Spis treści

1 Imię i nazwisko

Jan Daciuk

2 Posiadane dyplomy, stopnie naukowe — z podaniem nazwy, miejsca i roku ich uzyskania oraz tytułu rozprawy doktorskiej

- magister inżynier w specjalności „Budowa i oprogramowanie maszyn cyfrowych" — uzyskany na Wydziale Elektroniki Politechniki Gdańskiej w 1986 roku na podstawie przedstawionej pracy dyplomowej pt. *Makroasembler Z-80*.
- doktor nauk technicznych w zakresie informatyki — uzyskany na Wydziale Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej 26 stycznia 1999 roku na podstawie przedstawionej rozprawy doktorskiej pt. *Przyrostowe tworzenie automatów skończonych i ich wykorzystanie do przetwarzania języka naturalnego*.

3 Informacje o dotychczasowym zatrudnieniu w jednostkach naukowych

- 1.07.1988 do 28.02.1999 — asystent naukowo-dydaktyczny, Wydział Elektroniki (później Wydział Elektroniki, Telekomunikacji i Informatyki), Politechnika Gdańska
- 1.03.1999 do 28.02.2013 — adiunkt naukowo dydaktyczny, Wydział Elektroniki, Telekomunikacji i Informatyki, Politechnika Gdańska
- 1.02.2000 do 31.01.2003 — postdoc, Alfa Informatica, Faculteit der Letteren, Rijksuniversiteit Groningen, Holandia
- 1.10.2004 do 30.11.2004 — visiting professor, Ecole Polytechnique de l'Universite de Tours, Francja
- 1.03.2013 do chwili obecnej — starszy wykładowca, Wydział Elektroniki, Telekomunikacji i Informatyki, Politechnika Gdańska

4 Wskazanie osiągnięcia wynikającego z art. 16 ust.2 ustawy z dnia 14 marca 2003 r. o stopniach naukowych i tytule naukowym oraz o stopniach i tytule w zakresie sztuki (Dz. U. nr 65, poz. 595 ze zm.)

4.1 Tytuł osiągnięcia naukowego

„Optimization of automata"

4.2 Autor/autorzy, tytuł/tytuły publikacji, rok wydania, nazwa wydawnictwa

Monografia w języku angielskim: Jan Daciuk, *Optimization of Automata*, Gdańsk University of Technology Publishing House, Gdańsk, 2015. ISBN 978-83-7348-564-9. Dostępna także w Pomorskiej Bibliotece Cyfrowej pod adresem:
`http://pbc.gda.pl/dlibra/docmetadata?id=44644&from=&dirids=1&ver_id=&lp=1&QI=`.

4.3 Omówienie celu naukowego ww. pracy/prac i osiągniętych wyników wraz z omówieniem ich ewentualnego wykorzystania

Monografia może być postrzegana jako przewodnik po mojej pracy naukowej zbierający jej wyniki w jednym miejscu. Zawiera tematy poruszane we wcześniejszych moich publikacjach. Przygotowanie monografii wiązało się z przeglądem, ponowną weryfikacją i krytyczną oceną, co pozwoliło uzyskać w niektórych przypadkach inny, lepszy, do tego jednolity opis, a nawet dalsze usprawnienia algorytmów. W poniższym tekście odwołania do rozdziałów monografii oznaczam dla zwięzłości wielką literą R, np. rozdział trzeci jako R3, a podrozdział drugi czwartego rozdziału jako R4.2.

Moje badania zmierzały do poprawy jakości algorytmów i struktur danych dotyczących różnego rodzaju automatów wykorzystywanych głównie w przetwarzaniu języka naturalnego i przetwarzaniu mowy — szybko rozwijających się, przyszłościowych dziedzin oferującej zarówno naturalny interfejs systemów komputerowych, w tym także inteligentnych telefonów i innych urządzeń programowalnych, jak i dostęp do wiedzy przechowywanej głównie w postaci tekstów z możliwością ich przetwarzania. Ulepszenia dotyczyły zwiększenia szybkości działania, zmniejszenia zapotrzebowania na pamięć operacyjną, ale także możliwości innej, lepszej metody realizacji zadań takich jak dostęp do danych przez słowa lub drzewa, np. drzewa rozbioru składniowego. Usprawnienia dotyczą sytuacji, w których można wykorzystać szczególną postać danych lub gdy istnieją bądź to ograniczenia dostępnych zasobów (jak np. w telefonach, zegarkach, Internecie rzeczy), bądź to istnieje wyraźna preferencja użytkownika zwiększenia szybkości kosztem zajętości pamięci lub odwrotnie. Ulepszyłem także złożoność czasową istniejącego algorytmu minimalizacji automatu skończonego o dwa rzędy wielkości.

Automatami zajmowałem się także przed uzyskaniem doktoratu. Jednak były to wyłącznie minimalne, acykliczne, deterministyczne automaty skończone. Należy podkreślić, że w pracach po uzyskaniu doktoratu zajmowałem się różnymi innymi typami automatów, jak automaty cykliczne, automaty pseudominimalne i wstępujące automaty drzewiaste. Zajmowałem się też algorytmem minimalizacji dowolnych automatów skończonych.

Do moich najważniejszych osiągnięć zaliczam:

1. 9 nowych algorytmów tworzenia automatów minimalnych lub pseudominimalnych, w tym dwa algorytmy dodawania słów do minimalnego, deterministycznego, cyklicznego automatu skończonego, dwa algorytmy tworzenia pseudominimalnych, deterministycznych automatów skończonych i cztery algorytmy tworzenia minimalnych i pseudominimalnych, deterministycznych, wstępujących automatów drzewiastych — opisane dalej w punkcie 4.3.2. Algorytmy te charakteryzują się bardzo małymi wymaganiami pamięciowymi przy zachowaniu bardzo dużej szybkości działania, co odróżnia je od dotychczas

powszechnie stosowanych.

2. Udoskonalenie algorytmu przyrostowej minimalizacji deterministycznych automatów skończonych — opisane dalej w punkcie 4.3.3 — w sensie złożoności czasowej z $\mathcal{O}(|Q|^4)$ do $\mathcal{O}(|Q|^2 \log\log|Q|)$.

3. Opracowanie, wdrożenie i porównanie wybranych metod reprezentacji deterministycznych automatów skończonych — opisane dalej w punkcie 4.3.4. Wybrane zostały metody, w których zmniejszenie rozmiaru pamięci potrzebnej do przechowywania automatu nie jest okupione znacznym narzutem czasowym przy dostępie do danych.

4. Opracowanie i wdrożenie kilku realizacji funkcji mieszającej, w tym dynamicznej funkcji mieszającej i kilku funkcji mieszających z wykorzystaniem automatów drzewiastych (wcześniej żadna realizacja funkcji mieszającej z wykorzystaniem automatów drzewiastych nie była znana), a także opracowanie i realizacja zwartych modeli języka — opisane dalej w punkcie 4.3.5.

5. Zaprojektowanie i wdrożenie biblioteki funkcji do wykorzystywania automatów skończonych jako słowników z realizacją zwartych modeli języka oraz znaczące rozbudowanie dwóch pakietów programów do budowy i wykorzystywania do przetwarzania języka naturalnego automatów skończonych i automatów Mealy'ego — opisane dalej w punkcie 4.3.6. Pakiety zostały udostępnione społeczności użytkowników na serwerze ftp i są używane w wielu ośrodkach w wielu krajach.

Ważnym aspektem tego dorobku z punkty widzenia informatyki jako nauki technicznej jest pokrycie pełnego cyklu od projektu do wdrożenia znajdującego użytkowników także w ośrodkach zagranicznych.

4.3.1 Motywacja

Celem moich prac badawczych była optymalizacja automatów, głównie do zastosowań z obszaru przetwarzania języka naturalnego. Optymalizację rozumiem jako poszukiwanie rozwiązania, które daje najlepsze wyniki przy jak najmniejszych nakładach. W moich badaniach optymalizacja oznacza przede wszystkim zmniejszanie rozmiaru automatów, chociaż polepszyłem także o prawie dwa rzędy wielkości złożoność czasową algorytmu minimalizacji automatów skończonych i zdefiniowałem funkcje mieszające, w tym doskonałą funkcję mieszającą, dla automatów drzewiastych, co umożliwia realizację niektórych działań w inny, lepszy sposób, na przykład umożliwiają łatwą realizację odwzorowania drzew rozbioru składniowego w prawdopodobieństwa, co może znaleźć wykorzystanie w probabilistycznym analizatorze składniowym. Wykorzystałem doskonałą funkcję mieszającą na zwykłych automatach skończonych do realizacji modeli języka charakteryzujących się małą zajętością pamięci przy krótkim czasie dostępu.

