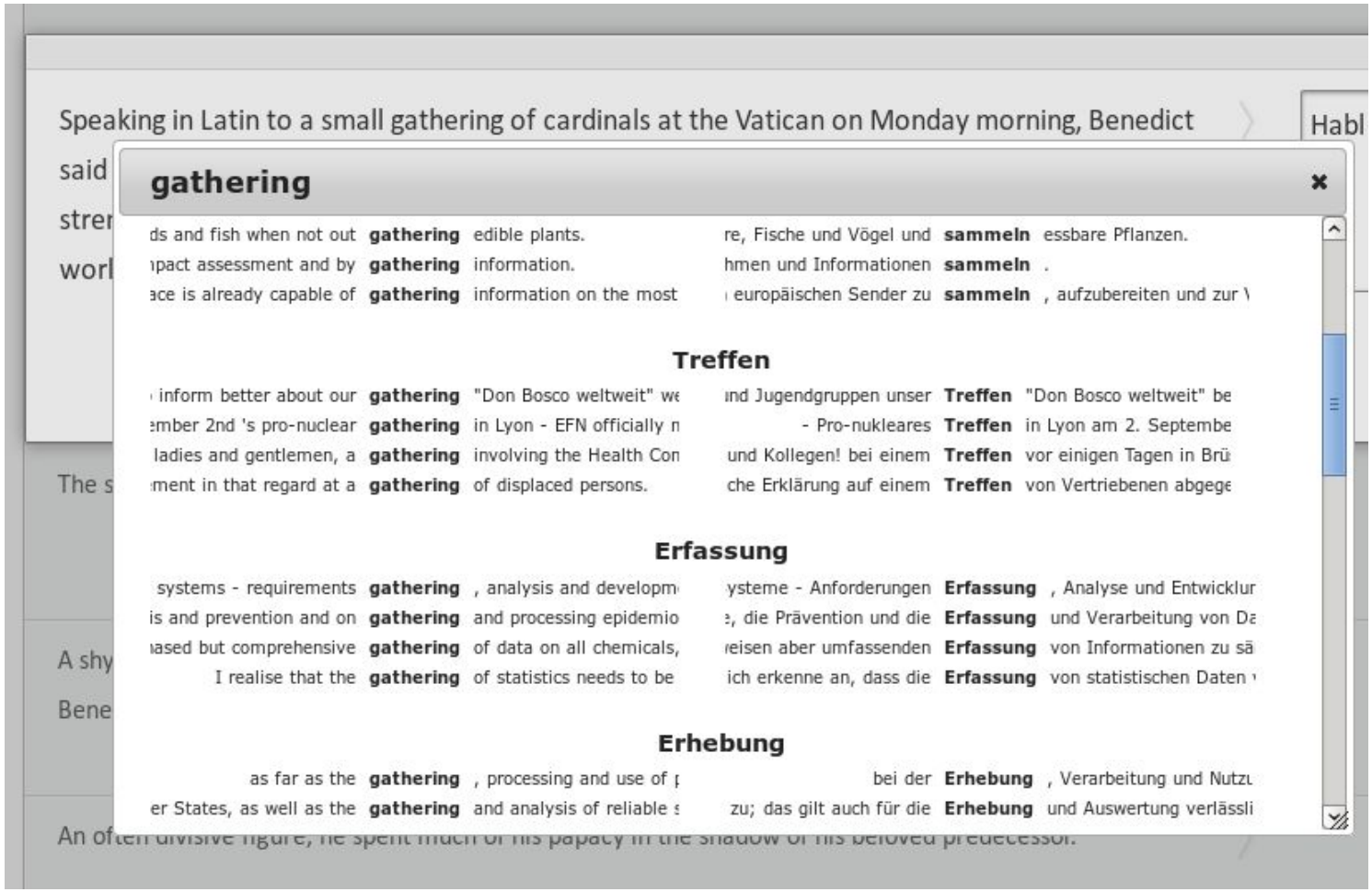
Najnowsze osiągnięcia w dziedzinie CAT

- Trados - lider rynku CAT-ów od wielu lat.
- Najnowsza wersja - SDL Trados Studio 2015
- Wybrane funkcjonalności:
  - pamięci tłumaczeń użytkownika, fuzzy search,
  - wyszukiwarka konkordancji,
  - AutoSuggest - podpowiadanie podczas pisania
  - PerfectMatch i ContextMatch - dopasowania “ponad 100%”
  - sprawdzanie terminologii (konieczne słowniki w formacie MultiTerm)
  - narzędzia do zarządzania projektem tłumaczenia, poprawiacze pisowni, poprawiacze znaków technicznych

Najnowsze osiągnięcia w dziedzinie CAT

- CASMACAT (Alabau et al., 2013)
- Oparty na wcześniejszym projekcie - MATECAT
- Kluczowe funkcjonalności:
  - adaptacyjne tłumaczenie automatyczne
  - sugestie tłumaczenia podczas pisania
  - dwujęzyczny concordancer

CASMACAT - bilingual concordancer



CASMACAT - bilingual concordancer

- Konieczność wpisywania słów/fraz do wyszukiwania.

- Brak automatycznego wyszukiwania fragmentów przydatnych tłumaczowi.

