załącznik 2

AUTOREFERAT

1. Imię i nazwisko. Adam Roman

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

2006 dr nauk matematycznych w dyscyplinie informatyka, Uniwersytet Jagielloński
tytuł pracy: *Problemy synchronizacji automatów skończonych*
2003 mgr informatyki, Uniwersytet Jagielloński (dyplom z wyróżnieniem)
tytuł pracy: *Algorytmy na słowach zabronionych w shiftach skończonego typu*

3. Informacje o dotychczasowym zatrudnieniu w jednostkach naukowych.

od 2008 adiunkt w Instytucie Informatyki, Uniwersytet Jagielloński
2006-2008 asystent w Instytucie Informatyki, Uniwersytet Jagielloński
2006-2007 adiunkt w Instytucie Matematyki, Akademia Świętokrzyska

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

a) tytuł osiągnięcia naukowego

Synchronizacja i Problem Kolorowania Drogi – algorytmy, złożoność obliczeniowa i uogólnienia

b) spis publikacji wchodzących w skład osiągnięcia naukowego

[H11] V. Vorel, A. Roman, Parameterized Complexity of Synchronization and Road Coloring, **Discrete Mathematics and Theoretical Computer Science** 17:1 (2015), 307–330

[H10] A. Roman, M. Drewienkowski, A complete solution to the complexity of Synchronizing Road Coloring for non-binary alphabets, **Information and Computation** 242 (2015), 383–393

[H9] V. Vorel, Adam Roman, Complexity of Road Coloring with Prescribed Reset Words **Lecture Notes in Computer Science** 8977 (2015), 161–172

[H8] A. Roman, I. Podolak, D. Jędrzejczyk, Application of Hierarchical Classifier to Synchronizing Problem, W: Rutkowski L. et al. (Eds.), Artificial Intelligence and Soft Computing, **Lecture Notes in Computer Science** 7267 (2012), 421–429

[H7] A. Roman, P–NP threshold for the Synchronizing Road Coloring, W: Dediu, A.H., Martín-Vide C. (Eds.) Language and Automata Theory and Applications, **Lecture Notes in Computer Science** 7183 (2012), 482–492

[H6] R. Kudłacik, A. Roman, H. Wagner, Effective synchronizing algorithms, **Expert Systems with Applications** 39 (2012), 11746–11757

[H5] A. Roman, NP-completeness of the Road Coloring Problem, **Information Processing Letters** 111 (2011), 342–347

[H4] K. Chmiel, A. Roman, COMPAS – a computing package for synchronization, W: Domaratzki M., Salomaa K. (Eds.), Implementation and Application of Automata, **Lecture Notes in Computer Science** 6482 (2011), 79–86

[H3] A. Roman, Decision Version of the Road Coloring Problem is NP-complete, W: Kutyłowski M., Charatonik W., Gębala M. (Eds.), Fundamentals of Computation Theory, **Lecture Notes in Computer Science** 5699 (2009) 287–297

[H2] A. Roman, Genetic algorithm for synchronization, W: Dediu A.H. et al. (Eds.), Language and Automata Theory and Applications, **Lecture Notes in Computer Science** 5457 (2009), 684–695

[H1] A. Roman, W. Foryś, Lower bound for the length of synchronizing words in partially-synchronizing automata, W: Geffert V. et al. (Eds.) SOFSEM 2008: Theory and Practice of Computer Science, **Lecture Notes in Computer Science** 4910 (2008), 448–459

Prace powiązane z rozprawą habilitacyjną[1]:

[P7] A. Roman, Experiments on synchronizing automata, **Schedae Informaticae** 19 (2010), 35–51

[P6] A. Roman, A note on Černý Conjecture for automata over 3-letter alphabet, **Journal of Automata, Languages and Combinatorics** 13(2) (2008) 141–143

[P5] A. Roman, Synchronizing Finite Automaton with Short Reset Words, **Applied Mathematics and Computation** 209(1) (2009), 125–136

[P4] A. Roman, Two simple methods for obtaining new classes of automata fulfilling Černý Conjecture, **Schedae Informaticae** 16 (2007), 35–46

[P3] A. Roman, Merging states and synchronization problem, **Schedae Informaticae** 15 (2006), 95–108

[P2] A. Roman, Synchronizing Finite Automaton with Short Reset Words,**Lecture Series on Computer and Computational Sciences** 4 (2005), 492–495

[P1] A. Roman, New Algorithms for Finding Short Reset Sequences in Synchronizing Automata, **Enformatika** 7 (2005), 13–17

c) omówienie celu naukowego/artystycznego ww. pracy/prac i osiągniętych wyników wraz z omówieniem ich ewentualnego wykorzystania

Trzy główne obszary moich zainteresowań naukowych to: synchronizacja automatów skończonych (w szczególności tzw. hipoteza Černego [Černý 1964] oraz Problem Kolorowania Drogi [Adler 1977]), zapewnianie jakości oprogramowania (techniki projektowania testów, modele jakości, modele wzrostu niezawodności, analiza mutacyjna, doskonalenie procesu testowego) oraz nauczanie maszynowe (problemy klasyfikacji). W niniejszym rozdziale skupię się na omówieniu wyników wchodzących w skład rozprawy habilitacyjnej, tj. dotyczących synchronizacji. Publikacje związane z pozostałymi obszarami badań omówione są w rozdz. 5, a projekty dotyczące współpracy z przemysłem IT w rozdz. 6.

[1] Prac [P1]–[P7] nie włączyłem do rozprawy, ponieważ zawierają wyniki przyczynkowe lub opisują wyniki uzyskane w doktoracie. Niemniej jednak, ze względu na powiązanie tematyczne z rozprawą i stanowienie wraz z nią pewnej spójnej całości, w rozdziale 4 omawiam również wyniki niektórych z tych prac.

Synchronizacja automatów skończonych

Automatem skończonym nazywamy trójkę $\boldsymbol{A} = (Q, A, \delta)$, gdzie Q jest niepustym, skończonym zbiorem stanów, A niepustym, skończonym zbiorem zwanym alfabetem, a $\delta: Q \times A \to Q$ to funkcja przejścia, którą zwyczajowo rozszerza się w naturalny sposób na $Q \times A^* \to Q$. Nie wyróżniamy stanów początkowych ani końcowych. O ile nie jest zaznaczone inaczej, zakładamy, że wszystkie rozważane przez nas automaty są deterministyczne, to znaczy, że funkcja δ jest funkcją zupełną, czyli jest dobrze określona dla dowolnego elementu z $Q \times A$.

Automat $\boldsymbol{A} = (Q, A, \delta)$ jest synchronizujący, jeśli spełniony jest następujący warunek:

$$(\exists w \in A^*)\ (\forall p, q \in Q)\ \delta(p, w) = \delta(q, w). \qquad (1)$$

Równoważnie można powiedzieć, że automat $\boldsymbol{A} = (Q, A, \delta)$ jest synchronizujący, jeśli $|\delta(Q, w)| = 1$. Oznacza to, że niezależnie od tego, w którym stanie aktualnie się znajdujemy, pod wpływem słowa w przejdziemy zawsze do jednego, ustalonego stanu. Nie każdy automat posiada tę własność – w szczególności nie są synchronizujące wszystkie te automaty, dla których graf wyznaczony funkcją przejść jest niespójny. Słowo w dla którego zachodzi własność (1) nazywane jest słowem synchronizującym. Czasami będziemy również używać pojęcia "sekwencja synchronizująca". Jeśli automat posiada słowo synchronizujące w o długości $|w| = k$, to łatwo pokazać, że posiada także słowo synchronizujące o dowolnej długości $l \geq k$. Aby takowe skonstruować, wystarczy wziąć słowo wv, gdzie $v \in A^*$ jest dowolnym słowem o długości $l - k$. Ta prosta obserwacja od razu skłania do zadania naturalnego pytania o *najkrótsze* możliwe słowo synchronizujące dla zadanego automatu. Takie słowa nazywać będziemy minimalnymi słowami synchronizującymi (MSS).