Można coraz częściej słyszeć opinie, że zmniejszanie obszaru pamięci zajmowanego przez dane nie ma sensu, ponieważ rozmiar dostępnych pamięci stale rośnie. Dotyczy to także słowników w postaci automatów. Należy jednak podkreślić, że ze wzrostem dostępnej pamięci rośnie też rozmiar wykorzystywanych danych. W moich badaniach używałem słowników w postaci automatów, które zajmowały w gotowej postaci około 0.5 MB pamięci. Jednak inni badacze

używają znacznie większych słowników. Na przykład na Politechnice w Brnie moje oprogramowanie jest używane do tworzenia słowników, których rozmiar przekracza 256 MB (taki słownik ma ponad 16 milionów haseł). Używane są do indeksowania bazy wiedzy powstałej na bazie Wikipedii.[1] Chociaż końcowy rozmiar automatów może być istotny, to ważniejszy jest rozmiar automatu w trakcie jego tworzenia. Dla tworzonych obecnie słowników nawet o średnim rozmiarze zabrakłoby miejsca w największej dostępnej pamięci wirtualnej dla komputerów klasy PC w trakcie tworzenia za pomocą tradycyjnych metod; dla słowników tworzonych na Politechnice w Brnie pamięci wirtualnej ledwie starcza nawet przy użyciu moich metod oszczędzających pamięć. Końcowy rozmiar automatów może być istotny szczególnie gdy oprogramowanie wykorzystuje kilka takich słowników lub gdy jest wykorzystywane na urządzeniach o ograniczonej pamięci (tablety, telefony komórkowe, zegarki, samochodowe urządzenia do nawigacji satelitarnej itp., a ostatnio także Internet rzeczy).

Przetwarzanie języka naturalnego i przetwarzanie mowy stanowią obecnie jedne z najszybciej rozwijających się dziedzin. Jest też na nie coraz większe zapotrzebowanie na rynku. Coraz więcej firm szuka specjalistycznych rozwiązań, które pozwoliłyby im zdobyć przewagę nad konkurencją. Wynika to z naturalności interfejsu głosowego (możliwości sterowania komputerem lub wręcz rozmowy z nim w języku naturalnym) i z faktu, że większość ludzkiej wiedzy przechowywana jest w postaci tekstowej. Moje badania i prace rozwojowe zmierzają do tego, by algorytmy i metody przetwarzania języka naturalnego i przetwarzania mowy można było zastosować w praktyce w dowolnych okolicznościach, w szczególności na różnorodnym sprzęcie i z różnymi danymi. Poszczególne składowe osiągnięcia zgłoszonego do oceny umożliwiają zastosowanie metod z tych dziedzin tam, gdzie mogłoby to być niemożliwe z użyciem innych algorytmów ze względu na ograniczenia mocy obliczeniowej, w szczególności dostępnej pamięci i szybkości procesorów. Niektóre wykorzystują szczególne cechy danych, takie jak ich uporządkowanie.

W przetwarzaniu języka naturalnego automaty mają szereg zastosowań:

- jako słowniki,
- jako narzędzia do realizacji reguł fonologicznych i ortograficznych (w morfologii, w rozpoznawaniu i syntezie mowy),
- jako narzędzia do płytkiej analizy składniowej,
- jako narzędzia do rozwijania skrótów i liczb w syntezie mowy,
- jako narzędzia do wyodrębniania informacji z tekstów.

Wyniki moich badań służą przede wszystkim wykorzystaniu automatów jako słowników, w tym:

- list słów (np. do korekty pisowni),
- słowników morfologicznych,
- analizatorów morfologicznych nieznanych słów (słów, których brak w słowniku — słownik *a tergo*),

[1] Informacje uzyskane od Petera Hostačný'ego — członka zespołu projektowego.

- słowników frekwencyjnych, słowników do tłumaczenia itp., zawierających informacje powiązane ze słowami.

Automaty drzewiaste używane są w wielu dziedzinach, a w przetwarzaniu języka naturalnego używane są głównie w połączeniu z językiem znacznikowym XML. Opracowane przeze mnie algorytmy nadają się szczególnie do użycia w oznaczonych zbiorach tekstów (ang. *annotated corpora*).

Automaty wykorzystywane w badaniach nad optymalizacją zgłoszonych jako moje osiągnięcie to automaty deterministyczne, chociaż niektóre moje wyniki (np. związane z reprezentacją automatów) stosują się także do automatów niedeterministycznych. Większość wyników dotyczy zwykłych automatów skończonych (bez wyjścia), ale część dotyczy automatów drzewiastych wstępujących.

Zmniejszenie końcowego rozmiaru automatu osiąga się na poziomie logicznym za pomocą minimalizacji. Minimalizacja oznacza takie przekształcenie automatu, które bez zmiany języka automatu zmniejsza liczbę stanów do najmniejszej możliwej. Dla automatów deterministycznych istnieje tylko jeden automat minimalny o danym języku i minimalna liczba stanów pociąga za sobą minimalną liczbę przejść. Głównie od liczby przejść zależy rozmiar automatu w pamięci. Do tej grupy należą algorytm przyrostowej minimalizacji dowolnego automatu deterministycznego (R7 i [A15]) i algorytmy budowy minimalnego automatu deterministycznego: zwykłego automatu bez wyjścia (R3.2 i R4.3 oraz [A5, A4]) i wstępującego automatu drzewiastego (R4.4 i [A2]). Minimalizacja może podlegać pewnym ograniczeniom. W szczególności można wymagać, aby w automacie znalazły się *przejścia własne* dla przechowywanych w nich danych (jest to możliwe tylko dla automatów acyklicznych, czyli pozbawionych pętli). Przejścia te nie są współdzielone z żadnymi innymi danymi przechowywanymi w automacie. Własność ta jest szczególnie przydatna do realizacji dowolnej funkcji mieszającej przyporządkowującej każdej danej dowolne informacje — na ogół stosuje się przyporządkowanie im pewnej liczby. Minimalny automat posiadający tę własność nazywa się automatem *pseudominimalnym*. Budowa automatów pseudominimalnych została omówiona w podrozdziałach monografii R3.3, R4.2, R5.1.1 oraz w [A11] (zwykłe automaty bez wyjścia), a także w R4.5, R5.1.4 i w [A9] (automaty drzewiaste).

Duża część badań dotyczyła nie tylko budowy automatu minimalnego lub pseudominimalnego, ale takiej jego budowy, która wymaga jak najmniejszej pamięci. Wynika to z faktu, że rozmiar automatu i ewentualnie innych pomocniczych struktur danych zależnych od rozmiaru automatu w trakcie jego tworzenia mogą być znacznie większe niż końcowy wynik. Zmniejszenie rozmiaru automatu w trakcie tworzenia osiąga się przez konstrukcję *przyrostową*. Polega ona na tym, że gdy do automatu zostaje dodana pojedyncza dana (napis lub słowo w przypadku automatu skończonego, drzewo w przypadku automatu drzewiastego), automat jest minimalizowany, ale taka minimalizacja wykorzystuje wiedzę o już zbudowanym automacie, tak że jedynie odbywa się poszukiwanie stanów równoważnych do nowych stanów lub stanów zmienionych w trakcie dodawania ostatniej danej. Umożliwia to znaczne zmniejszenie nakładu pracy w porównaniu z minimalizacją od początku. Przyrostowa konstrukcja została omówiona w R3 i R4. Możliwa jest też niepełna przyrostowość — półprzyrostowość, definiowana jako proces, w którym automat po dodaniu nowej danej jest zmniejszany, ale nie minimalizowany, tzn. możliwa jest dalsza redukcja jego rozmiaru (liczby stanów). Po zakończeniu dodawania wszystkich danych automat zostaje poddany dodatkowej minimalizacji, która także jest tylko miejscowa,

czyli dotycząca jedynie zmienionej części automatu, a więc wymagająca mniejszego nakładu pracy niż minimalizacja od początku. Zaletą takiego podejścia jest szybkość. Opracowane przeze mnie algorytmy półprzyrostowe zostały omówione w R5 oraz w [A11, A6].

Zmniejszenie rozmiarów może też odbywać się na poziomie reprezentacji. Logicznie automat pozostaje taki sam, natomiast stosowane są przeze mnie techniki takie jak przechowywanie reprezentacji jednego stanu wewnątrz reprezentacji innego stanu (współdzielenie pamięci), stosowanie wartości domyślnych, przechowywanie liczb na najmniejszej możliwej liczbie bajtów itp. Moje badania dotyczące kompresji automatów zostały omówione w R8. Można też znaleźć je w [A12, A14].