Concordia - cel badań

- Cel badań - opracowanie algorytmu wyszukiwania przybliżonego, który można zastosować w narzędziu ułatwiającym pracę tłumaczowi.

- Wynik dotychczasowych badań - Concordia - wydobywanie z pamięci tłumaczeń wszystkich fragmentów zdania wejściowego.

- Główne wymaganie niefunkcjonalne - szybkość działania.

Projekt Concordia

- Projekt OpenSource dostępny na platformie SourceForge: http://tmconcordia.sourceforge.net/
- Łączenie cech klasycznego przeszukiwania pamięci tłumaczeń z wyszukiwaniem konkordancji.
- Nazwa od rzymskiej bogini zgody.

Concordia – przykład

- Demonstracyjna wersja Concordii jest dostępna pod adresem:

http://concordia.vm.wmi.amu.edu.pl/jrc_plen/

Algorytm Concordia

- Algorytm wyszukiwania Concordia:

Kodowanie słów

Konstrukcja tablicy sufiksowej

Wyszukanie wszystkich fragmentów zdania wejściowego w indeksie

Znalezienie najlepszego pokrycia

Concordia - indeks

- Każde zdanie dodawane do indeksu jest reprezentowane jako ciąg liczb, będących kodami słów.
- Tekstem do przeszukiwania jest ciąg liczb, powstały jako konkatenacja kodów słów ze wszystkich zdań.
- Na jego podstawie konstruowana jest tablica sufiksowa.

Concordia - indeks

- Problem - wielkość alfabetu.
- Znakami są słowa, a ich może być kilkadziesiąt lub kilkaset tysięcy.
- Znany sposób indeksowania tekstów nad dużym alfabetem (Ferragina P. et al., 2004), wielkość indeksu:

$$nH(T) + O((n \log \log n)/ \log |\Sigma| n),$$

- Zbyt długi czas konstrukcji.

Concordia - indeks

- Rozwiązanie - algorytmy konstrukcji klasycznej tablicy sufiksowej (Burkhardt, 2003) oraz (Po, 2003).

- Autorski sposób rozwiązania problemu dużego alfabetu - rozdrobnienie.

Concordia - rozdrobnienie

- Rozdrobnienie polega na podziale szerokich znaków (kodów słów) na mniejsze.
- Niech s oznacza liczbę części, na które rozkładamy każdy znak.
- Po rozdrobnieniu otrzymujemy tekst długości $s \cdot n$, na którym uruchamiamy algorytmy konstruowania i przeszukiwania tablicy sufiksowej.

Concordia - rozdrobnienie

Przykład kodowanego tekstu:

30781 | 12912 | 124

gdzie:
30781 → komisja, 12912 → praw, 124 → człowieka

W zapisie szesnastkowym, dwa bajty na znak (=słowo):
78 3D | 32 70 | 00 7C

Dla *s* = 2 otrzymujemy tekst:
78 | 3D | 32 | 70 | 00 | 7C

Concordia - rozdrobnienie

- Na podstawie rozdrobnionego tekstu konstruowana jest tablica sufiksowa.
- Jest ona wprawdzie *s* razy większa, niż w wersji nierozdrobnionej, ale operuje na znacznie mniejszym alfabecie.
- Algorytm wyszukujący należy zmodyfikować tak, aby po znalezieniu podciągów sprawdzał, czy znaleziona pozycja jest podzielna przez *s*.

Concordia - indeks

- Na indeks składają się następujące struktury danych (wszystkie załadowane do pamięci RAM):
  - *hashedIndex* - konkatenacja kodów wszystkich dodanych zdań, rozdzielonych znakami końca zdania (tekst do przeszukiwania)
  - *suffixArray* - tablica sufiksowa
  - *markersArray* - wyjaśnione na następnym slajdzie

Concordia - *markersArray*

- Na przykład, niech indeks zawiera dwa zdania:
  - komisja (*kod=1*) praw (*2*) człowieka (*3*), *id zdania = 49*
  - łamanie (*4*) praw (*2*) imigrantów (*5*), *id zdania = 23*

	0	1	2	3	4	5	6	7
HI	1	2	3	EOS	4	2	5	EOS
MA	(49, 0)	(49, 1)	(49, 2)	(49, 3)	(23,0)	(23,1)	(23,2)	(23,3)

- Dzięki temu *simpleSearch* zwraca wynik “fraza *praw imigrantów* znaleziona w zdaniu *id=23* na pozycji *1*” a nie: “na pozycji *5* w całym tekście”

- Czas odszukania tej informacji: *O(1)*

Concordia - szybkość indeksu

Szybkość (1.70GHz CPU, 3GB RAM):

Liczba zdań (korpus JRC)	1 917 637
Liczba słów	20 062 518
Czas dodawania (*addSentence*)	7min 11s
Czas generowania (*generateIndex*)	7.9s
Wyszukiwanie 1	0.001s
Wyszukiwanie 2	<0.001s