Hipoteza Černego [Černý 1964] związana jest z ograniczeniem górnym na długość MSS. Głosi ona, że dla każdego n-stanowego automatu synchronizującego długość jego MSS wynosi co najwyżej $(n-1)^2$. Najlepszym znanym do tej pory ograniczeniem górnym jest $(n^3 - n)/6$ [Pin 1983]. Jak widać, istnieje duża różnica pomiędzy wartością z hipotezy (rzędu $O(n^2)$) a znanym ograniczeniem (rzędu $O(n^3)$). Chociaż pokazano, że hipoteza Černego jest prawdziwa dla pewnych klas automatów (np. [Ananichev 2004], [Černý 1971], [Dubuc 1998], [Eppstein 1990], [Kari 2003], [Rystsov 1992], [Trahtman 2004]), to jednak w ogólnym przypadku wciąż pozostaje problemem otwartym. Wiadomo, że dla każdego $n \geq 1$ istnieje automat (zwany automatem Černego), który posiada MSS o długości równej dokładnie $(n-1)^2$. Ograniczenie z hipotezy jest więc osiągalne dla dowolnego n. Jeśli hipoteza jest fałszywa, to musi istnieć n-stanowy automat synchronizujący posiadający MSS o długości większej niż $(n-1)^2$.

Zastosowania teorii synchronizacji

Problem znajdowania MSS ma wiele bardzo praktycznych zastosowań, na przykład:

- w przemyśle (projektowanie tzw. orienterów części [Ananichev 2004]),
- w bioinformatyce (problem resetowania biokomputera [Benenson 2003]),
- w teorii kodów (kody synchronizujące [Jürgensen 2008], [Biskup 2008]),
- w korekcji błędów w automatach oraz w tzw. problemie identyfikacji lidera w sieciach ([Eppstein 2002], [Kari 2002]).

Jednak z punktu widzenia informatyki najważniejsze zastosowanie MSS dotyczy testowania oprogramowania, a dokładniej testowania systemów reaktywnych opartego na modelu (tzw. *model-based conformance testing*). W szczególności maszyna stanowa (automat skończenie stanowy lub tzw. automat z wyjściem, np. automat Mealy'ego lub Moore'a), w której poszukiwane jest możliwie krótkie słowo synchronizujące może być modelem testowanego obwodu elektronicznego lub protokołu komunikacyjnego ([Broy 2005], [Fukada 2001], [Pomeranz 1998],

[Zhao 2010]). Słowa synchronizujące oraz ich odmiany (np. tzw. *homing sequences*, *distinguihing sequences*, *UIO sequences* itp.) wykorzystywane są w trzech zasadniczych obszarach testowania systemów reaktywnych. Są to: identyfikacja stanu (mając dany model należy zidentyfikować w jakim stanie system się znajduje), weryfikacja stanu (sprawdzenie, czy system rzeczywiście znajduje się w danym stanie) oraz testowanie zgodności (mając system i jego model należy sprawdzić, czy system działa zgodnie ze specyfikacją).

W zastosowaniach przemysłowych może zachodzić potrzeba testowania dziesiątek lub setek tysięcy układów elektronicznych z użyciem sekwencji synchronizującej. Co więcej, w procesie testowania pojedynczego układu słowo synchronizujące może zostać wykorzystane wielokrotnie w każdym teście. Ponieważ testowanie każdego egzemplarza układu zajmuje pewien czas, kluczowym problemem staje się znalezienie możliwie niedługiego słowa synchronizującego tak, by całkowity czas testowania był jak najkrótszy.

Algorytmy znajdowania słów synchronizujących

Ze względu na wspomniane powyżej zastosowania, w szczególności związane z testowaniem systemów reaktywnych, istnieje potrzeba szybkiego, efektywnego znajdowania możliwie jak najkrótszych słów synchronizujących. Niestety, Eppstein [Eppstein 1990] pokazał, że w wersji decyzyjnej problem znajdowania słowa synchronizującego (tzn. na wejściu jest automat $\boldsymbol{A}$ i liczba k, a pytamy o to, czy dla $\boldsymbol{A}$ istnieje słowo synchronizujące o długości co najwyżej k) jest NP-zupełny. Niedawno Olschewski i Ummels pokazali, że jeśli ograniczymy się tylko do minimalnych słów synchronizujących, to problem jest DP-zupełny [Olschewski 2010]. Dlatego, zakładając, że $P \neq NP$, nie da się zbudować efektywnego algorytmu znajdującego minimalne słowa synchronizujące. W obszarze algorytmów o wielomianowym czasie działania pozostają nam jedynie algorytmy heurystyczne. W 2010 roku Berlinkov wykazał, że o ile $P \neq NP$, nie istnieje również żaden wielomianowy algorytm aproksymujący długość minimalnego słowa synchronizującego[2] ze stałym czynnikiem, tzn. zwracający jako długość MSS maksymalnie $C \cdot |w|$, gdzie $C \geq 1$ jest stałą a w – minimalnym słowem synchronizującym [Berlinkov 2010].

Pierwszym wielomianowym algorytmem znajdującym dla zadanego automatu możliwie krótkie słowo synchronizujące był algorytm skonstruowany przez Natarajana [Natarajan 1986], działający w czasie $O(n^4 + |A|n^2)$, gdzie n oznacza liczbę stanów automatu. Eppstein [Eppstein 1990], dokonując prostej modyfikacji algorytmu Natarajana związanej z wykonaniem pewnych obliczeń wstępnych związanych z funkcją przejścia, uzyskał algorytm działający w czasie $O(n^3 + |A|n^2)$. Jest to najpopularniejszy zachłanny algorytm synchronizacji. Działa on w następujący sposób: przy pomocy tzw. automatu par znajduje on dwa stany p, q, które synchronizują się do pewnego wspólnego stanu za pomocą możliwie najkrótszego słowa w_1. Przekształcamy Q przez słowo w_1 i z tak uzyskanego zbioru (o mocy co najwyżej $|Q| - 1$) znów wybieramy parę stanów o powyższej własności. Załóżmy, że para ta synchronizowana jest słowem w_2. Przekształcamy aktualny zbiór stanów $\delta(Q, w_1)$ przez słowo w_2 i powtarzamy procedurę, dopóki nie znajdziemy się w zbiorze zawierającym tylko jeden stan. Jeśli automat jest synchronizujący, dokonamy tego w co najwyżej $n - 1$ krokach. Algorytm zwraca słowo synchronizujące $w_1 w_2 \dots w_k$, gdzie k jest liczbą wykonanych kroków lub – w przypadku, gdy nie jest w stanie przekształcić całego zbioru Q do zbioru jednoelementowego – stwierdza, że automat nie jest synchronizujący.

Cykl prac [H6], [P1], [P2] oraz [P5] poświęcony jest problemowi budowy oraz badania efektywności nowych, wielomianowych algorytmów synchronizujących. Prace [P1] i [P2] są wersjami konferencyjnymi [P5]. W pracy [P5] przedstawiono 3 wersje wielomianowego algorytmu znajdującego, dla zadanego automatu, słowo synchronizujące w czasie[3] $t(n) \approx (41/320)n^5 + O(|A|n^2)$, czyli w $O(n^5 + |A|n^2)$. Algorytm jest o rząd wielkości gorszy od algorytmu Natarajana, ale

[2] Wynik ten dotyczy automatów nad alfabetem o 3 lub więcej literach.
[3] Jest to złożoność pesymistyczna.

daje o wiele lepsze (w sensie długości znalezionego słowa) wyniki niż algorytmy Natarajana i Eppsteina. Mówiąc obrazowo, jeśli algorytm Eppsteina wybiera w każdym kroku parę "najlepszych" lokalnie stanów patrząc "krok naprzód", to algorytm z pracy [P5] patrzy "dwa kroki naprzód", analizując nie tylko to, jak krótkie jest słowo synchronizujące daną parę stanów, ale również jak będzie wyglądał cały aktualny zbiór stanów po przekształceniu go przez słowo odpowiadające tej parze.

Algorytm daje bardzo dobre wyniki w sensie długości znajdowanych słów synchronizujących. Na przykład dla wszystkich możliwych 6-stanowych automatów nad binarnym alfabetem w 89% przypadków zwraca MSS, podczas gdy algorytm Eppsteina tylko w 51%. Jednak jego złożoność czasowa jest w oczywisty sposób niezadawalająca. Algorytm jest niepraktyczny dla automatów o dużej liczbie stanów. Próbą zaradzenia temu problemowi jest praca [H6], w której zmodyfikowano algorytm z [P5] tak, że udało się uzyskać złożoność czasową o rząd wielkości lepszą, a więc taką, jaką ma algorytm Natarajana, zachowując wciąż dobrą jakość algorytmu (optymalne działanie w 82% przypadków dla 6-stanowych automatów nad binarnym alfabetem). Przeprowadzone eksperymenty pokazały również dobrą efektywność czasową algorytmu, pomimo pesymistycznej złożoności rzędu $O(n^4)$.