Słownik maszynowy to nie tylko prosta lista słów. Ze słowami mogą być powiązane różne informacje, jak np. część mowy, wymowa czy znaczenie. Jeżeli informacje te zależą od postaci słów, można je przechowywać bezpośrednio w słowniku stosując odpowiednie kodowanie. Dzieje się tak np. w słownikach morfologicznych. Próba umieszczenia informacji, które nie są związane z postacią słowa bezpośrednio w automacie, skończyłyby się tak wielkim zwiększeniem jego wielkości, że metoda ta nie miałaby żadnego praktycznego zastosowania. Dlatego stosuje się inne rozwiązania. Słowa kojarzone są z liczbami, które pełnią rolę indeksów w dodatkowych strukturach danych wykorzystywanych w słownikach. Technika ta nazywana jest funkcją mieszającą w automatach. Na ogół stosuje się minimalną, doskonałą funkcję mieszającą w powiązaniu z automatami minimalnymi. Funkcja jest minimalna, ponieważ zwraca spójny zakres liczba całkowitych. Jest doskonała, ponieważ różnym słowom przyporządkowuje różne liczby. Tradycyjne funkcje mieszające nie mają tej właściwości, co prowadzi do konfliktów i konieczności ich rozwiązywania. Automaty pseudominimalne pozwalają na stosowanie dowolnej funkcji mieszającej, w tym z wartościami, które nie muszą być liczbami. Realizacja funkcji mieszającej została omówiona w R6 oraz w artykułach [A1, A9, A10, A7]. Szczególnym zastosowaniem tej techniki jest reprezentacja modeli języka [A13]. Najczęściej wykorzystywanym modelem języka są n-gramy czyli liczby wystąpień kolejnych n słów w tekście. Zaproponowana i zrealizowana przeze mnie w bibliotece `fadd` reprezentacja przechowuje słowa w minimalnym automacie z minimalną, doskonałą funkcją mieszającą, zaś same n-gramy zawierają numery słów; prowadzi to do oszczędności pamięci bez straty szybkości dostępu do danych.

Uzyskane wyniki w zakresie zgłoszonego osiągnięcia zostały podsumowane w monografii. Można je podzielić na pięć grup:

- tworzenie automatów (R3–R5),
- minimalizacja automatów (R7),
- kompresja automatów (R8),
- realizacja dodatkowych funkcji za pomocą automatów (funkcja mieszająca, modele języka – R6),
- wdrożenia powyższych algorytmów i metod.

4.3.2 Osiągnięcia w zakresie tworzenia automatów

Algorytmy tworzenia automatów mają za zadanie utworzenie automatu o danej własności lub własnościach. Tą własnością jest w większości przypadków minimalność (najmniejsza liczba

stanów wśród automatów rozpoznających ten sam język). Jeśli dodatkową własnością automatu jest posiadanie przynajmniej jednego własnego przejścia dla każdego słowa rozpoznawanego przez automat, wówczas powstający automat jest pseudominimalny.

Dodawanie słów do minimalnego, cyklicznego automatu skończonego (Alg1 i Alg2). Wśród tej grupy algorytmów, opracowałem dwa nowe algorytmy do dodawania słów do minimalnego automatu skończonego cyklicznego. Dany jest istniejący cykliczny automat skończony zbudowany za pomocą dowolnej, innej metody oraz zbiór słów, które należy dodać do języka automatu. Pierwsze rozwiązanie tego problemu zostało zaproponowane przez Rafaela Carrasco i Mikela Forcadę w [C1]. Działa w czasie $\mathcal{O}(n)$, gdzie n jest łączną liczbą znaków w dodawanych słowach, przy założeniu, że operacje na rejestrze stanów są dokonywane w stałym czasie. Było rozszerzeniem mojego wcześniejszego algorytmu (przytoczonego jako R4.1, a opisane wcześniej w [B3, B2, drugi algorytm]). Rozwiązanie to jest uniwersalne w tym sensie, że nie wymaga posortowania danych. Jednak taka uniwersalność kosztuje. Zauważyłem, że posortowanie danych umożliwia przyspieszenie konstrukcji poprzez uniknięcie wielokrotnego przetwarzania tych samych stanów automatu. Przetwarzanie stanów posiadających więcej niż jedno przychodzące przejście wiąże się z koniecznością klonowania takiego stanu, tzn. skopiowania stanu wraz ze wszystkimi jego wychodzącymi przejściami. Sklonowanie jednego stanu w ścieżce rozpoznającej przedrostek dodawanego słowa powoduje automatycznie konieczność klonowania wszystkich kolejnych stanów na tej ścieżce, co jest kosztowne. Powoduje też konieczność usunięcia ich z rejestru przechowującego stany w przetworzonej, „zminimalizowanej" części automatu. Po dodaniu słowa stany te należy ponownie porównać z innymi stanami automatu zawartymi w rejestrze i jeśli znaleziony zostanie stan równoważny, bieżący stan powinien zostać nim zastąpiony. Stany z więcej niż jednym przejściem przychodzącym powstają w wyniku minimalizacji. Wielokrotne przetwarzanie tych samych stanów jest następstwem decyzji o całkowitej minimalizacji automatu po dodaniu pojedynczej danej. Dodanie następnej danej może sprawić, że prawostronny język stanu się zmieni w taki sposób, że stan stanie się równoważnym innemu stanowi w automacie i stanie się zbędny. Posortowanie danych wejściowych pozwala opóźnić minimalizację do czasu, gdy stan nie będzie już zmieniany, a więc stany z więcej niż jednym przychodzącym przejściem w ścieżce przedrostków dodawanych słów będą pochodzić tylko z oryginalnego automatu.

Algorytm Alg1 dodawania słów do minimalnego, cyklicznego automatu skończonego (R3.4 i [A5]) jest przyrostowy i wymaga danych posortowanych alfabetycznie. Chociaż ma tę samą asymptotyczną złożoność obliczeniową co algorytm Carrasco i Forcady z [C1], jest szybszy potrzebując jednocześnie nieznacznie mniej pamięci. Od algorytmu [B3, B2, drugi algorytm] (opisanego też w R4.1) różni się tym, że klonuje stany z więcej niż jednym przychodzącym przejściem, jeżeli takowe napotyka. Stanów takich nie napotyka w części automatu utworzonej w wyniku dodania nowych słów. Można zauważyć, że nie jest konieczne sortowanie danych wejściowych w całości. Często występuje sytuacja, w której dane ze źródła danych przychodzą w posortowanych fragmentach, np. kolejne formy odmienione (formy fleksyjne) odmiany (paradygmatu) konkretnego słowa. Wówczas algorytm można wywołać dla każdego takiego fragmentu. Jest to nadal szybsze niż wywołanie uniwersalnego algorytmu dla całości danych.

Algorytm Alg2 dodawania słów do minimalnego, cyklicznego automatu skończonego (R5.1.2 i [A6]), półprzyrostowy, jest jeszcze szybszy (przy tej samej asymptotycznej złożoności oblicze-

niowej) kosztem większej pamięci. Wymaga danych posortowanych według malejącej długości słów. Wykorzystuje ten sam mechanizm klonowania napotykanych stanów i opóźniania minimalizacji w tworzonej części automatu co pierwszy algorytm, rozwijając koncepcję z algorytmu Watsona [C4]. Oba algorytmy mogą być stosowane do budowy słowników rozpoznających obok zwykłych, skończonych słów także słowa o (potencjalnie) nieograniczonej długości, np. liczby, adresy internetowe itp., kiedy dane do słownika mogą zostać wcześniej posortowane lub są dostępne w postaci posortowanej. Mogą być wykorzystywane do dodawania do słowników słów partiami. Oba algorytmy zostały opisane w [D3], gdzie uwydatnione są podobieństwa między nimi, a jeden z algorytmów jest przedstawiony w innej, prostszej, lepszej formie.

Tworzenie pseudominimalnych automatów skończonych (przyrostowe Alg3 i Alg4 oraz półprzyrostowy Alg5). Następną podgrupę stanowią algorytmy (R3.3, R4.2, R5.1.1 i [A11]) przyrostowego i półprzyrostowego tworzenia pseudominimalnych automatów skończonych. Takie automaty są zawsze acykliczne. Zostały po raz pierwszy uzyskane przez Revuza [C3]. Starał się opracować przyrostowy algorytm budowy automatu minimalnego. Okazało się jednak, że algorytm wymaga dodatkowej fazy minimalizacji, ponieważ automaty powstające stopniowo podczas dodawania nowych słów i zmniejszaniu rozmiaru automatu nie były minimalne. Innymi słowy powstały algorytm [C3] był półprzyrostowy, a automaty otrzymane w pierwszej fazie to właśnie automaty pseudominimalne. Ich ciekawe właściwości zostały zauważone po raz pierwszy przez Denisa Maurela [C2] — mogą być użyte do realizacji dowolnej (nie tylko minimalnej doskonałej) funkcji mieszającej. Algorytm Dominique'a Revuza tworzy automaty pseudominimalne w szczególny, oryginalny sposób. Zaproponowane przeze mnie dwa nowe algorytmy zostały zainspirowane [B3, B2] i algorytmem Watsona [C4] w celu obejścia niedogodności algorytmu Revuza. Zauważyłem, że prawidłowe otrzymywanie automatu pseudominimalnego może być bardzo podobne do minimalizacji. Kluczową rolę w minimalizacji (oprócz algorytmu Janusza Brzozowskiego, który znacząco różni się od wszystkich innych ogólnych algorytmów minimalizacji) odgrywa relacja równoważności. W automacie minimalnym każda klasa abstrakcji (klasa równoważności) ma dokładnie jednego przedstawiciela. Relacja równoważności występuje w twierdzeniu Myhill-Nerode, na którym opiera się minimalizacja (oprócz algorytmu Janusza Brzozowskiego). Relacja równoważności z definicji musi spełniać trzy warunki: zwrotność, symetryczność i przechodniość. Relacja pseudorównoważności stanów, w której dodatkowo wymaga się, by prawostronne języki stanów zawierały pojedyncze słowa, także spełnia te warunki.