Wyszukiwanie 1: fraza "*Parlamentu Europejskiego*", 9080 wystąpień w korpusie

Wyszukiwanie 2: fraza "*Dostęp do zatrudnienia*", 6 wystąpień w korpusie

concordiaSearch - serce algorytmu

- Centralny punkt algorytmu - procedura *concordiaSearch.*
- Inspiracja - klasyczne przeszukiwanie tablicy sufiksowej.
- Wejście - kwerenda w postaci ciągu słów.
- Wyjście - lista fragmentów zdań z indeksu, które pokrywają kwerendę (zwaną *pattern*).

concordiaSearch - serce algorytmu

Ogólny schemat algorytmu *concordiaSearch*:

```
fragments = empty_set
for i=0 to length(pattern)
   currentSuffix = pattern[i:]
   fragments.add(
      longestPrefixes(currentSuffix))
bestOverlay = selectBestOverlay(fragments)
return bestOverlay
```

concordiaSearch - longestPrefixes

- Funkcja *longestPrefixes(a)* służy do znalezienia fragmentów z indeksu, mających najdłuższy wspólny początek z ciągiem słów *a*.

- Efektywne wykonanie tej funkcji jest możliwe dzięki wykorzystaniu tablicy sufiksowej.

concordiaSearch - longestPrefixes

- Wykonanie funkcji *longestPrefixes* na ciągu słów:

w0	w1	w2	w3

tablica sufiksowa:

concordiaSearch - longestPrefixes

- Wykonanie funkcji *longestPrefixes* na ciągu słów:

w0	w1	w2	w3

tablica sufiksowa:

concordiaSearch - longestPrefixes

- Wykonanie funkcji *longestPrefixes* na ciągu słów:

w0	w1	w2	w3

tablica sufiksowa:

concordiaSearch - longestPrefixes

- Wykonanie funkcji *longestPrefixes* na ciągu słów:

w0	w1	w2	w3

tablica sufiksowa:

concordiaSearch - longestPrefixes

- Wykonanie funkcji *longestPrefixes* na ciągu słów:

w0	w1	w2	w3

tablica sufiksowa:

concordiaSearch - longestPrefixes

- Wykonanie funkcji *longestPrefixes* na ciągu słów:

w0	w1	w2	w3

tablica sufiksowa:

} wyniki wyszukiwania

concordiaSearch - best overlay

- Wynik - lista fragmentów z indeksu, które pokrywają *pattern*.
- Do wyznaczenia tzw. najlepsze pokrycie (*best overlay*) zdania wejściowego (*pattern*).
- Pokryciem zdania wejściowego nazywamy listę parami rozłącznych fragmentów zdań z indeksu, które zawierają się w *pattern*.

concordiaSearch - best overlay

Istnieją lepsze i gorsze pokrycia:

lepsze:

To	jest	zdanie	testowe	do	wyszukania

gorsze:

To	jest zdanie	testowe do		wyszukania

Dobre pokrycie składa się z małej liczby długich fragmentów.

concordiaSearch - best overlay

- Klasyczne sposoby obliczenia oceny pokrycia - indeks Jaccarda, WER

- Nie biorą one pod uwagę długości fragmentów wchodzących w skład pokrycia.

- Opracowano autorską modyfikację indeksu Jaccarda, w którym proporcjonalne pokrycie wzorca przez fragment jest modyfikowane czynnikiem z przedziału (0,1].

concordiaSearch - best overlay

Do oceny pokrycia stosuje się autorski wzór:

$$score = \sum_{f \in O} \frac{len(f)}{len(p)} \cdot \left(\frac{\log(len(f)+1)}{\log(len(p)+1)} \right)^{1/\alpha}$$

gdzie:

- *O* - pokrycie (*overlay*)
- *len* - liczba słów
- *f* - fragment
- *p* - *pattern*
- *α* - współczynnik wygładzenia

concordiaSearch - best overlay

- Dla α = 1, wzór ten oblicza ocenę pokrycia, która w 2 eksperymentach z udziałem tłumaczy wykazała korelację -0.73 oraz -0.50 z czasem tłumaczenia zdania (Jaworski et al., 2016)

- Parametr α można dostrajać w zależności od konkretnych zastosowań.

- Wyższe jego wartości - mniejsze kary za skracanie fragmentów.

concordiaSearch - best overlay

- Cel procedury *computeBestOverlay* - znalezienie wśród zbioru wyników wyszukiwania takiego pokrycia zdania wejściowego (*pattern*), które ma najwyższy *score*.

- Procedura rekurencyjnie przeszukuje przestrzeń możliwych pokryć i oblicza *score* każdego pokrycia.