Drugim istotnym wynikiem pracy [H6] jest pokazanie, że zastosowanie pewnych specyficznych struktur danych do reprezentacji automatu, funkcji przejścia i jej obliczania pozwala na znaczne zredukowanie czasu działania obliczeń w przypadku dokładnego, wykładniczego algorytmu znajdującego MSS. Wykładniczy algorytm znajduje minimalne słowo synchronizujące poprzez przeglądanie skierowanego grafu podzbiorów zbioru stanów (tzw. *power-set automaton*). Każdy wierzchołek grafu reprezentuje pewien podzbiór $P \subseteq Q$ zbioru wszystkich stanów, a krawędź $P \to R$, gdzie $P, R \subseteq Q$ występuje wtedy i tylko wtedy, gdy $\exists\, a \in A\colon \delta(P, a) = R$. Algorytm rozpoczyna działanie w wierzchołku Q i przegląda graf wszerz. W momencie znalezienia pierwszego wierzchołka, który reprezentuje podzbiór jednoelementowy, działanie algorytmu kończy się. Konkatenacja etykiet liter odpowiadających krawędziom tworzącym ścieżkę od Q do tego wierzchołka jest minimalnym słowem synchronizującym. Algorytm ten jest zaimplementowany m.in. w popularnym pakiecie obliczeniowym TESTAS [Trahtman 2003], który pozwala m.in. obliczać słowa synchronizujące za pomocą kilku predefiniowanych metod.

Ponieważ algorytm dokładny działa w czasie wykładniczym, niezwykle istotne jest, aby zoptymalizować jego działanie poprzez wykorzystanie efektywnych struktur danych. Czas działania algorytmu zależy w istotny sposób od takich operacji jak: obliczanie funkcji przejścia, reprezentacja stanu, implementacja kolejki, implementacja mechanizmu zaznaczania odwiedzonych stanów, reprezentacja drzewa przeszukiwań BFS. W pracy [H6] zbadano wpływ na czas działania algorytmu takich struktur danych jak: drzewa AVL i drzewa czerwono-czarne do reprezentacji zbiorów, wektory bitowe czy indeksowane wektory bitowe. Okazało się, że zastosowanie powyższych rozwiązań znacznie skraca czas obliczeń. Na przykład, dla 14-stanowego automatu Černego różne wersje zaproponowanego w [H6] algorytmu wykładniczego dawały wyniki od 6 do 1800 razy lepsze od algorytmu zaimplementowanego w pakiecie TESTAS. Algorytm z pakietu TESTAS ukończył pracę w czasie 18s, różne warianty algorytmu z [H6] działały w czasie od 3.3s do 0.01s. Dla 18-stanowego automatu Černego TESTAS nie zdołał ukończyć obliczeń, natomiast najlepszy z wariantów algorytmu z [H6] ukończył obliczenia w czasie 0.2s. Niedawno Kisielewicz, Kowalski i Szykuła [Kisielewicz 2013] skonstruowali bardzo efektywny algorytm wykładniczy wykorzystując dwukierunkowy BFS oraz tzw. drzewa radix do przechowywania i szybkiego porównywania podzbiorów zbioru stanów.

Algorytm genetyczny dla synchronizacji

Ponieważ znajdowanie MSS jest problemem trudnym, nasuwającym się naturalnym rozwiązaniem jest wykorzystanie metod sztucznej inteligencji. W pracy [H2] zaprojektowano algorytm genetyczny znajdujący możliwie krótkie słowa synchronizujące dla zadanego automatu. Algorytm

genetyczny operuje na populacjach osobników (tzw. chromosomów), z których każdy reprezentuje zakodowaną postać rozwiązania problemu. Na populacjach dokonywane są operacje genetyczne, takie jak krzyżowanie, mutacja czy selekcja, które naśladują darwinowski dobór naturalny. Każdy osobnik jest oceniany poprzez zadaną z góry funkcję przystosowania. Selekcja do następnej populacji dokonywana jest zazwyczaj w taki sposób, że prawdopodobieństwo znalezienia się osobnika w nowej populacji jest proporcjonalne do udziału jego przystosowania w sumie przystosowań wszystkich osobników populacji. Największą szansę przejścia do kolejnej populacji mają zatem osobniki reprezentujące najlepsze znalezione rozwiązania. Mutacja i krzyżowanie mają za zadanie wprowadzić zróżnicowanie genetyczne w populacji, dzięki czemu algorytm nie utyka w ekstremum lokalnym, ale ma zdolność do efektywnego przeszukiwania całej przestrzeni rozwiązań.

Algorytm genetyczny nadaje się niemalże wprost do zastosowania w problemie znajdowania MSS. Chromosom nie musi nawet kodować rozwiązania problemu – on *jest* tym rozwiązaniem, gdyż stanowi ciąg znaków reprezentujących poszczególne litery słowa synchronizującego. Algorytm jest modyfikacją tzw. prostego algorytmu genetycznego (ang. *SGA – simple genetic algorithm*). Modyfikacja była konieczna z trzech powodów: po pierwsze, w SGA chromosomy mają stałą długość, a tutaj siłą rzeczy muszą mieć długość zmienną. Po drugie, w SGA prawdopodobieństwa wykonania operatorów genetycznych mutacji i krzyżowania są stałe. W algorytmie [H2] zmieniają się w zależności od jego działania. Po trzecie, SGA operuje na alfabecie binarnym (istnieją tylko 2 rodzaje genów: 0 i 1), zaś alfabet automatu synchronizującego niekoniecznie musi być binarny. Oczywiście każdy alfabet można zakodować w alfabecie binarnym, jednak, jak wspomniano powyżej, w algorytmie genetycznym chromosom reprezentuje rozwiązanie w sposób bezpośredni. Pozwala to uniknąć czynności związanych z kodowaniem i dekodowaniem chromosomów na rozwiązania, a tym samym potencjalnych problemów związanych z możliwością wygenerowania niepoprawnego rozwiązania w wyniku źle zaprojektowanego operatora genetycznego. W algorytmie [H2] wykorzystano również pewne metody probabilistyczne dla generowania rozkładu prawdopodobieństwa zadanego na alfabecie. Rozkład ten wykorzystywany jest w operatorze mutacji. Wyniki działania algorytmu genetycznego są bardzo obiecujące. Z przeprowadzonych doświadczeń wynika np., że dla 16-stanowego automatu Černego algorytm znajdował słowa jedynie ok. 1.16 razy dłuższe od MSS.

Automaty ekstremalne

Eksperymenty numeryczne pokazują, że średnia długość MSS dla losowego n-stanowego automatu synchronizującego nad alfabetem binarnym jest liniowa lub subliniowa [Skvortsov 2011]. W pracy [Skvortsov2010] Skvortsov i Zaks pokazali, że dla losowego n-stanowego automatu synchronizującego o mocy alfabetu $m(n) > 36 \ln n$ automat ten jest z prawdopodobieństwem 1 synchronizowany słowem o długości logarytmicznej względem liczby stanów. Pokazali oni ponadto, że dla $m(n) > n^{\beta}$, $\beta > 0.5$, losowy automat synchronizujący spełnia hipotezę Černego z dużym prawdopodobieństwem. Jedną z trudności w badaniach nad hipotezą Černego jest to, że bardzo trudno znaleźć automaty, dla których długość MSS osiąga postawioną w hipotezie wartość $(n-1)^2$ (takie automaty nazywać będziemy ekstremalnymi) lub jest przynajmniej kwadratowa względem n. Jedyną znaną nieskończoną serią automatów ekstremalnych jest seria opisanych wcześniej automatów Černego. Oprócz niej znaleziono do tej pory jedynie 8 innych automatów ekstremalnych.