Algorytm Alg3 (R3.3) wymaga danych posortowanych alfabetycznie i charakteryzuje się małą zajętością pamięci i dużą prędkością. W stosunku do pierwszego algorytmu tworzącego takie automaty (autorstwa Revuza [C3]) wymaga danych posortowanych w zwykły sposób (często dane są już tak posortowane), podczas gdy oryginalny algorytm Revuza wymaga danych posortowanych alfabetycznie według odwróconych słów (pierwsza litera jest ostatnią itd.). Algorytm Alg4 (R4.2) działa na danych nieuporządkowanych, co okupione jest trochę wolniejszym działaniem przy praktycznie tych samych wymaganiach pamięciowych. Algorytmy mogą być wykorzystywane do tworzenia automatów realizujących dowolną funkcję mieszającą, np. przy dynamicznym słowniku. Tak więc w stosunku do oryginalnego algorytmu Dominique'a Revuza oba moje algorytmy cechują się większą elastycznością i wygodą użycia.

Algorytm Watsona [C4] uważany jest za najszybszy algorytm tworzenia minimalnych auto-

matów skończonych[2]. Jest algorytmem półprzyrostowym, tak jak oryginalny algorytm Dominique'a Revuza. Zastąpienie przeze mnie zwykłej relacji równoważności relacją pseudorównoważności prowadzi do algorytmu półprzyrostowego tworzenia automatów pseudominimalnych (R5.1.1). Należy jednak zauważyć istotną różnicę w stosunku do pierwszej fazy algorytmu Revuza, którego pierwsza faza jest całkowicie przyrostowym algorytmem tworzenia automatów pseudominimalnych i nie wymaga dodatkowej minimalizacji. Moje rozszerzenie Alg5 algorytmu Watsona jest algorytmem półprzyrostowym, który jest wprawdzie wolniejszy od pierwszej fazy algorytmu Revuza, natomiast jest szybszy od algorytmów przyrostowych Alg3 i Alg4 opisanych w poprzednim akapicie. Zaletą w stosunku do algorytmu Revuza jest prostsze przygotowanie danych. Tak jak dla algorytmu Watsona, dane muszą być posortowane dowolnie w kolejności malejącej długości słów — takie sortowanie jest znacznie szybsze niż sortowanie leksykograficzne; można go dokonać w czasie liniowym, np. przez sortowanie kubełkowe (*bucket sort*). Dlatego mimo że sam algorytm ustępuje szybkością pierwszej fazie algorytmu Revuza, to rozpatrując algorytmy łącznie z fazą przygotowania danych, mój algorytm jest szybszy.

Tworzenie wstępujących automatów drzewiastych: nowe przyrostowe algorytmy Alg6 i Alg7 oraz nowe półprzyrostowe algorytmy Alg8 i Alg9. Ostatnią podgrupę algorytmów budowy automatów tworzą dwa przyrostowe i dwa półprzyrostowe algorytmy tworzenia wstępujących automatów drzewiastych. Należy tu podkreślić, że wszystkie one są zalążkiem nowej klasy algorytmów dla automatów drzewiastych wstępujących. Dotychczasowe, znane algorytmy tworzą najpierw automat bez znaczącego ograniczania jego rozmiaru, a potem go minimalizują. Wymaga to dużej pamięci — podobnie jak w przypadku automatów skończonych. Nowe, opracowane przeze mnie algorytmy mają znacznie mniejsze wymagania pamięciowe.

Mój pierwszy algorytm przyrostowy Alg6 (R4.4 i [A2]) tworzenia automatów drzewiastych opracowany został we współpracy z Rafaelem Carrasco i Mikelem Forcadą — po tym gdy Rafael Carrasco zwrócił się do mnie o pomoc w znalezieniu błędu w jego algorytmie. Jego algorytm dawał poprawne wyniki, ale utworzenie automatu dla większego zbioru danych (zbioru drzew) trwało więcej niż dwie doby na notebooku Sony Vaio PCG-C1XS (12GB twardy dysk, 64MB RAM, 400MHz Pentium 2). Powodem było niewłaściwe zrozumienie przez niego idei przyrostowości — co prawda automat był minimalizowany po dodaniu każdego drzewa, ale była to minimalizacja globalna, która nie wykorzystywała wiedzy o stanach wcześniej utworzonego automatu. Mój wkład polegał na wprowadzeniu pełnej przyrostowości, tak że minimalizacja jest dokonywana jedynie lokalnie i dotyczy tylko stanów dodanych lub zmienionych w wyniku dodania nowego drzewa. Moje ulepszenie jest istotne: w pełni przyrostowy algorytm był w stanie zbudować automat drzewiasty z tego samego zbioru drzew w kilka minut. Ten algorytm tworzy automat minimalny. Ma wiele cech wspólnych z drugim algorytmem w [B3, B2], chociaż należy tu podkreślić, że automaty drzewiaste są znacząco różne od zwykłych automatów skończonych i intuicja wyrosła na gruncie zwykłych automatów często może zwodzić badaczy. Przykładowo w automatach drzewiastych odpowiednikiem ścieżki jest drzewo, w którym kolejność przetwarzania węzłów (stanów) nie jest ściśle określona i niektóre stany występują wielokrotnie. Jest to algorytm Alg6 przyrostowego tworzenia automatów drzewiastych ze zbioru drzew.

[2]Opinia wyrażona w rozmowie przez prof. Watsona z Technische Universiteit Eindhoven, University of Pretoria i Stellenbosch University przytaczającego wyniki doświadczeń dokonanych przez osobę trzecią

Dotychczasowe metody nie są przyrostowe, a więc wymagają dużo pamięci do przechowywania pośredniego, nieminimalnego automatu.

Mój algorytm Alg7 (przyrostowy dla automatów drzewiastych) R4.5 i [A9, A1] zaproponowany wspólnie z Rafaelem Carrasco tworzy automat pseudominimalny. Taki automat pozwala zrealizować dowolną funkcję mieszającą działającą na drzewach. Algorytm został opracowany w oparciu o opisany w poprzednim akapicie algorytm tworzenia minimalnych drzewiastych automatów wstępujących. Musieliśmy zmienić definicję równoważności, podobnie jak dla automatów skończonych. Dodatkowy warunek, który zamienia równoważność w pseudorównoważność, jest trochę inaczej sformułowany dla drzew. Spore trudności badaczom z doświadczeniem z dziedziny zwykłych automatów skończonych sprawia określenie stanów odpowiadających stanom zbieżnym (z lewostronnym językiem zawierającym więcej niż jeden element) i stanom rozbieżnym (z prawostronnym językiem zawierającym więcej niż jeden element) w zwykłych automatach skończonych. Lewostronny język stanu q to zbiór słów powstałych przez sklejenie kolejnych etykiet przejść na ścieżkach prowadzących od stanu początkowego do stanu q. Prawostronny język stanu q to zbiór słów powstałych przez sklejenie kolejnych etykiet przejść na ścieżkach prowadzących od stanu q do stanów końcowych. Wyszukiwanie tych stanów opisane w [A9, A1] nie jest w pełni wydajne, ale zostało poprawione w mojej monografii w R4.5. Oba algorytmy przyrostowe wymagają mało pamięci przy dużej prędkości działania i nie potrzebują sortowania danych. Są wykorzystywane do tworzenia automatów przechowujących struktury XML z możliwością realizacji funkcji mieszającej, przy czym w przypadku pierwszego algorytmu funkcja jest minimalna, natomiast drugiego — dowolna.

W R5.1.3 i R5.1.4 oraz w pracy [A8] zaproponowałem ponadto algorytmy półprzyrostowe, które także tworzą odpowiednio automat minimalny i pseudominimalny. Od przyrostowych różnią się większą prędkością i większą zajętością pamięci w czasie procesu tworzenia automatu. Oparte są na półprzyrostowym algorytmie Bruce'a Watsona [C4] przystosowanym do automatów drzewiastych. W moich algorytmach tworzenia automatów skończonych i drzewiastych istotne i oryginalne jest użycie tablicy rozproszonej z funkcją mieszającą pozwalającą szybko odnaleźć stanu o pożądanych właściwościach w zminimalizowanej części automatu. Rafael Carrasco zainspirowany tą techniką zaproponował wykorzystanie jej w minimalizacji automatu drzewiastego [D1].