concordiaSearch - best overlay

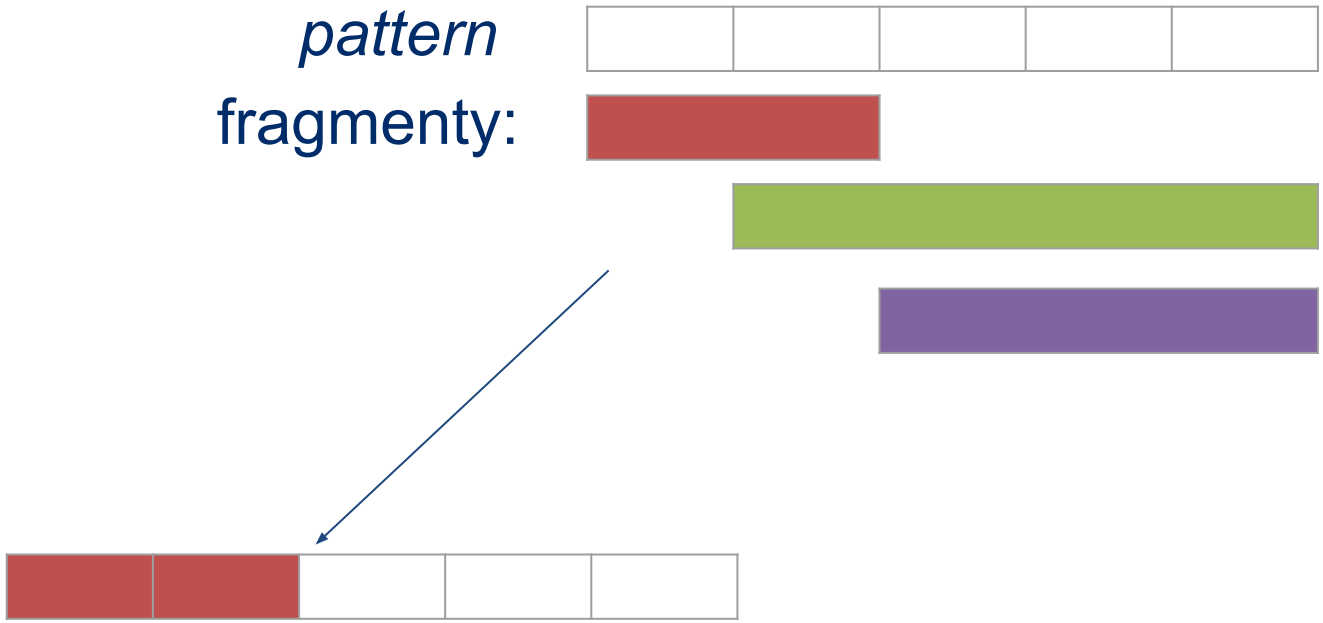
- Podobne rozwiązanie (Koehn et al. 2010)
- Wykonuje walidację znalezionych dopasowań celem obliczenia odległości Levenshteina pomiędzy zdaniem wyszukiwanym a odnalezionym w pamięci tłumaczeń.
- Łączenie mniejszych dopasowań w większe i znalezienie optymalnego dopasowania zdania wyszukanego do wzorca.
- Technika łączenia dopasowań opiera się na idei algorytmu A*.

concordiaSearch - best overlay

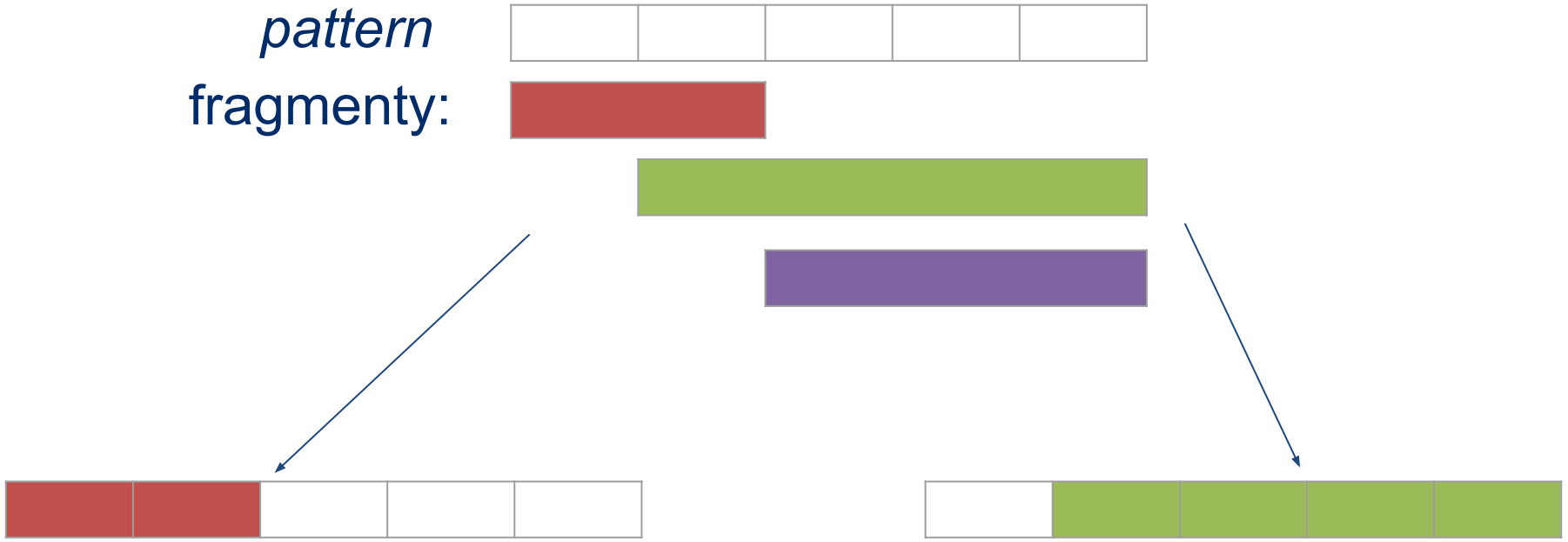
pattern

fragmenty:

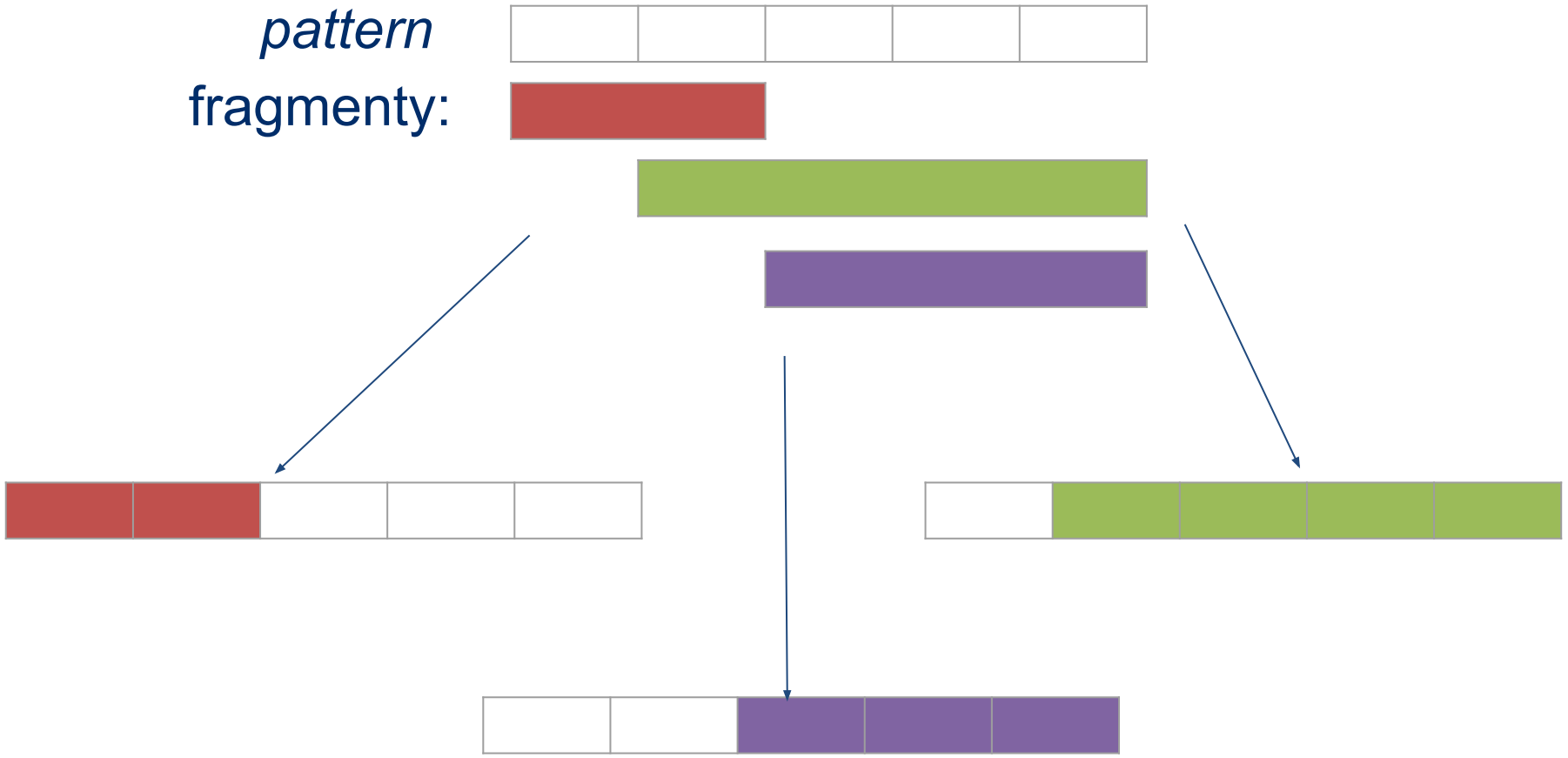
concordiaSearch - best overlay



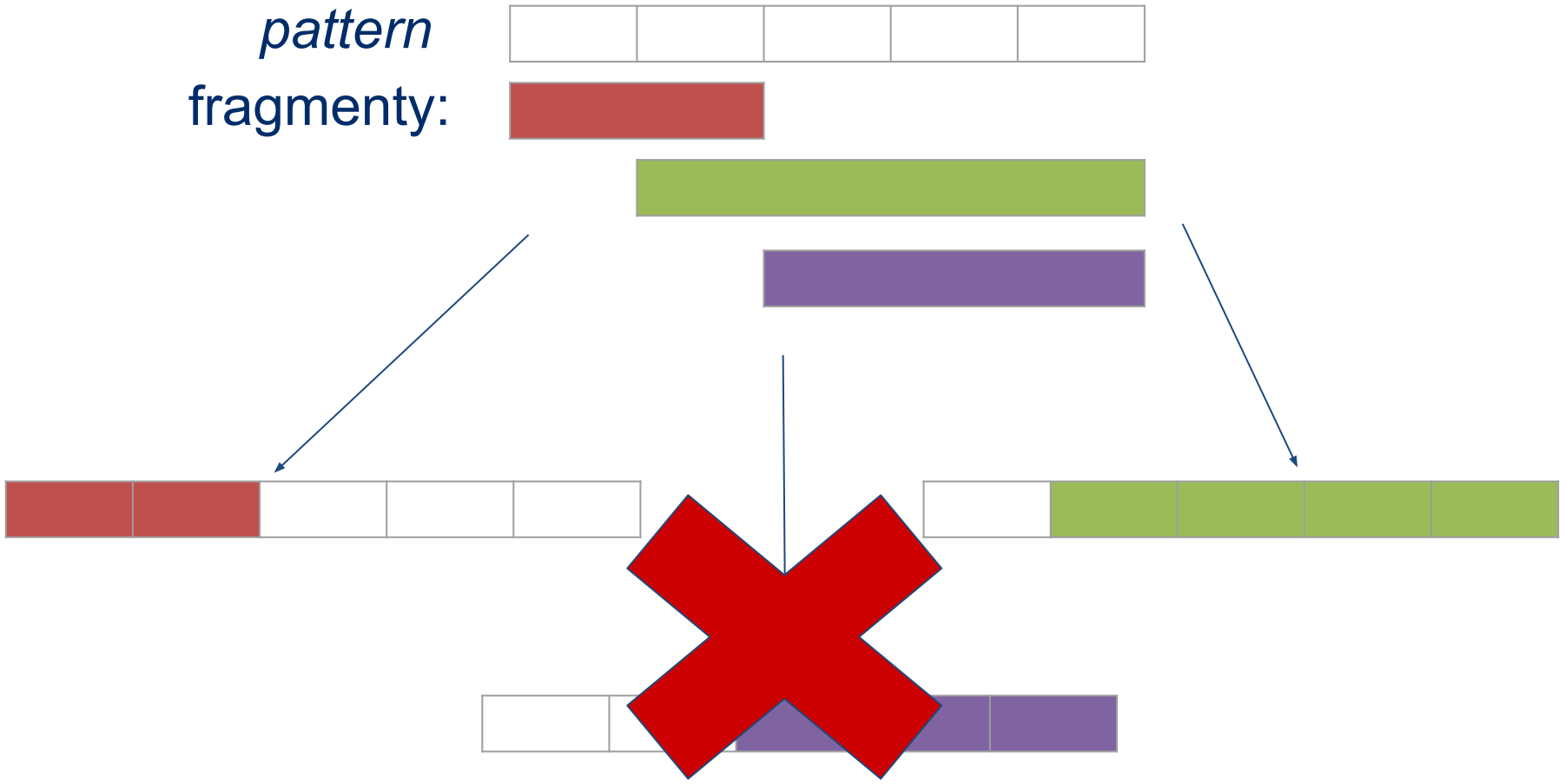
concordiaSearch - best overlay



concordiaSearch - best overlay

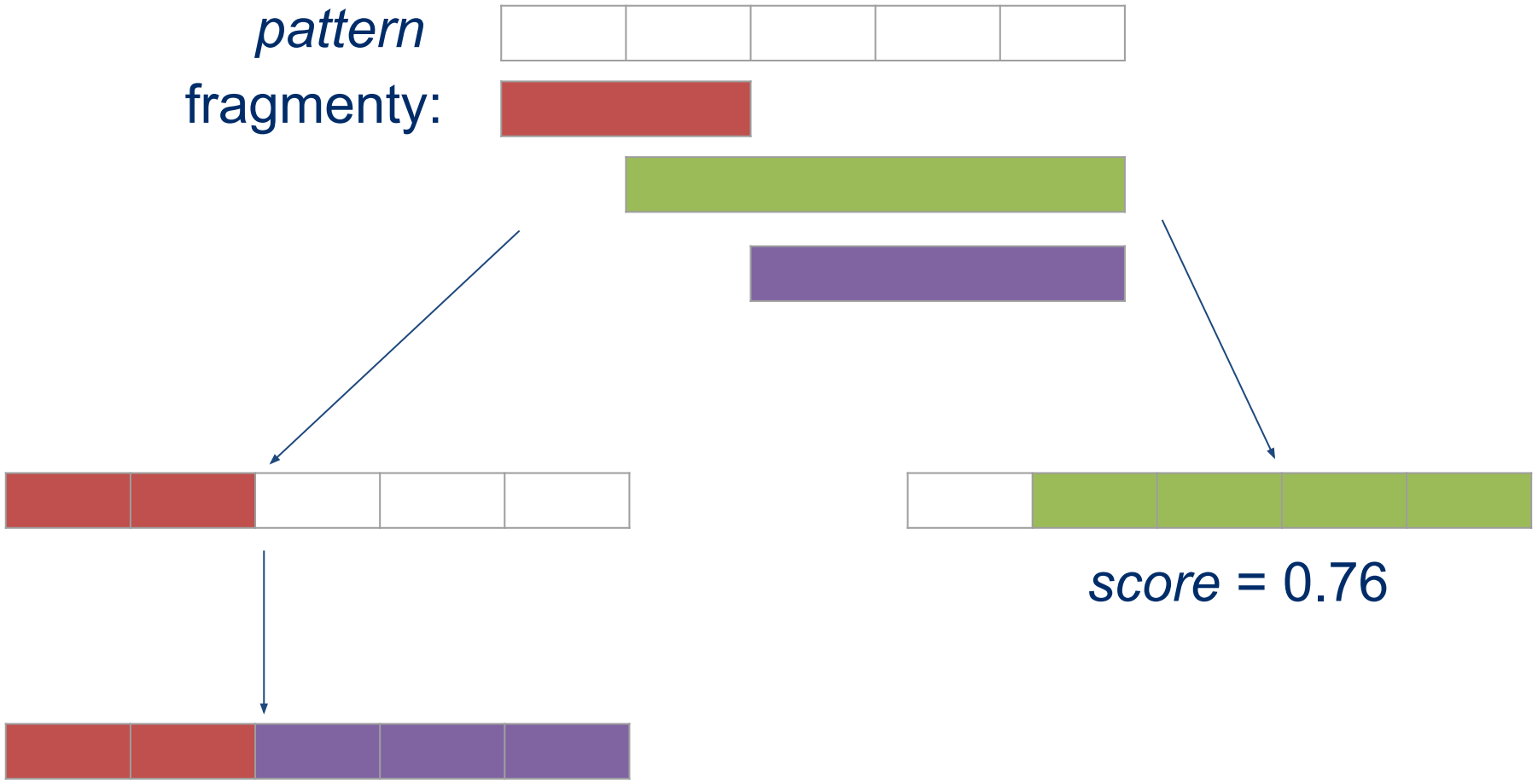


concordiaSearch - best overlay

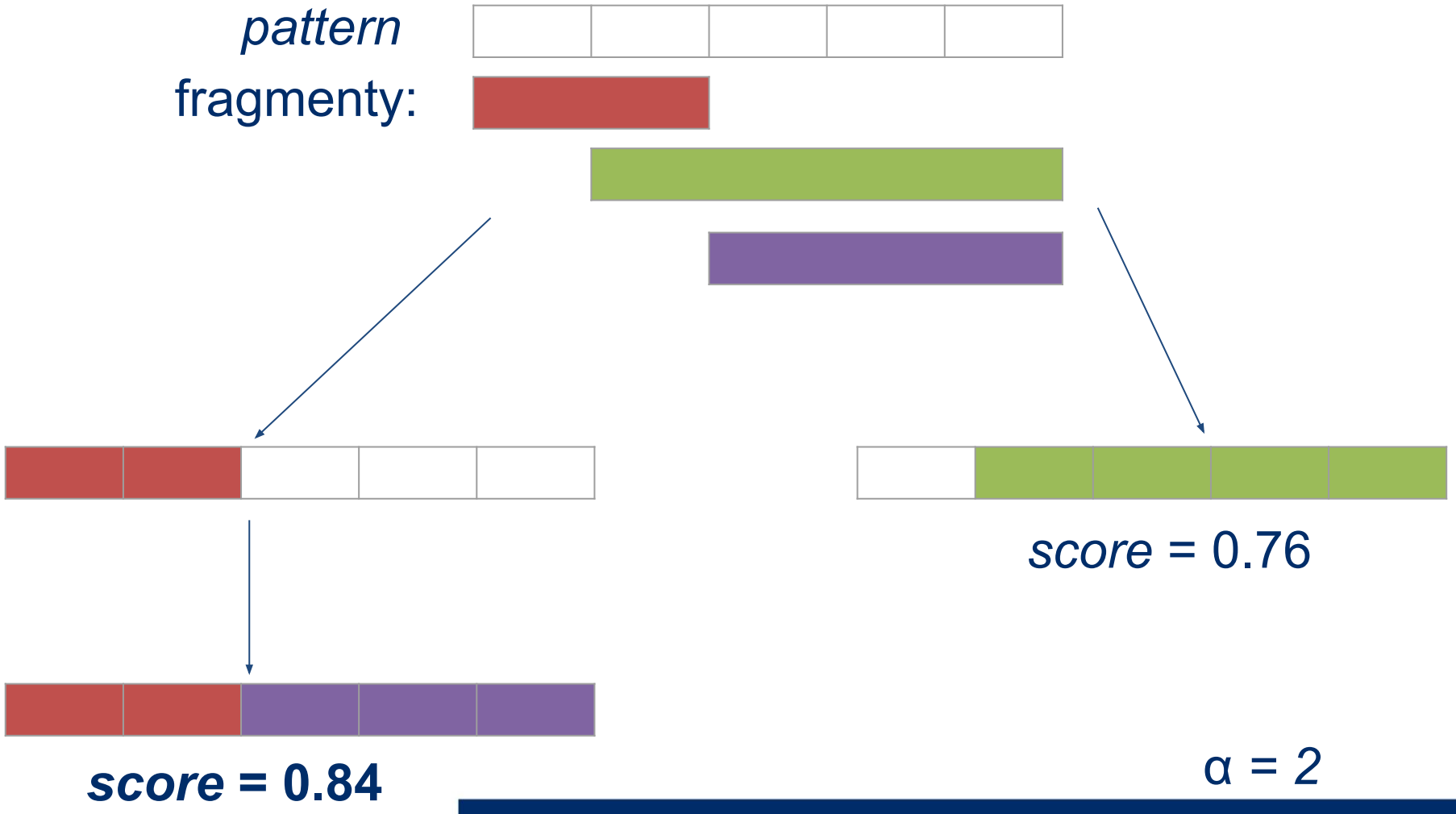


concordiaSearch - best overlay

concordiaSearch - best overlay



concordiaSearch - best overlay



Concordia-Server

- Concordia stanowi jedynie indeks do szybkiego wyszukiwania.
- W celu przechowywania całej pamięci tłumaczeń konieczny jest osobny system - Concordia-Server

https://github.com/rjawor/concordia-server

- Jest to aplikacja webowa działająca w trybie web service, z interfejsem JSON.
- W celu zapewnienia maksymalnej wydajności zastosowano:
  - NGINX
  - FastCGI++
  - PostgreSQL

Concordia-Server

- Concordia-Server dokonuje wyszukania par zdań z pamięci tłumaczeń, których fragmenty znalazły się w *best overlay* zdania wejściowego.

- Istnieje możliwość załadowania do systemu pamięci tłumaczeń zrównoleglonej na poziomie słów, aby uzyskać podświetlenie fragmentów również po stronie docelowej.