Jeden z tych ośmiu automatów, oznaczany jako $\boldsymbol{A}_5$, jest opisany w pracy [P6]. Jego istnienie oraz własności potwierdzone zostały później w badaniach numerycznych przeprowadzonych przez Trahtmana [Trahtman 2006]. Automat posiada 5 stanów oraz trzyelementowy alfabet. Wynik ten, choć na pierwszy rzut oka wydaje się niezbyt interesujący, posiada pewną wartość poznawczą. W zbiorze wszystkich 5-stanowych automatów nad alfabetem binarnym jedynym automatem ekstremalnym jest automat Černego. Przykład $\boldsymbol{A}_5$ pokazuje, że zwiększając moc alfabetu jesteśmy w stanie tak skomplikować strukturę automatu, aby "utrudnić" proces synchronizacji. Sugeruje to, gdzie należy szukać "trudnych" automatów (o długich słowach synchronizujących) lub wręcz

kontrprzykładu na hipotezę Černego: jeśli hipoteza jest fałszywa, to kontrprzykładem będzie prawdopodobnie automat nad dużym alfabetem. Większość dotychczasowych badań prowadzona była dla automatów nad alfabetem binarnym. W szczególności pod tę kategorię podpadają automaty Černego. Przykład $\boldsymbol{A}_5$ sugeruje, że należy przyglądnąć się uważniej automatom ekstremalnym. Posiadają one ciekawe własności, czego przykładem może być inny, 6-stanowy automat ekstremalny znaleziony przez J. Kari [Kari 2001]. Automat ten jest kontrprzykładem na tzw. uogólnioną hipotezę Černego autorstwa J.E. Pina, która głosi, że jeśli n-stanowy automat posiada słowo $w \in A^*$ takie, że $|\delta(Q,w)| = n-k$, to istnieje słowo v o tej samej własności, posiadające długość co najwyżej $|v| = k^2$. Dla $k = n-1$ otrzymujemy hipotezę Černego. Hipoteza Pina jest prawdziwa dla $k = 1,2,3$, ale automat Kari'ego stanowi kontrprzykład dla $k = 4$, gdyż najkrótsze słowo sprowadzające pełny zbiór stanów Q do zbioru dwuelementowego ma długość 17, co jest sprzeczne z zadaną w hipotezie wartością $k^2 = 4^2 = 16$.

Uogólnione pojęcie synchronizacji

Pewien specyficzny przypadek zastosowania automatów synchronizujących w konstrukcji orienterów części doprowadził badaczy (np. [Martjugin 2006]) do nieco ogólniejszej definicji automatu synchronizującego. Mianowicie, dopuszcza się częściową funkcję przejścia, przy czym przejście $\delta(P,a)$ ze zbioru stanów $P \subseteq Q$ pod wpływem $a \in A$ jest możliwe wtedy i tylko wtedy, gdy dla każdego stanu $p \in P$ funkcja $\delta(p,a)$ jest dobrze zdefiniowana. Inni badacze (np. [Ito 2004]) rozważają pojęcie synchronizacji w przypadku *niedeterministycznych* automatów skończonych, w których pod wpływem jednej litery można przejść do więcej niż jednego stanu. W obu tych podejściach bada się dolne ograniczenie na długość MSS aby stwierdzić, czy zmodyfikowane pojęcie synchronizacji ułatwia, czy utrudnia proces synchronizacji.

Praca [H1] wpisuje się w ten właśnie nurt badań. W [H1] pojęcie synchronizacji zostało uogólnione jeszcze bardziej niż w [Martjugin 2006]. Dopuszczamy, by proces synchronizacji rozpoczynał się nie w całym zbiorze Q, ale w pewnym jego podzbiorze $P \subseteq Q$. Intuicja podpowiada, że w takim przypadku słowa synchronizujące generalnie powinny być krótsze, gdyż startujemy z mniejszego podzbioru stanów. W [H1] pokazano jednak, że jest dokładnie na odwrót: o ile w klasycznym problemie synchronizacji znane jest wielomianowe ograniczenie na długość MSS, o tyle w przypadku synchronizacji uogólnionej dolne ograniczenie jest wykładnicze ze względu na liczbę stanów. Wykorzystanie narzucającego się tu twierdzenia Spernera pozwala uzyskać ograniczenie rzędu $\binom{n}{n/2}$. Przeprowadzając w [H1] analizę kombinatoryczną przy zliczaniu pewnych obiektów związanych z podzbiorami zbioru stanów udało się uzyskać ograniczenie rzędu $\binom{n+1}{n/2}$.

Synchronizacja i Problem Kolorowania Drogi – złożoność obliczeniowa

Problem synchronizacji jest silnie związany z tzw. Problemem Kolorowania Drogi (PKD). PKD po raz pierwszy pojawił się w pracy Adlera [Adler 1970], a formalnie zdefiniowany został 7 lat później w [Adler 1977]. PKD można sformułować w następujący sposób: niech G będzie silnie spójnym grafem skierowanym o stałym stopniu wychodzącym wierzchołków takim, że NWD długości wszystkich cykli w G wynosi 1. Grafy o takich własnościach nazywać będziemy w skrócie PKD-grafami. Jakie warunki musi spełniać G, by istniało etykietowanie krawędziowe zmieniające G w automat synchronizujący? Łatwo można pokazać, że względna pierwszość długości wszystkich cykli w grafie jest warunkiem koniecznym. Zasadniczym pytaniem było to, czy jest to również warunek wystarczający. PKD został rozwiązany przez Trahtmana w słynnej pracy [Trahtman 2009] w której dowiódł on, że względna pierwszość długości cykli grafu jest w istocie warunkiem koniecznym i wystarczającym do istnienia takiego etykietowania. Wynik ten otworzył szerokie pole dla badań leżących na pograniczu PKD i synchronizacji. Każdy problem dotyczący synchronizacji można bowiem wyrazić w języku PKD w następujący sposób:

Problem klasyczny

WEJŚCIE: automat $\boldsymbol{A}$
WYJŚCIE: TAK wtw, gdy zachodzi $COND$, gdzie $COND$ jest pewnym warunkiem nałożonym na $\boldsymbol{A}$

Problem w wersji PKD

WEJŚCIE: PKD-graf G
WYJŚCIE: TAK wtw, gdy istnieje etykietowanie krawędziowe G zamieniające ten graf w automat $\boldsymbol{A}$ taki, że zachodzi $COND$

Konsekwencją twierdzenia Trahtmana jest powstanie szeregu problemów z zakresu złożoności obliczeniowej. Jeden z takich problemów został postawiony przez prof. M. Volkova na konferencji "Around the Černý Conjecture" we Wrocławiu w 2008 roku. Volkov zapytał o złożoność obliczeniową następującego problemu: mając na wejściu PKD-graf G sprawdzić, czy istnieje etykietowanie zmieniające G w automat synchronizujący posiadający słowo synchronizujące o długości k.

Wiadomo, że odpowiadający mu problem klasyczny: "mając na wejściu automat $\boldsymbol{A}$ sprawdzić, czy istnieje dla niego słowo synchronizujące o długości k" jest NP-zupełny. Intuicyjnie, problemy w wersji PKD powinny być łatwiejsze od odpowiadających im problemów klasycznych, gdyż mamy swobodę w wyborze etykietowania dla grafu G. Na przykład problem weryfikacji, czy automat jest synchronizujący jest kwadratowy ze względu na liczbę stanów, $O(|A| \cdot n^2)$, podczas gdy weryfikację, czy graf jest PKD-grafem można przeprowadzić w czasie $O(|A| \cdot n)$, czyli liniowym ze względu na liczbę krawędzi[4]. Okazuje się, że wcale tak być nie musi. Cykl prac [H3], [H5], [H7], [H9] i [H10] podejmuje i rozwija problem postawiony przez Volkova. W pokonferencyjnej pracy [H3] udowodniono, że problem ten jest NP-zupełny. W [H5], która jest pełną wersją [H3], pokazano znacznie silniejszy rezultat, mianowicie NP-zupełność następującego problemu, parametryzowanego dwoma liczbami naturalnymi ℓ oraz C:

Problem $SRCP_\ell^C$

WEJŚCIE: PKD-graf G o stopniu wychodzącym wierzchołków $\ell \geq 3$
WYJŚCIE: TAK wtw, gdy istnieje etykietowanie krawędziowe zmieniające G w automat posiadający słowo synchronizujące o długości C

gdzie $C > 7$ jest stałą! W oczywisty sposób odpowiadający temu problemowi problem klasyczny: "mając automat $\boldsymbol{A}$ nad alfabetem co najmniej 3-literowym sprawdzić, czy jest on synchronizowany słowem długości C" jest rozwiązywalny wielomianowo w czasie $O(n \cdot \ell^C)$, gdzie n jest liczbą stanów automatu. Wystarczy bowiem dla każdego ze słów o długości C sprawdzić, czy synchronizuje ono $\boldsymbol{A}$.

Praca [H5] pokazuje nie tylko nietrywialną asymetrię pomiędzy rodziną klasycznych problemów synchronizacyjnych a ich odpowiednikami wyrażonymi w języku PKD, ale też przeczy wcześniej wspomnianej intuicji: pozornie łatwiejszy problem w wersji PKD w istocie okazuje się o wiele trudniejszy od swojego klasycznego odpowiednika.