4.3.3 Osiągnięcia w zakresie minimalizacji automatów

Druga grupa z algorytmów będących przedmiotem zainteresowania mojej pracy badawczej to grupa zawierająca algorytmy minimalizacji automatów. Opracowałem dla niej jeden algorytm przyrostowej minimalizacji automatu skończonego. Autorem pierwotnej wersji algorytmu [C5] jest Watson. W istotny ją sposób udoskonaliłem (R7 i [A15]). Algorytm w wersji oryginalnej jest dość prosty. Polega na porównywaniu stanów parami i jeśli równoważność pary stanów zależy od innej pary stanów, to najpierw nastąpi sprawdzenie równoważności tej drugiej pary. Aby zapobiec nieskończonym pętlom, odwiedzone pary przechowuje się w zbiorze odwiedzonych par, który jest opróżniany po ustaleniu równoważności pary stanów na najwyższym poziomie (przy zerowym poziomie rekurencji). Zanim proces sprawdzania równoważności pary stanów w każdym kolejnym rekurencyjnym wywołaniu się rozpocznie, sprawdza się, czy para figuruje już w zbiorze odwiedzanych par. Wówczas para zostaje uznana za równoważną dla potrzeb badania równoważności pary z zerowego poziomu rekurencji. Po ustaleniu równoważności pary

z zerowego poziomu rekurencji, równoważność innych par sprawdzona przy okazji jest zapominana. Dokonałem trzech istotnych udoskonaleń. Pierwsze moje udoskonalenie tego rozwiązania polega na wstępnym posortowaniu stanów ze względu na ich końcowość i na liczbę wychodzących przejść. Takie posortowanie może być wykonane w czasie liniowym (np. przez wspomniane wcześniej sortowanie kubełkowe), więc nie zwiększa istotnie czasu wykonywania całego algorytmu. W zależności od rodzaju wejściowego automatu, posortowanie może znacząco zmniejszyć liczbę sprawdzeń równoważności par stanów poprzez podział przestrzeni stanów na pewną liczbę klas. Moje drugie udoskonalenie polega na zapamiętywaniu w zbiorze odwiedzonych par wyłącznie par z najwyższego poziomu wywołań oraz par ze stanem z więcej niż jednym dochodzącym przejściem. Tylko takie pary mogą rozpoczynać pętle. Dwa pierwsze udoskonalenia mają znaczenie tylko dla szczególnych danych.

Moje trzecie udoskonalenie jest kluczowe dla działania algorytmu. Polega ono na zapamiętywaniu wyników badania równoważności par stanów z głębszych poziomów rekurencji. Te wyniki mają często charakter warunkowy, tzn. zależą od równoważności innych par leżących płycej w drzewie rekurencyjnych wywołań. Stąd konieczność przechowywania i łączenia list zależności. To udoskonalenie zmniejsza złożoność obliczeniową algorytmu z $\mathcal{O}(n^4)$, gdzie n jest liczbą stanów automatu, do prawie $\mathcal{O}(n^2)$. W praktyce oznacza to, że automaty, których minimalizacja za pomocą starszej wersji trwała dłużej niż jedną dobę na procesorze DEC Alpha i musiała zostać przerwana, mogły zostać zminimalizowane w czasie nie przekraczającym jednej doby, a w niektórych przypadkach mogła zostać dokonana ponad 10 000 razy szybciej niż dla wersji pierwotnej. W odróżnieniu od innych opracowanych przeze mnie algorytmów i metod, ten nie jest używany w przetwarzaniu języka naturalnego. Jego zaletą jest możliwość przerwania procesu minimalizacji w dowolnej chwili z uzyskaniem mniejszego, chociaż niekoniecznie minimalnego automatu.

4.3.4 Osiągnięcia w zakresie kompresji automatów

Zmniejszanie liczby stanów nie jest jedynym sposobem zmniejszania rozmiaru automatu w pamięci. Wiele mogłem osiągnąć stosując odpowiednią reprezentację automatu — w pamięci i na dysku. Prowadziłem badania nad takimi reprezentacjami (R8 i [A3]), porównując różne rozwiązania. Zostały one przeze mnie wdrożone w pakiecie programów `fsa` oraz w bibliotece `fadd`. Pakiet `fsa` jest zbiorem programów i skryptów do tworzenia automatów skończonych i ich wykorzystywania w przetwarzaniu języka naturalnego. Opracowany został w ramach pracy doktorskiej, jednak został później wzbogacony o nowe mechanizmy kompresji, nowe programy i nowe funkcje programów opracowanych wcześniej. Pakiet jest wykorzystywany w wielu ośrodkach w kraju i za granicą. Biblioteka `fadd` została opracowana w ramach stażu po doktoracie (*postdoc*) w Alpha Informatica na Rijksuniversiteit Groningen. Zawiera funkcje odpowiadające programom z pakietu `fsa` (oprócz tworzenia automatów). Jest napisana w języku C++, ale zawiera interfejs w języku C, co pozwala na jej wykorzystywanie z innymi językami programowania, np. z Prologiem. Biblioteka `fadd` oprócz mechanizmów czytania automatów poddanych kompresji zawiera również funkcje odczytu modeli języka oraz skrypty służące do tworzenia skompresowanych reprezentacji modeli języka[D6], [A13]. Modele języka zawierają prawdopodobieństwa n-krotek słów przy ustalonej liczbie słów w krotce. Tradycyjnie takie modele przechowuje się w bazach danych. W moim rozwiązaniu słowa w krotkach przechowywane są w automacie z minimalną, doskonałą funkcją mieszającą dostarczającą niepowtarzalny numer

słowa. Ten numer jest przechowywany w tablicy krotek. Taka reprezentacja jest nie tylko oszczędna, ale i szybka w porównaniu do metod wykorzystujących bazy danych.

Mechanizmy kompresji wykorzystane w pakiecie `fsa` i w bibliotece `fadd` nie są jedynymi możliwymi do wykorzystania. Ich wspólną cechą jest zmienna długość reprezentacji pojedynczego przejścia automatu (stany są reprezentowane niejawnie). Zbadałem możliwość użycia reprezentacji ze stałą długością przejścia wykorzystującą wariant kompresji LZ. Wykazałem [A12] m.in., że kompresja LZ powinna być wykonana przed zastosowaniem innych metod, ponieważ w innym przypadku wyniki nie są zachęcające.

4.3.5 Osiągnięcia w zakresie opracowania nowych metod realizacji dodatkowych funkcji za pomocą automatów

W moich badaniach (oprócz przyrostowej minimalizacji) automaty wykorzystywane są w charakterze słowników lub (w przypadku automatów drzewiastych) oznaczonych zbiorów tekstowych. Jednak w niektórych przypadkach przechowanie całej informacji w automacie nie jest wykonalne, ponieważ powodowałoby gwałtowny, nieakceptowalny wzrost jego rozmiaru. Wówczas automat służy do obliczenia indeksu informacji przechowywanej w innych strukturach danych, np. tablicach. Oblicza zatem funkcję mieszającą na przechowywanych słowach (R6). Jeśli automat jest minimalny, to może realizować minimalną, doskonałą funkcję mieszającą (R6.1 i R6.2), czyli przyporządkowywać słowom kolejne numery (numerować słowa). Możliwe jest także obliczanie funkcji odwrotnej, czyli otrzymanie słowa na podstawie jego numeru. W automatach pseudominimalnych, w których wszystkie przechowywane słowa zakończone są specjalnym symbolem — symbolem końca słowa, każde słowo posiada w automacie co najmniej jedno przejście, które należy pokonać rozpoznając dane słowo i które nie jest dzielone z żadnym innym słowem. Można zatem te przejścia własne wykorzystać do przechowania dowolnej informacji związanej z poszczególnymi słowami, czyli także do realizacji dowolnej funkcji mieszającej. Część moich badań dotyczyła realizacji funkcji mieszającej przy dynamicznym słowniku (R6.5 i [A10]). Słownik taki zmienia się w trakcie działania programu: są do niego dodawane nowe słowa, mogą być z niego także usuwane słowa (co jest mniejszym problemem, bo wtedy przenumerowanie słów nie jest konieczne). Ponieważ w realizacji minimalnej, doskonałej funkcji mieszającej przy użyciu automatów minimalnych słowa z języka automatu są numerowane kolejno, dodanie słowa do słownika w postaci automatu minimalnego zmienia numerację słów. Stanowi to pewną trudność w wielu zastosowaniach, w których numer słowa służy jako jego identyfikator i jest indeksem w zewnętrznych strukturach danych przechowujących dodatkowe informacje o słowie. Porównywałem takie sposoby realizacji funkcji mieszającej, które nie zmieniają numeracji słów po dodaniu lub usunięciu słowa. Jednym z nich jest zastosowanie automatów pseudominimalnych. W R6.3 i w [A11] zaproponowałem algorytmy ich tworzenia z uwzględnieniem realizacji funkcji mieszającej, tzn. z rozmieszczaniem dodatkowej informacji służącej realizacji tej funkcji. W R8.3 i [A14] (rozszerzona wersja [D7]) z Dawidem Weissem badaliśmy wpływ nowej metody kompresji na realizację funkcji mieszającej, w szczególności dowolną funkcję mieszającą wykorzystującą automat pseudominimalny. Maurel w [D8] bazując na moich rozwiązaniach w czasie mojego pobytu na Uniwersytecie w Tours we Francji zaproponował jeszcze inny sposób realizacji funkcji mieszającej wykorzystujący niektóre pomysły z moich algorytmów. Opisuję go w R6.4.