Podsumowanie

- Opracowano następujące efektywne elementy algorytmu wyszukiwania przybliżonego:
  - konstrukcja **rozdrobnionego** indeksu
  - tablica ***markersArray*** **-** przyspieszenie kolekcjonowania wyników
  - autorski algorytm ***concordiaSearch***
  - wzór na **ocenę pokrycia** wzorca fragmentami
  - autorski algorytm ***computeBestOverlay***

Podsumowanie

- Niska złożoność obliczeniowa - zastosowanie w narzędziu wspomagania pracy tłumacza.
- Przyszłość - zebranie znacznej liczby fraz przydatnych tłumaczowi dzięki mechanizmowi *phraseSearch.*
- Opracowanie detektora fraz najbardziej użytecznych dla tłumacza (bootstrapping na CommonCrawl, *distributional similarity,* klasyfikacja za pomocą regresji).
- Detektor zostanie użyty do obliczania oceny najlepszego pokrycia w algorytmie Concordia.

Dalsza lektura

Concordia w publikacjach:

- Jaworski, R; Dunđer, I; Seljan, S. Usability Analysis of the Concordia Tool Applying Novel Concordance Searching. Springer Verlag, 2016 (in print). Presented at HrTAL2016 10th International Conference on Natural Language Processing (HrTAL2016), Dubrovnik, Croatia