Wyniki pracy [H5] dotyczą tzw. złożoności parametrycznej, zatem w naturalny sposób pojawia się pytanie o próg P-NP dla rodziny problemów $SRCP_\ell^C$: dla jakich wartości ℓ i C problem jest NP-zupełny, a dla jakich staje się problemem rozwiązywalnym wielomianowo? W pracy [H7] pokazano,

[4] Zob. np. http://www.ces.clemson.edu/~shierd/Shier/markov.pdf

że problem jest NP-zupełny już dla $C \geq 4$ i $\ell \geq 3$, natomiast staje się wielomianowy dla $C \geq 2$. Dla alfabetów co najmniej 3-literowych pozostała luka w przypadku $C = 3$. Została ona zamknięta w pracy [H10], w której udowodniono, że dla $C = 3$ problem da sie rozwiązać w czasie wielomianowym. Wyniki tej pracy zamykają więc cały problem dla niebinarnych alfabetów. O ile techniki dowodzenia NP-zupełności wykorzystane w pracach [H3], [H5] i [H7] są do siebie bardzo podobne, o tyle w pracy [H10] metoda ta się nie sprawdziła. Należało tu zastosować bardziej wyrafinowane podejście. Mówiąc najogólniej, rozważa się problemy $SRCP_\ell^C$ dla ustalonych typów słów:

Problem $SRCP_\ell^C(w)$ WEJŚCIE: PKD-graf G o stopniu wychodzącym wierzchołków $\ell \geq 3$ WYJŚCIE: TAK wtw, gdy istnieje etykietowanie krawędziowe zmieniające G w automat posiadający słowo synchronizujące w

a następnie bada się relacje pomiędzy nimi. Na przykład, dla $C = 3$ i $\ell \geq 3$ rozważono problemy $SRCP_\ell^C(w)$ dla wszystkich możliwych trzyliterowych słów z dokładnością do nazewnictwa liter, a więc dla $w \in \{a^3, a^2b, aba, ab^2, abc\}$. Łatwo pokazać, że jeśli $w \in \{a^3, a^2b, aba\}$, to $SRCP_\ell^C(w)$ jest wielomianowy. Kluczową obserwacją jest fakt, że $SRCP_\ell^C(abc)$ oraz $SRCP_\ell^C(ab^2)$ w ogólności mogą być NP-zupełne, ale jeśli ograniczymy się tylko do tych instancji $SRCP_\ell^C(abc)$ oraz $SRCP_\ell^C(abb)$ dla których wiadomo, że nie istnieje słowo synchronizujące postaci aba ani a^3, to takie wersje $SRCP_\ell^C(abc)$ oraz $SRCP_\ell^C(ab^2)$ da się rozwiązać w czasie wielomianowym. Z powyższych faktów w prosty sposób wynika konstrukcja wielomianowego algorytmu rozstrzygającego problem $SRCP_\ell^3$. Jeśli bowiem przez G_w oznaczymy wszystkie instancje $SRCP_\ell^3(w)$, to zachodzi

$$SRCP_\ell^3 = G_{a^3} \cup (G_{a^2b} \setminus G_{a^3}) \cup (G_{aba} \setminus G_{a^3}) \cup (G_{abc} \setminus G_{a^3} \setminus G_{aba}) \cup (G_{abb} \setminus G_{a^3} \setminus G_{abc}),$$

i każdy ze składników sumy reprezentuje problem rozwiązywalny w czasie wielomianowym.

Konferencyjna praca [H9] stanowi kontynuację badań nad synchronizacją i Problemem Kolorowania Drogi w sytuacji, gdy słowa synchronizujące zadane są z góry. Scharakteryzowano słowa nad alfabetem binarnym, dla których problem $SRCP_2(w)$ jest NP-zupełny (tutaj rozważamy problem dla słów o dowolnej długości, więc nie jest on parametryzowany stałą C). W przypadku, gdy nie zakładamy silnej spójności grafów analizowanych automatów, własność ta zachodzi dla wszystkich słów poza $w = a^k b$ oraz $w = a^k$ dla $k \geq 1$, z dokładnością do nazewnictwa liter. Interesującym wynikiem uzyskanym w pracy jest fakt, że przyjęcie dosyć naturalnego założenia o silnej spójności grafu zmienia nieco wynik: w przypadku $w = abb$ problem staje się wielomianowy.

W 2013 roku Fernau, Heggernes i Villanger [Fernau 2013] przeprowadzili analizę multiwariacyjną złożoności obliczeniowej (ang. *multivariate complexity analysis*) dla dwóch problemów decyzyjnych dotyczących automatów skończonych. Analiza ta, przeprowadzana dla problemów trudnych obliczeniowo, bada przyczyny tej trudności poprzez systematyczną analizę tzw. naturalnych parametrów opisujących problem. W przypadku automatów skończonych takimi naturalnymi parametrami mogą być: liczba stanów, wielkość alfabetu, długość słowa czy postać słowa. W analizie multiwariacyjnej wykorzystuje się ważne pojęcie tzw. jądra (ang. *kernel*). Jest to wielomianowa transformacja przekształcająca dowolną daną, sparametryzowaną instancję problemu w równoważną instancję, o rozmiarze i parametrze ograniczonymi przez funkcję parametru wejściowego. Jądro jest wielomianowe, jeśli rozmiar i parametr wyjścia są wielomianowo ograniczone przez parametr wejścia.

Autorzy pracy [Fernau 2013] postawili pytanie, czy problem synchronizacji posiada jądro wielomianowe. W pracy [H11] rozwiązano ten problem udzielając odpowiedzi negatywnej: problem, parametryzowany liczbą stanów, nie posiada jądra wielomianowego (przy założeniu, że

hierarchia wielomianowa nie ulega kolapsowi). Ponadto, w [H11] przeprowadzono pełną analizę multiwariacyjną dla problemu $SRCP$ pokazując, że problem ten, parametryzowany liczbą stanów, posiada jądro wielomianowe i zamykając w ten sposób wcześniejsze badania dotyczące wpływu na złożoność obliczeniową ograniczeń na rozmiar alfabetu i długość słów synchronizujących.

Pakiet obliczeniowy dla synchronizacji

Wiele automatów synchronizujących o interesujących własnościach można odkryć przy pomocy obliczeń numerycznych. Przykładem jest wspomniany wcześniej automat J. Kari będący kontrprzykładem na hipotezę Černego-Pina. Ponadto niektórzy autorzy wykorzystują podejście typu *brute-force* do generowania wszystkich automatów o określonej liczbie stanów w celu badania rozkładu długości minimalnych słów synchronizujących, poszukiwania automatów ekstremalnych czy empirycznej weryfikacji różnorakich hipotez dotyczących synchronizacji. W pracy [H4] opisano bibliotekę, stworzoną wyłącznie na potrzeby takich właśnie eksperymentów. Jest to pierwszy tego typu pakiet dedykowany do badań nad synchronizacją, nie licząc pakietu TESTAS [Trahtman 2003], który niestety nie posiada otwartej architektury i składa się tylko z predefiniowanych algorytmów oraz niezbyt wygodnego interfejsu użytkownika.

Pakiet COMPAS (COMputing PAckage for Synchronization) napisany został w C++, ale wykorzystuje również bindowanie Python-C++, pozwalające użytkownikowi na pisanie skryptów w Pythonie, który jest bardzo "czytelnym" językiem. Architektura pakietu składa się z trzech warstw:

- biblioteki napisanej w C++, zawierającej implementację różnego typu automatów oraz algorytmów operujących na tych automatach,
- modułu pozwalającego na użycie biblioteki z poziomu języka skryptowego,
- graficznego interfejsu użytkownika, tworzącego środowisko podobne do Matlabowego, w którym można pisać skrypty w Pythonie.

Priorytetami podczas budowy pakietu były (w kolejności od najważniejszego):

- Generyczność, elastyczność, skalowalność – co pozwala na stosowanie pakietu do różnego typu problemów, np. operacji zarówno na automatach deterministycznych jak i niedeterministycznych, dla funkcji przejść zupełnych oraz częściowych, itd.
- Efektywność – algorytmy operujące na automatach zostały napisane od zera w C lub C++ i zoptymalizowane pod kątem szybkości działania, np. poprzez wykorzystanie biblioteki boost.
- Użyteczność – w tym aspekcie jakościowym najważniejsza była łatwość użycia. Użytkownik biblioteki nie musi być ekspertem w C++, ale posiadając przynajmniej podstawową wiedzę o tym języku jest w stanie wykorzystać większość zaawansowanych funkcjonalności pakietu. Maksymalna optymalizacja pakietu wymaga już jednak zaawansowanej wiedzy z C++.