Moim nowatorskim pomysłem była propozycja funkcji mieszającej dla wstępujących auto-

matów drzewiastych. Opisałem ją w R6.6 i [A7]. Tak jak słowa są numerowane przez minimalną doskonałą funkcję mieszającą w minimalnym automacie skończonym, tak drzewa są numerowane we wstępującym automacie drzewiastym, pod warunkiem że automat jest acykliczny, czyli jego język jest skończonym zbiorem skończonych drzew. Podałem także sposób realizacji funkcji odwrotnej. Należy podkreślić, że to osiągnięcie znacznie ułatwia indeksowanie zdań bądź części zdań w oznaczonych zbiorach tekstów, jeśli oznaczenia dokonywane są za pomocą XML lub dają się do tego formatu łatwo sprowadzić.

Realizacja dowolnej funkcji mieszającej dla pseudominimalnego wstępującego automatu drzewiastego okazała się trudniejsza. Dokonałem jej wspólnie z Rafaelem Carrasco (R6.7 i [A9, A1]). Opis w monografii (R6.7) został udoskonalony w stosunku do przedstawionego w artykule. Przede wszystkim opis został znacznie uproszczony. Uproszczenie nie sprowadza się tylko do lepszego opisu, udoskonalony algorytm jest także szybszy, ponieważ dla takich samych danych musi dokonać mniej czynności.

Główną motywacją dla moich badań było wykorzystanie wyników w przetwarzaniu języka naturalnego. Algorytmy tworzenia minimalnych automatów skończonych służą przede wszystkim do wydajnego i oszczędnego tworzenia różnego rodzaju słowników (szczegółowe omówienie tych zagadnień zamieściłem z Jakubem Piskorskim i ze Strahilem Ristovem w [D5]). Algorytmy charakteryzują się różną prędkością i zapotrzebowaniem na pamięć —- wyniki badań nad nimi zamieściłem w [A4].

4.3.6 Wdrożenie biblioteki wykorzystującej automaty skończone i rozbudowa dwóch pakietów oprogramowania wykorzystujących automaty skończone i automaty Mealy'ego

Na zaproszenie prof. Gertjana van Noorda odbywałem w okresie od lutego 2000 r. do stycznia 2003 r. w Alfa Informatica, Rijksuniversiteit Groningen w Holandii staż podoktorski. Brałem tam udział w projekcie Alpino, którego celem było opracowanie analizatora składniowego o szerokim pokryciu dla języka niderlandzkiego. Był to projekt PIONIER finansowany przez Nederlandse Wetenschapelijke Organizatie — holenderski odpowiednik KBN. W ramach tego projektu opracowałem bibliotekę `Fadd`. Zawiera ona funkcje umożliwiające wykorzystanie słowników w postaci automatów skończonych do analizy i syntezy morfologicznej, lematyzacji, korekty pisowni, dopisywania znaków diakrytycznych oraz realizacji minimalnej, doskonałej funkcji mieszającej oraz jej odwrotności, a także obsługę skompresowanych modeli języka i skrypty do ich tworzenia. Biblioteka została napisana w języku C++, ale zawiera interfejs w języku C pozwalający korzystać z niej w wielu innych językach programowania, np. w projekcie Alpino była dołączana do programu w języku Prolog. Zastosowanie opracowanej przeze mnie i realizującej moje algorytmy biblioteki w projekcie Alpino zmniejszyło zapotrzebowanie na pamięć ze strony słowników o rząd wielkości i umożliwiło zainstalowanie systemu na komputerze o znacznie mniejszej (o rząd wielkości) pamięci. W następnej wersji systemu zapotrzebowanie na pamięć znowu znacząco wzrosło, ale odbyło się to wskutek użycia znacznie większej ilości danych, co nie byłoby możliwe bez biblioteki `Fadd`. Ponieważ analiza leksykalna stanowi drobny ułamek czasu analizy składniowej, biblioteka nie przyspieszyła znacząco systemu, za to znacznie krótszy stał się czas przygotowania go do pracy. Wcześniej słowniki miały postać tablic rozproszonych i nadanie im wartości początkowych zajmowało kilka minut. W bibliotece `Fadd` te same słowniki były ładowane w ułamku sekundy.

Nieustannie także ulepszałem swoje autorskie pakiety programów `utr` i `fsa` [D2, D4]. Na przykład wzbogaciłem pakiet `fsa` o funkcję odgadywania opisu morfologicznego słowa dla programu `mmorph` — popularnego narzędzia do tworzenia morfologii z instytutu ISSCO w Genewie, dostępnego także jako pakiet systemowy w systemie Ubuntu Linux. Moje rozwiązanie wykorzystuje automaty odgadujące [B1]. Formy odmienione w tekście są analizowane przez automat odgadujący, który przypisuje im różne możliwe formy hasłowe i wzory odmiany. Formy hasłowe i wzory odmiany są grupowane i zliczane, by w za pomocą interfejsu graficznego zaproponować użytkownikowi te, które są najbardziej prawdopodobne. Użytkownik ma możliwość przetestować swoje hipotezy, uzyskując formy odmienione z programu `mmorph` w oknie interfejsu. Oba pakiety korzystają także z moich badań nad wymagającą mało pamięci reprezentacją automatów. Wdrożyłem w nich różne metody, w wyniku czego użytkownik może wybrać metodę, która najlepiej odpowiada jego potrzebom i charakterystyce zadań, do których będzie wykorzystywał oprogramowanie.

Służyłem swoją wiedzą i doświadczeniem w tworzeniu słowników w postaci automatów skończonych pracownikom Instytutu Podstaw Informatyki PAN podczas opracowywania drugiej wersji bardzo dobrego i szeroko stosowanego analizatora morfologicznego Morfeusz. W szczególności nadzorowałem wdrożenie reprezentacji słowników i algorytmów dostępu do słowników.

Opracowałem także realizacje programowe różnych algorytmów tworzenia minimalnych i pseudominimalnych automatów skończonych, które są przedmiotem osiągnięcia zgłoszonego do oceny i udostępniłem je społeczności na serwerze ftp. Zrobiłem to samo dla algorytmów minimalizacji automatów skończonych, włączając w to algorytm minimalizacji przyrostowej z moimi autorskimi ulepszeniami. Mogą one posłużyć jako punkt startowy do realizacji różnych systemów.

Ponieważ biblioteka `Fadd` nie udostępnia funkcji tworzenia automatów, opracowałem nową bibliotekę z takimi funkcjami. Tworzenie automatów jest szybsze niż w pakiecie `fsa`, ale wymaga więcej pamięci i nie wszystkie możliwości programów `fsa_build` i `fsa_ubuild` z pakietu `fsa` są dostępne w nowej bibliotece.

Publikacje własne omawiające zagadnienia przedstawione w monografii

[A1] Rafael Carrasco and Jan Daciuk. A perfect hashing incremental scheme for unranked trees using pseudo-minimal automata. *RAIRO - Theoretical Informatics and Applications*, 43(04):779–790, October 2009. DOI: http://dx.doi.org/10.1051/ita/2009018, IF=0.361, 5yIF=0.451, SJR=0.776, IPP=0.539, SNIP=1.087.

[A2] Rafael C. Carrasco, Jan Daciuk, and Mikel L. Forcada. Incremental construction of minimal tree automata. *Algorithmica*, 55(1):95–110, September 2009. IF=0.917, 5yIF=1.166, SJR=1.433, IPP=1.060, SNIP=1.479, cyt SCOPUS=2, cyt WoS=1.

[A3] Jan Daciuk. Experiments with automata compression. In Sheng Yu and Andrei Păun, editors, *Implementation ans Application of Automata. 5th International Conference CIAA*

2000. London, Ontario, Canada, July 2000. Revised Papers., number 2088 in LNCS, pages 105–112. Springer, 2001. IF=0.415, WoS cyt=3.

[A4] Jan Daciuk. Comparison of construction algorithms for minimal, acyclic, deterministic, finite-state automata from sets of strings. In *Seventh International Conference on Implementation and Application of Automata CIAA 2002*, LNCS, Tours, France, July 2003. Springer. JCR=0.339, IPP=0.356, SNIP=0.460.

[A5] Jan Daciuk. Comments on incremental construction and maintenance of minimal finite-state automata by Rafael C. Carrasco and Mikel Forcada. *Computational Linguistics*, 30(2):227–235, June 2004. IF=1.657, SJR=2.425, IPP=2.417, SNIP=4.921, cyt SCOPUS=2.

[A6] Jan Daciuk. Semi-incremental addition of strings to a cyclic finite automaton. In Krzysztof Trojanowski Mieczysław A. Kłopotek, Sławomir T. Wierzchoń, editor, *Proceedings of the International IIS: IIP WM '04 Conference*, Advances in Soft Computing, pages 201–207, Zakopane, Poland, May 2004. Springer. SJR=0.149, IPP=0.103, SNIP=0.154.