- Jaworski R.: "A novel method for finding and scoring valuable translation memory repetitions", Human Language Technologies as a Challenge for Computer Science and Linguistics, pp. 155-159, 2015

- Jaworski R.: "Approximate sentence matching and its applications in corpus-based research", The Future of Information Sciences: e-Institutions, Openness, Accessibility and Preservation, pp. 21-30 (**keynote paper**)

Bibliografia

- Alabau V., Bonk R., Buck C., Carl M., Casacuberta F., García-Martínez M., González J., Koehn P., Leiva L., Mesa-Lao B., Ortiz D., Saint-Amand H., Sanchis G, Tsoukala C. “CASMACAT: An open source workbench for advanced computer aided translation”, The Prague Bulletin of Mathematical Linguistics, vol. 100, pages 101-112, 2013
- Burkhardt S., Karkkainen J. Fast lightweight suffix array construction and checking. Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching, LNCS 2676, Springer, pp. 55-69, 2003.
- Ferragina P., Manzini G., Mäkinen V., Navarro G. Alphabet-friendly FM-index. Proceedings of SPIRE 2004, pp. 150-160, LNCS 3246, 2004.
- Ko P., Aluru S. Space-efficient linear time construction of suffix arrays. Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching, pp. 200-210, 2003.

Bibliografia

- Jaworski, R; Dunđer, I; Seljan, S. “Usability Analysis of the Concordia Tool Applying Novel Concordance Searching.” Springer Verlag, 2016 (in print). Presented at HrTAL2016 10th International Conference on Natural Language Processing (HrTAL2016), Dubrovnik, Croatia 29 September–1 October 2016.
- Koehn, P., Senellart, J. “Fast approximate string matching with suffix arrays and A* parsing”, Proceedings of AMTA 2010: The Ninth Conference of the Association for Machine Translation in the Americas , 2010
- Somers, H., “Translation Memory Systems”. In: H. Somers (ed.), Computers and Translation: A Translator’s Guide. Amsterdam/Philadelphia: John Benjamins, 31-46, 2003
- O’Brien, S., “Practical Experience of ComputerAided Translation Tools in the Software Localisation Industry”. In: L. Bowker, M. Cronin, D. Kenny & J. Pearson (eds), Unity in Diversity? Current Trends in Translation Studies. Manchester: St. Jerome Publishing, 115-122, 1998

Dziękuję za uwagę!

Seminarium ZIL, IPI PAN, Warszawa