Pakiet wykorzystuje polimorfizm statyczny, czyli możliwość wykorzystania obiektów różnego typu poprzez wspólny, ujednolicony interfejs. Zaletą tego podejścia jest uzyskanie zwiększonej wydajności, gdyż nie traci się czasu na realizację wirtualnych wywołań.

Pakiet COMPAS jest dostępny na stronie http://www.assembla.com/spaces/compas.

Wykorzystanie metod AI do problemu synchronizacji

W rozdz. 6. omówione zostaną między innymi prace dotyczące nauczania maszynowego, a dokładniej – tzw. klasyfikatora hierarchicznego z nakładającymi się klastrami. Praca [H8] stanowi próbę praktycznego wykorzystania tego narzędzia do problemu predykcji długości MSS. Znajdowanie długości MSS można bowiem przedstawić jako problem klasyfikacji, w którym na

wejściu zadany jest automat, a wyjście stanowi klasa odpowiadająca długości MSS dla tego automatu.

W problemie klasyfikacji opisanym w [H8] konstruuje się zbiór uczący złożony z automatów o znanych długościach MSS. Dane te podaje się na wejście klasyfikatorowi hierarchicznemu, który stara się odkryć zależności pomiędzy strukturą automatu a długością jego MSS. Ponieważ etykiety klas odpowiadających długościom słów określone są na skali stosunkowej, możliwe jest liczenie wprost błędu klasyfikatora poprzez stosowanie operacji arytmetycznych na etykietach.

W tak zdefiniowanym problemie kluczowa jest postać wejścia do klasyfikatora. Ponieważ wejściem tym musi być wektor o stałej długości, nie można reprezentować automatów wprost, np. poprzez określenie ich funkcji przejścia. Z drugiej strony, badania miały na celu odpowiedź na pytanie: jakie strukturalne własności automatu mają wpływ na długość jego MSS? Pytanie to jest o tyle istotne, że znajomość przynajmniej szacunkowej długości MSS daje dobry punkt startowy dla algorytmu genetycznego (zdefiniowanie maksymalnej długości chromosomu) czy wykładniczego. W związku z powyższym ustalono, że wejściem będzie wektor liczb rzeczywistych kodujących określone własności strukturalne, zwane cechami automatu. Przyjęto następujący zbiór siedmiu cech:

(F1) największy wspólny dzielnik długości cykli etykietowanych tą samą literą,
(F2) średnia długość takich cykli,
(F3) maksymalna długość ścieżki do cyklu po krawędziach etykietowanych tą samą literą,
(F4) odsetek stanów leżących na cyklach,
(F5) liczba stanów synchronizowalnych[5],
(F6) średni poziom synchronizowalności stanu,
(F7) średnia długość ścieżki do stanu-singletonu w automacie par.

Zbadano również korelacje pomiędzy wartościami tych cech a długością MSS. Nie była zaskoczeniem bardzo wysoka korelacja MSS z cechą (F7). Wysoką korelację zaobserwowano również w przypadku cech (F4) i (F6). Klasyfikator przetestowano na automatach liczących od 4 do 8 stanów. Błąd MSE klasyfikatora wahał się między 0.77 a 10.29 i był o wiele niższy od błędu dla warstwowej sieci neuronowej (od 5.19 do 42.4).

Wyniki pracy [H8] pokazują, że modele nauczania maszynowego, takie jak klasyfikator hierarchiczny, mogą być bardzo dobrym rozwiązaniem wydajnego (szybkiego) i efektywnego (mały błąd) oszacowania długości MSS dla zadanego automatu. Zarówno model klasyfikatora jak i doświadczenie oprogramowane zostały w postaci skryptów pakietu statystycznego R.

Minimalne słowa synchronizujące a struktura stanów synchronizujących

Pomimo wielu lat badań nad synchronizacją do tej pory nie znamy żadnego "analitycznego" warunku koniecznego i wystarczającego na to, by automat był synchronizujący. Jedyny tego typu warunek ma postać:

$\boldsymbol{A} = (Q, A, \delta)$ jest synchronizujący $\Leftrightarrow (\forall\, p, q \in Q)\ (\exists w \in A^*)\ \ \delta(p, w) = \delta(q, w)$.

Jest to jednak warunek o charakterze "algorytmicznym": aby go sprawdzić, należy zweryfikować, czy każdą parę stanów da się zsynchronizować jakimś słowem. Gdyby udało się uzyskać WKW opisany np. w terminach struktury monoidu przejść automatu, czy też wyrażony w inny sposób, np. za pomocą ilościowych własności strukturalnych grafu zadanego funkcją przejścia, można by wtedy zapewne znaleźć szybki algorytm weryfikacji, czy automat jest synchronizujący oraz uzyskać lepsze niż sześcienne górne ograniczenie na długość MSS. Być może nawet udałoby się udowodnić/obalić hipotezę Černego.

[5] Tzn. takich, do których wchodzą co najmniej 2 krawędzie etykietowane tą samą literą

Cykl prac [P3], [P4] i [P7] jest próbą znalezienia regularności pomiędzy długością MSS, a strukturą automatu wyrażoną w terminach tzw. stanów zawężających, czyli stanów, w których może następować synchronizacja. Aby stan był zawężający, muszą do niego wchodzić co najmniej dwie krawędzie etykietowane tą samą literą. Inspiracją do podjęcia tych badań było intuicyjne przekonanie, że im mniej w automacie "okazji" do synchronizacji, tym dłuższe będzie MSS dla tego automatu. Na przykład, w automatach Černego (nad alfabetem binarnym) jedna litera tworzy cykl na wszystkich stanach (a więc nie da się przy jej pomocy niczego zsynchronizować), druga zaś tworzy jednoelementowe cykle (pętle) na wszystkich stanach oprócz jednego, dla którego pod wpływem tej litery następuje przejście do innego stanu. Formalnie, funkcja przejścia dla n-stanowego automatu Černego $C_n = (\{0, 1, \dots, n-1\}, \{a, b\}, \delta)$ wygląda następująco:

$$\delta(p, x) = \begin{cases} p + 1 \ (mod\ n) \ \ dla \ \ x = a, \\ p \ \ dla \ \ p \neq n-1 \wedge x = b, \\ 0 \ \ dla\, p = n-1 \wedge x = b. \end{cases}$$

Jedynym miejscem, w którym może zajść jakakolwiek synchronizacja kilku stanów do jednego, jest zbiór stanów $\{0, n-1\}$, który pod wpływem b synchronizuje się do stanu 0. Automaty Černego są przykładem "najtrudniejszej" synchronizacji: nie dość, że jest tylko jedno miejsce, w którym może nastąpić synchronizacja stanów, to dotyczy ona jedynie 2 stanów. Zauważmy, że automaty Černego są ekstremalne, czyli długości ich MSS osiągają dokładnie wartość z hipotezy Černego. Intuicyjnie, im więcej miejsc synchronizacji i im więcej stanów w takich miejscach schodzi się do jednego stanu pod wpływem tej samej litery, tym łatwiej powinno być zsynchronizować automat, a więc tym krótsze powinno być MSS. W pracy [P3] przedstawiono wyniki eksperymentów numerycznych, które pokazują pewne regularności pomiędzy strukturą automatu wyrażoną w terminach stanów zawężających a długością MSS. W pracy [P7] rozszerzono badania na automaty nad 3 i 4-elementowymi alfabetami. W obu pracach udowodniono pewne twierdzenia uzasadniające występowanie wyżej opisanych regularności. W pracy [P4] zastosowano teorię stanów zawężających do skonstruowania klasy automatów spełniających hipotezę Černego. Twierdzenie opisujące tę klasę wykorzystuje metodę szacowania długości MSS, która daje ograniczenie sześcienne. Korzystając jednak z własności stanów zawężających można pokazać, że dla pewnych klas automatów ograniczenie to będzie kwadratowe.