[A7] Jan Daciuk. Perfect hashing tree automata. In Thomas Hanneforth and Kay-Michael Würzner, editors, *proceedings of FSMNLP 2007*, pages 97–106, July 2007.

[A8] Jan Daciuk. *Optimization of Automata*. Gdańsk University of Technology Publishing House, 2014. ISBN 978-83-7348-564-9.

[A9] Jan Daciuk and Rafael Carrasco. Perfect hashing with pseudo-minimal bottom-up deterministic tree automata. In Mieczysław A. Kłopotek, Adam Przepiórkowski, Sławomir T. Wierzchoń, and Krzysztof Trojanowski, editors, *Intelligent Information Systems XVI, Proceedings of the International IIS'08 Conference held in Zakopane, Poland, June 16-18, 2008*. Academic Publishing House Exit, 2008.

[A10] Jan Daciuk, Denis Maurel, and Agata Savary. Dynamic perfect hashing with pseudo-minimal automata. In Mieczysław A. Kłopotek, Sławomir Wierzchoń, and Krzysztof Trojanowski, editors, *Intelligent Information Processing and Web Mining, Proceedings of the International IIS: IIPWM'05 Conference held in Gdańsk, Poland, June 13-16, 2005*, volume 31 of *Advances in Soft Computing*. Springer, 2005. indeksowane przez WoS, SJR=0.149, IPP=0.103, SNIP=0.154.

[A11] Jan Daciuk, Denis Maurel, and Agata Savary. Incremental and semi-incremental construction of pseudo-minimal automata. In Jacques Farre, Igor Litovsky, and Sylvain Schmitz, editors, *Implementation and Application of Automata: 10th International Conference, CIAA 2005*, volume 3845 of *LNCS*, pages 341–342. Springer, 2006. indeksowane przez Web of Science, SJR=0.339, IPP=0.356, SNIP=0.460, WoS cyt=2.

[A12] Jan Daciuk and Jakub Piskorski. Gazetteer compression technique based on substructure recognition. In *Intelligent Information Processing and Web Mining, Proceedings of the International Conference on Intelligent Information Systems, Ustroń, Poland, June 19-22, 2006*, pages 87–95. Book Series Advances in Soft Computing, Vol. 35/2006, Springer,

Berlin-Heidelberg, 2006. SJR=0.149, IPP=0.103, SNIP=0.154, SCOPUS cyt=2, WoS cyt=2.

[A13] Jan Daciuk and Gertjan van Noord. Finite automata for compact representation of tuple dictionaries. *Theoretical Computer Science*, 313(1), 16 February 2004. IF=0.676, SJR=1.096, IPP=0.995, SNIP=1.313, cyt=1.

[A14] Jan Daciuk and Dawid Weiss. Smaller representation of finite-state automata. *Theoretical Computer Science*, 450:10–21, September 2013. IF=0.489, 5yIF=0.697, SJR=0.339, IPP=356, SNIP=0.460, SCOPUS cyt=2.

[A15] Bruce W. Watson and Jan Daciuk. An efficient incremental DFA minimization algorithm. *Natural Language Engineering*, 9(1):49–64, March 2003. SJR=0.725, IPP=0.695, SNIP=2.065, cyt=21.

Publikacje własne przed doktoratem

[B1] Jan Daciuk. Treatment of unknown words. In *proceedings of Workshop on Implementing Automata WIA'99*, pages IX–1 – IX–9, Potsdam, Germany, July 1999.

[B2] Jan Daciuk, Stoyan Mihov, Bruce Watson, and Richard Watson. Incremental construction of minimal acyclic finite state automata. *Computational Linguistics*, 26(1):3–16, April 2000.

[B3] Jan Daciuk, Richard E. Watson, and Bruce W. Watson. Incremental construction of acyclic finite-state automata and transducers. In *Finite State Methods in Natural Language Processing*, Bilkent University, Ankara, Turkey, June – July 1998.

Publikacje innych autorów cytowane w kontekście osiągnięcia

[C1] Rafael C. Carrasco and Mikel L. Forcada. Incremental construction and maintenance of minimal finite-state automata. *Computational Linguistics*, 28(2), June 2002.

[C2] Denis Maurel. Pseudo-minimal transducer. *Theoretical Computer Science*, 231(1):129–139, 2000.

[C3] Dominique Revuz. *Dictionnaires et lexiques: méthodes et algorithmes*. PhD thesis, Institut Blaise Pascal, Paris, France, 1991. LITP 91.44.

[C4] Bruce Watson. A fast new (semi-incremental) algorithm for the construction of minimal acyclic DFAs. In *Third Workshop on Implementing Automata*, pages 91–98, Rouen, France, September 1998. Lecture Notes in Computer Science, Springer.

[C5] Bruce Watson. An incremental DFA minimization algorithm. In Lauri Karttunen, Kimmo Koskenniemi, and Gertjan van Noord, editors, *proceedings of FSMNLP 2001, ESSLLI workshop*, 2001.

Inne publikacje własne po doktoracie

[D1] Rafael Carrasco, Jan Daciuk, and Mikel Forcada. An implementation of deterministic tree automata minimization. In Jan Holoub and Jan Zdarek, editors, *12th International Conference on Implementation and Application of Automata*, number 4783 in LNCS, pages 122–129, Prague, Czech Republic, July 2007. Springer.

[D2] Jan Daciuk. Finite state tools for natural language processing. In *proceedings of the COLING 2000 workshop Using Toolssets and Architectures to Build NLP Systems*, pages 34–37, Luxembourg, August 5th 2000. Centre Universitaire.

[D3] Jan Daciuk. Extension of selected adfa construction algorithms to the case of cyclic automata. In *Eindhoven FASTAR Days 2004*, Eindhoven, 2004.

[D4] Jan Daciuk. Finite-state lexical tools. In Witold Abramowicz, editor, *7th International Conference on Business Information Systems*, pages 21–23, Poznań, April 2004. Wydawnictwo Akademii Ekonomicznej w Poznaniu.

[D5] Jan Daciuk, Jakub Piskorski, and Strahil Ristov. *Scientific Applications of Language Methods*, chapter Natural Language Dictionaries Implemented as Finite Automata, pages 133–204. Imperial College Press, 2010.

[D6] Jan Daciuk and Gertjan van Noord. Finite automata for compact representation of language models in nlp. In *Sixth International Conference on Implementation and Applications of Automata, CIAA'2001*, volume 2494 of *Lecture Notes in Computer Science*, pages 65–73, Pretoria, South Africa, 2003. Springer.

[D7] Jan Daciuk and Dawid Weiss. Smaller representation of finite-state automata. In Beatrice Bouchou-Markhoff, Pascal Caron, Jean-Marc Champarnaud, and Denis Maurel, editors, *proceedings of 16th International Conference on Implementation and Application of Automata*, number 6807 in LNCS, pages 118–129. Springer, 2011.

[D8] Denis Maurel and Jan Daciuk. Les transducteurs a sorties variables. In Piet Mertens, Cedrick Fairon, Anne Dister, and Patrick Watrin, editors, *Actes de la 13ème conference annuelle sur le traitement des langues naturelles*, volume 1, pages 237–245, Louvain, Belgium, 10–13 April 2006. UCL Presses Universitaires de Louvain.

5 Omówienie pozostałych osiągnięć naukowo — badawczych

W projekcie ISPAD (Inteligentny system pozyskiwania i analizy danych z serwisów społecznościowych w Internecie finansowany ze środków na naukę w latach 2009–2011 przez MNiSW i NCBR) zrealizowanym wspólnie z gdyńska firmą Fido Intelligence współpracowałem w charakterze głównego wykonawcy przy opracowywaniu części słownikowej i płytkiej analizy składniowej do systemem wykrywania określonych treści w mediach społecznościowych prowadząc

szkolenia pracowników. Koordynowałem także pracami zespołu pracowników mojej katedry w projekcie.

Obecnie współpracuję z firmą Voice Lab zajmującą się opracowywaniem systemów automatycznego rozpoznawania mowy. Współpraca ta już przyniosła wymierne wyniki w postaci znacznego zwiększenia rozpoznawanego słownictwa, które systematycznie firma wdraża w swoich produktach.

Byłem kilkukrotnie zapraszany na gościnne wykłady.

- Trzy wykłady zaproszone w instytucie Seminar für Sprachwissenschaft Uniwersytetu Tübingen w Niemczech: dwa w roku 2000 i jeden w 2006.
- Cztery wykłady zaproszone jako *visiting professor* na Uniwersytecie François Rabelais w Tours we Francji w 2004 roku.
- Jeden wykład zaproszony na uniwersytecie Alicante w Hiszpanii w 2004 roku.
- Jedno wystąpienie na seminarium z cyklu „Przetwarzanie Języka Naturalnego" w Instytucie Podstaw Informatyki PAN w 2001 roku.
- Jeden wykład zaproszony (plenarny) na konferencji HLT Days 2012 w Warszawie skupiającej ośrodki zajmujące się przetwarzaniem języka naturalnego w Polsce.

Wykonałem bardzo wielu recenzji dla czasopism międzynarodowych, w tym dla:

- *Computational Linguistics*,
- *Theoretical Computer Science*,
- *Software: Practice & Experience*,
- *Applied Mathematics and Information Sciences*,
- *Journal of Language and Computation*,
- *Journal of Universal Computer Science*,
- *Journal of Logic and Computation*,
- *Computational Linguistics — Applications*,
- *Computing and Informatics*,
- *Theory of Computing Systems*,
- *Zentralblatt für Math*,
- *International Journal of Applied Mathematics and Computer Science*,
- *Journal of Natural Language Engineering*,

- *Information and Computation.*
- *South African Computer Journal*

Jestem członkiem komitetu programowego czasopisma *Journal of Language Modeling.*

Jestem członkiem komitetów programowych cyklów konferencji:

- *Finite State Methods in Natural Language Processing* od 2005 roku,
- *Conference on Implementation and Applications of Automata* w 2012 i w 2017 roku,
- *Intelligent Information Systems* (dla sesji poświęconych przetwarzaniu języka naturalnego).

Recenzowałem także artykuły w następujących cyklach konferencji:

- *Descriptional Complexity of Formal Systems* i
- *Empirical Methods in Natural Language Processing.*

Na konferencjach CIAA i FSMNLP wielokrotnie przewodniczyłem sesjom.

Recenzowałem wnioski o finansowanie projektów naukowych zgłoszonych do National Science Foundation w Republice Południowej Afryki. Recenzowałem wniosek o przedłużenie zatrudnienia naukowca w Ruđer Bošković Institute w Zagrzebiu w Chorwacji.

Byłem recenzentem dwóch rozpraw doktorskich w Hiszpanii:

- Muntsa Padró Cirera, *Applying Causal-State Splitting Reconstruction Algorithm to Natural Language Processing Tasks* obronionej w 2008 r. na Universitat Politècnica de Catalunya,
- Juan Otero, *Análisis lexico robusto* obronionej w 2009 r. na uniwersytecie w Vigo.

Stworzona i utrzymywana przeze mnie strona internetowa poświęcona algorytmom tworzenia i wykorzystywania automatów skończonych:
`http://www.eti.pg.gda.pl/~jandac/fsm_algorithms/`
jest popularna w środowiskach naukowych zajmujących się automatami skończonymi.

Ponadto na konferencji CIAA 2012 w Tours mój artykuł napisany wspólnie z Dawidem Weissem (Apache Software Foundation) uzyskał nagrodę Best Paper Award.

5.1 Wskaźniki bibliometryczne według Web of Science

liczba cytowań	80.
liczba cytowań bez cytowań własnych	62.
cytowania w artykułach	65.
cytowania w artykułach bez cytowań własnych	54.
średnia liczba cytowań na publikację	5.
indeks Hirscha	3.
Sumaryczny Impact Factor	6.191
Sumaryczny Impact Factor po doktoracie	4.515

6 Podsumowanie

Moje osiągnięcia naukowe obejmują opracowanie 8 nowych algorytmów tworzenia automatów, znaczące udoskonalenie (redukcja pesymistycznej złożoności czasowej o dwa rzędy) algorytmu przyrostowej minimalizacji automatów skończonych, opracowanie, wdrożenie i porównanie wybranych metod reprezentacji deterministycznych automatów skończonych, opracowanie i wdrożenie kilku realizacji funkcji mieszającej (w tym dla automatów drzewiastych), zaprojektowanie i wdrożenie biblioteki funkcji do wykorzystywania automatów skończonych jako słowników z realizacją zwartych modeli języka oraz znaczące rozbudowanie dwóch pakietów programów do budowy i wykorzystywania do przetwarzania języka naturalnego automatów skończonych i automatów Mealy'ego.

Opracowałem 8 nowych algorytmów tworzenia automatów wykorzystywanych do tworzenia różnego rodzaju słowników głównie dla potrzeb przetwarzania języka naturalnego i rozpoznawania mowy. Wszystkie te algorytmy charakteryzują się małymi wymaganiami pamięciowymi przy zachowaniu dużej prędkości działania, a wybór algorytmu zależy od rodzaju i charakterystyki danych wejściowych, rodzaju i przeznaczenia słownika oraz gotowości użytkownika do dodatkowego zwiększenia prędkości kosztem zajętości pamięci lub odwrotnie. Ograniczanie zajętości pamięci w czasie konstrukcji automatu ma nadal duże znaczenie praktyczne mimo gwałtownego wzrostu pojemności dostępnych współcześnie pamięci operacyjnych, ponieważ równie znacząco rośnie zapotrzebowanie, a zatem i rozmiar danych przechowywanych w słownikach.

Opracowałem także różne metody realizacji funkcji mieszającej, wykorzystywanej w automatach do indeksowania informacji przechowywanej poza automatami. Metody te obejmują dynamiczną funkcję mieszającą, która umożliwia zachowanie niezmiennych numerów (indeksów) słów przy zmieniającym się słowniku (przy dodawaniu i usuwaniu słów w trakcie użytkowania słownika). Obejmują też opracowanie funkcji mieszającej (minimalnej doskonałej oraz dowolnej) z użyciem wstępujących, deterministycznych automatów drzewiastych i odwrotność minimalnej, doskonałej funkcji mieszającej. Jest to pierwsze takie rozwiązanie na świecie.

Opracowałem, wdrożyłem i porównałem różne metody kompresji automatów. Razem z realizacją funkcji mieszającej posłużyły mi do także do wdrożenia bardzo oszczędnych pamięciowo modeli języka.

Znacząco ulepszyłem algorytm przyrostowej minimalizacji dowolnych deterministycznych automatów skończonych zmniejszając jego złożoność czasową o dwa rzędy wielkości, co oznacza, że algorytm może być wykorzystywany w praktyce dla wszystkich automatów, a co nie było możliwe w jego oryginalnej wersji.

Wyniki badań prezentowałem na konferencjach międzynarodowych i publikowałem w czasopismach, w tym w czasopismach indeksowanych przez JCR — por. wskaźniki bibliometryczne wymienione w poprzednim punkcie. Współpracowałem i nadal współpracuję z wieloma zespołami przy realizacji projektów badawczych finansowanych ze środków prywatnych i publicznych zarówno w kraju, jak i zagranicą.

Byłem wielokrotnie zapraszany do wygłaszania gościnnych wykładów na różnych zagranicznych uczelniach. Przewodniczyłem sesjom na międzynarodowych konferencjach.

Jestem członkiem komitetu programowego międzynarodowego czasopisma naukowego i członkiem programowym serii konferencji międzynarodowych. Recenzowałem artykuły dla różnych konferencji i dla wielu międzynarodowych czasopism, w tym wielu z bazy JCR. Recenzowałem

dwie prace doktorskie z Hiszpanii.

W ujęciu syntetycznym mojego dorobku względem odpowiedniego zapisu w rozporządzeniu Ministra Nauki i Szkolnictwa Wyższego z dnia 3 października 2014 w sprawie szczegółowego trybu i warunków przeprowadzania czynności w przewodzie doktorskim, w postępowaniu habilitacyjnym oraz w postępowaniu o nadaniu tytułu profesora, Dziennik Ustaw Rzeczypospolitej Polskiej, mój dorobek przedstawia się następująco.

Jestem autorem i współautorem publikacji naukowych w czasopismach znajdujących się w bazie Journal Citation Reports (JSR) (§ 3, ust. 4a rozporządzenia ministra), a także autorem zrealizowanego oryginalnego osiągnięcia technologicznego pod postacią biblioteki funkcji (§ 3, ust. 4b). Jestem autorem monografii oraz autorem i współautorem publikacji w czasopismach międzynarodowych innych niż wymienionych w § 3 (§ 4 ust. 1). Jestem autorem i współautorem rozdziałów w książkach o zasięgu międzynarodowym opublikowanych w renomowanych wydawnictwach (§ 4, ust. 2). Wygłosiłem wiele referatów na konferencjach międzynarodowych (§ 4, ust. 8). Uczestniczyłem w zagranicznych i krajowych programach badawczych (§ 5, ust. 1). Uczestniczyłem w międzynarodowych konferencjach naukowych (§ 5, ust. 2). Jestem członkiem rady programowej czasopisma międzynarodowego (§ 5, ust. 6). Odbyłem staże w zagranicznych ośrodkach akademickich (§ 5, ust. 11). Recenzowałem znaczną liczbę publikacji w wielu czasopismach międzynarodowych (§ 5, ust. 14).

Na zakończenie pragnę podkreślić, że badania nad automatami skończonymi rozpocząłem jeszcze przed uzyskaniem doktoratu, ale w toku mojej działalności naukowej znacznie je rozszerzyłem na automaty cykliczne, automaty pseudominimalne i wstępujące automaty drzewiaste. Moim podstawowym dążeniem było stworzenie zespołu metod i algorytmów pozwalających na uzyskiwanie najlepszych wyników dla dowolnych danych i przy zróżnicowanych wymaganiach użytkownika.