Podsumowanie

W rozprawie można wyróżnić pięć następujących, głównych wyników:

- wyniki z zakresu badania złożoności problemów synchronizacji i kolorowania drogi: rozwiązanie problemu złożoności PKD postawionego przez M. Volkova w 2008 roku, problemu istnienia jądra wielomianowego dla problemu synchronizacji postawionego przez H. Fernaua w 2013 roku, zbadanie parametrycznej złożoności obliczeniowej dla problemów związanych z PKD oraz wykazanie nieintuicyjnego faktu, że niektóre problemy wyrażane w języku PKD są znacznie trudniejsze niż ich odpowiedniki wyrażane w terminach "klasycznych" automatów (prace [H3], [H5], [H7], [H9], [H10], [H11]),
- zaprojektowanie i implementacja pakietu obliczeniowego w C++ i Pythonie dedykowanego dla problemów synchronizacji oraz algorytmu genetycznego dla znajdowania możliwie krótkich słów synchronizujących dołączonego do pakietu COMPAS, a także empiryczne wykazanie skuteczności tego algorytmu (prace [H2], [H4]),
- opracowanie i implementacja wielomianowych, heurystycznych algorytmów synchronizujących, zastosowanie specyficznych struktur danych w celu optymalizacji ich działania, określenie złożoności obliczeniowej algorytmów i porównanie ich z istniejącymi rozwiązaniami, w tym klasycznym algorytmem Eppsteina (praca [H6]),

- uogólnienie pojęcia synchronizacji oraz wykazanie, że słowa synchronizujące dla uogólnionej synchronizacji mogą mieć wykładniczy rozmiar w stosunku do liczby stanów automatu, podczas gdy w przypadku klasycznej synchronizacji długość tych słów jest wielomianowa ($O(n^3)$) (praca [H1]),
- wykorzystanie metod nauczania maszynowego w postaci Klasyfikatora Hierarchicznego do określenia wpływu pewnych cech automatów na długość minimalnego słowa synchronizującego (praca [H8]).

5. Omówienie pozostałego dorobku naukowego.

[11] Adam Roman, Testowanie i jakość oprogramowania. Modele, techniki, narzędzia, **Wydawnictwo PWN** (2015), xlvi+1080 pp., ISBN 978-83-01-18160-4

[10] Bartosz Zieliński, Adam Roman, Agata Dróżdż, Agata Kowalewska, Marzena Frołow, A new approach to automatic continuous artery diameter measurement, **IEEE Proc. of the 2014 Federated Conf. on Comp. Sci. and Inf. Systems** (**FEDCSIS**) (2014), 247–251

[9] Adam Roman, Testowanie eksploracyjne – cudowny lek czy zwykłe placebo?, **Software Developer's Journal** 6/2014 (2014), 4–15

[8] W. Beckler, K. Osiewalski, A. Roman, K. Bartocha, lastminute.com's 12-week geo-experiment reveals a 43% uplift in Google AdWords values, **Google AdWords Blog**, adwords.blogspot.com (2013)

[7] I.T. Podolak, A. Roman, Theoretical foundations and practical results for the hierarchical classifier, **Computational Intelligence** 29(2) (2013), 357–388

[6] A. Roman, I.T. Podolak, A. Deszyńska, On the number of clusterings in a HC model with overlapping clusters, **Schedae Informaticae** 20 (2011), 137–157

[5] I.T. Podolak, A. Roman, Risk function estimation for subproblems in a hierarchical classifier, **Pattern Recognition Letters** 32(15) (2011), 2136–2142

[4] I.T. Podolak, A. Roman, Risk estimation for hierarchical classifier, W: Corchado E., Kurzyński M., Woźniak M. (Eds.) Hybrid Artificial Intelligent Systems, **Lecture Notes in Artificial Intelligence** 6678 (2011), 156–163

[3] I.T. Podolak, A. Roman, Fusion of supervised and unsupervised training methods for a multi-class classification problem, **Pattern Analysis and Applications** 14(4) (2011), 395–413

[2] I.T. Podolak, A. Roman, A new notion of weakness in classification theory, W: Kurzyński M., Woźniak M. (Eds.) Computer Recognition Systems 3, **Advances in Intelligent and Soft Computing** 57 (2009), 239–245

[1] P. Kalita, I.T. Podolak, A. Roman, B. Bierkowski, Algorithm for Intelligent Prediction of Requests in Business Systems, W: Geffert V. et al. (Eds.) SOFSEM 2008: Theory and Practice of Computer Science, **Lecture Notes in Computer Science** 4910 (2008), 696–707

Prace [2]–[7] rozwijają koncepcję tzw. hierarchicznego klasyfikatora z nakładającymi się klastrami (HCOC), który został zaproponowany przez Podolaka [Podolak 2007], [Podolak 2008]. Badania podjęte w tych publikacjach dotyczą zarówno własności teoretycznych HCOC jak i wykorzystania HCOC do rozwiązywania praktycznych problemów. HCOC ma strukturę drzewa słabych klasyfikatorów, z których każdy rozwiązuje problem na swoim poziomie i albo zwraca końcowy

wynik (wektor klasyfikacji do klas), albo wskazuje podział odpowiadającego mu problemu na dalsze podproblemy. W pracach [2]–[7] zbadano następujące zagadnienia:

- metoda oceny za pomocą symulacji, czy dany problem nadaje się do dalszego podziału na podproblemy (w sensie HCOC) *przed* jego rzeczywistym wykonaniem,
- analiza działania HCOC w odniesieniu do teorii słabego nauczania (ang. *weak learning*),
- wprowadzenie nowej, zmodyfikowanej definicji słabego klasyfikatora, która jest odpowiednia dla problemu klasyfikacji do więcej niż dwóch klas,
- efektywność działania HCOC dla różnych architektur modelu (sposób podziału na podproblemy, rodzaj klasyfikatorów w poszczególnych węzłach itp.),
- doświadczenia – zbadanie działania HCOC dla znanych repozytoriów danych dla problemów uczenia maszynowego.

Praca [1] powstała w ramach grantu ASK-IT. Stanowi próbę zastosowania technik teoriografowych do przewidywania żądań w systemach biznesowych. Praca [8] jest studium wykonanym na potrzeby firmy lastminute.com dotyczącym marketingu internetowego przy użyciu narzędzia Google AdWords. Dotyczy ona wpływu tzw. *non-brand keywords*[6] na inne (w szczególności tzw. *brandowe*) słowa kluczowe. W celu wykazania istnienia takiego wpływu zaprojektowano eksperyment polegający na czasowym wyłączeniu kampanii reklamowych na określonych typach słów w określonych geolokalizacjach. Następnie zbadano wspomniany wcześniej wpływ przy pomocy metod statystyki bayesowskiej. Okazało się, że każde zamówienie złożone przez klienta, który dotarł na stronę lastminute.com w wyniku wyszukiwania słów typu *non-brand* ma wpływ na złożenie średnio 0.43 zamówienia dokonanego w wyniku użycia słów typu *brand*. Doświadczenie to pozwala optymalizować budżety kampanii internetowych poprzez efektywniejszy podział pieniędzy na poszczególne kanały marketingowe, w szczególności kanały SEO-nonbrand i SEO-brand. Praca [8] jest krótka i może wydawać się dosyć prosta, jednak cała trudność polegała tu na bardzo uważnym przygotowaniu eksperymentu tak, by uniknąć różnego rodzaju błędów oraz zminimalizować wariancję pochodzącą z innych źródeł niż rodzaj słów kluczowych. Ponadto, przeprowadzenie eksperymentu było dosyć kosztowne, ponieważ wymagało wyłączenia przez lastminute.com kampanii marketingowych w Google na dużym obszarze Wielkiej Brytanii. Praca [9] jest tekstem popularno-naukowym skierowanym do społeczności testerskiej w Polsce i omawia jedną z ważnych strategii testowania opartą na testowaniu eksploracyjnym.

Monografia "Testowanie i jakość oprogramowania. Modele, techniki, narzędzia"

W 2015 roku PWN wydało monografię "Testowanie i jakość oprogramowania. Modele, techniki, narzędzia" [11], którą uważam za najważniejszą z moich dotychczas napisanych prac. Celem publikacji jest przedstawienie polskiemu czytelnikowi w możliwie najpełniejszy sposób zagadnień testowania i jakości oprogramowania w usystematyzowany sposób. Książka przeznaczona jest zarówno dla początkujących, jak i zaawansowanych testerów, a także kierowników testów, kierowników projektów, programistów i pracowników nauki prowadzących badania w zakresie zapewniania jakości oprogramowania. Szczegółowo i krytycznie omawia techniki projektowania testów oparte o specyfikację, strukturę oraz doświadczenie. Opisuje modele jakości oraz testowanie charakterystyk jakościowych zgodnych z nową rodziną norm ISO 25000. Wiele miejsca poświęcone jest kwestiom związanym z dokumentacją i zarządzaniem procesem testowym. W monografii przedstawiono także różne podejścia do udoskonalania procesu testowego, zarówno przy pomocy modeli referencyjnych procesu, modeli referencyjnych zawartości jak i podejścia analitycznego. Ostatnia część książki stanowi wprowadzenie do inżynierii jakości oprogramowania. Omawia metryki i modele jakości, w tym modele defektów, efektywności usuwania defektów oraz wzrostu

[6] *Non-brand keyword* to słowo kluczowe nie zawierające odniesienia do marki produktu. Słowem takim może być np. wpisana w wyszukiwarkę Google fraza "tanie samochody", w przeciwieństwie do zwrotu "tanie samochody marki Honda", które związane jest z określoną marką produktu.

niezawodności. Wszystkie pojęcia, techniki i modele są bogato ilustrowane przykładami ich praktycznych zastosowań.

Planowane publikacje oraz badania

W ramach działalności naukowej prowadzę obecnie szereg badań związanych z testowaniem oraz zapewnianiem jakości oprogramowania. Projekty te (aktualne i planowane w najbliższej przyszłości) obejmują m.in.:

- Badania nad nowymi metrykami jakości oprogramowania opisującymi dynamikę tworzenia kodu i służącymi do budowy modelu predykcji defektów. Przeprowadzony w Instytucie Informatyki UJ eksperyment na ok. 80 studentach dał bardzo obiecujące wyniki, omówione w obronionej w 2014 roku pracy magisterskiej pisanej pod moją opieką. Obecnie przygotowywana jest publikacja podsumowująca te badania.
- Badania nad rozproszonym, skalowalnym, adaptacyjnym systemem do przeprowadzania testów mutacyjnych oraz nad automatycznym odkrywaniem nowych operatorów mutacyjnych na podstawie analizy repozytorium kodu źródłowego.
- Optymalizacja suit testowych dla metody testowania maszyny stanowej przy użyciu technik teorii grafów oraz sztucznej inteligencji.
- Przewidywanie tzw. *infeasible paths* oraz błędnego wykonania przy użyciu metod nauczania maszynowego.
- Ocena charakterystyk jakościowych oprogramowania przy użyciu metod nauczania maszynowego.
- Współpraca (R&D) z firmą Sabre Holdings w zakresie optymalizacji oraz zwiększenia efektywności testów dla projektu związanego z aplikacją do wyszukiwania tras przelotów statków powietrznych.

Współpraca z przemysłem IT

Od kilku lat intensywnie współpracuję z firmami z branży IT. Współpraca ta ma zarówno charakter naukowy jak i wdrożeniowo-implementacyjny, przy czym realizowane przeze mnie prace projektowe i wdrożeniowe zwykle nie były zadaniami czysto technicznymi, lecz wymagały podejścia naukowego.

Współpraca z Motorola Poland

Od 2009 roku prowadzę w Instytucie Informatyki UJ seminarium "Testowanie i jakość oprogramowania". Na część spotkań zapraszani są pracownicy firmy Motorola Poland, którzy przedstawiają zagadnienia związane z testowaniem i jakości oprogramowania. Efektem tych działań jest np. praca magisterska pisana pod moją opieką we współpracy z Motorolą. Praca polegała na stworzeniu narzędzia do manipulacji danymi binarnymi na potrzeby automatycznej generacji fragmentów kodu dla protokołów wykorzystywanych w systemach TETRA oraz APCO 25. Wyniki tej pracy są wdrożone i wykorzystywane w firmie. Byłem także opiekunem obronionej w 2014 r. pracy magisterskiej dotyczącej szacowania ETA (Estimated Time of Arrival) dla przeprowadzanych w chmurze symulacji działania sieci telekomunikacyjnych. Kolejnym projektem tworzonym we współpracy z Motorolą jest skalowalny, adaptacyjny system do automatycznego przeprowadzania testów mutacyjnych.

Współpraca z firmą Yellow-Lab

W ramach współpracy z firmą Yellow-Lab prowadziłem prace nad platformą do automatycznej oceny zdolności programistycznych deweloperów. Istotną częścią systemu jest moduł automatycznej oceny jakości rozwiązań przesyłanych przez programistów. Jednym z elementów oceny jest automatyczna identyfikacja złożoności obliczeniowej algorytmów stworzonych przez

deweloperów. Drugim elementem oceny jest poprawność algorytmu. Sprawdzana jest ona na podstawie starannie skonstruowanych suit testowych, które w zamierzeniu starają się pokryć 100% najważniejszych kryteriów pokryć (np. instrukcyjne, decyzyjne, warunków itp.) dla *każdego* przesłanego algorytmu rozwiązującego zadany problem.

Współpraca z firmą Blackbird

W ramach współpracy z firmą Blackbird (staż w ramach projektu Wiedza-Praktyka-Kadry organizowanego przez Małopolską Agencję Rozwoju Regionalnego w Krakowie) prowadziłem prace nad platformą do automatycznej oceny umiejętności testerów oprogramowania. Według mojej najlepszej wiedzy jest to pierwszy tego typu system na świecie. Ocena kompetencji testerskich polega na automatycznym obliczeniu stopnia pokrycia szeregu biało- oraz czarnoskrzynkowych kryteriów pokrycia, takich jak: pokrycie decyzyjne, pokrycie rozgałęzień, pokrycie pair-wise, pokrycie specyfikacji itp. Ważną cechą systemu jest to, że zadania generowane są automatycznie i można definiować dla nich różną skalę trudności. Istotnym elementem zaprojektowanego systemu jest metoda oceny kompetencji testerskich, biorąca pod uwagę nie tylko stopień pokrycia różnych kryteriów, ale także rozmiar suity testowej, efektywność skonstruowanych przypadków testowych oraz czas rozwiązania zadania.

Bibliografia

[Adler 1970] Adler R.L., Weiss B., Similarity of automorphisms of the torus, Memoirs of the Amer. Math. Soc. 98 (1970), Providence, RI.

[Adler 1977] Adler R.L., Goodwyn L.W., Weiss B., Equivalence of topological Markov shifts, Israel J. of Math. 27 (1977), 49–63.

[Ananichev 2003] Ananichev D., Volkov M., Synchronizing monotonic automata, DLT 2003, Z. Esik and Z. Fulop (eds.), LNCS 2710 (2003), 111–121.

[Ananichev 2004] Ananichev D.S., Volkov M.V., Synchronizing Generalized Monotonic Automata, Workshop on Synchronizing Automata, Turku, Finland, 2004.

[Benenson 2003] Benenson Y., Adar R., Paz-Elizur T., Livneh L., Shapiro E., DNA molecule provides a computing machine with both data and fuel, Proc. National Acad. Sci. USA 100 (2003), 2191–2196.

[Berlinkov 2010] Berlinkov M.V., Approximating the Minimum Length of Synchronizing Word is Hard, Lecture Notes in Computer Science 6072 (2010), 37–47.

[Biskup 2008] Biskup M.T., Shortest Synchronizing Strings for Huffman Codes, Lecture Notes in Computer Science 5162 (2008), 120–131.

[Broy 2005] Broy M., Jonsson B., Katoen J.-P., Leucker M., Pretschner A. (eds.), Model-Based Testing of Reactive Systems. Advanced Lectures, Lecture Notes in Computer Science 3472 (2005).

[Černý 1964] Černý J., Poznámka k homogénnym experimentom s konecnymi automatmi, Mat. fyz. Čas SAV 14 (1964), 208–215.

[Černý 1971] Černý J., Pirická A., Rosenauerova B., On directable automata, Kybernetica 7 (1971), 289–298.

[Dubuc 1998] Dubuc L., Sur les automates circulaires et la conjecture de Černý, RAIRO Inform. Theor. Appl. 32 (1998) 21–34.

[Eppstein 1990] Eppstein D., Reset sequences for monotonic automata, SIAM J. Comput. 19 (1990), 500–510.

[Eppstein 2002] Eppstein D., Falmagne J.-C., Algorithms for media, ACM Computing Research Repository, arXiv:cs.DS0206033 v1 (2002).

[Fernau 2014] Fernau H., Higgernes P., Villanger Y., A multi-parameter analysis of hard problems on deterministic finite automata, Journal of Computer and System Sciences 81(4) (2015), 747–765.

[Frankl 1982] Frankl P., An Extremal Problem for two Families of Sets, Europ. J. Comb. 3 (1982), 125–127.

[Fukada 2001] Fukada A., Nakata A., Kitamichi J., Higashino T., Cavalli A., A conformance testing method for communication protocols modeled as concurrent DFSMs. Treatment of non-observable non-determinism, Proc. IEEE 15th Int. Conf. on Information Networking 2001, 155–162.

[Ito 2004] Ito M., Shikishima-Tsuji K., Some results on directable automata, Lecture Notes in Computer Science 3113 (2004), 125–133.

[Jürgensen 2008] Jürgensen H., Synchronization, Information and Computation 206(9-10) (2008), 1033–1044.

[Kari 2001] Kari J., A counter example to a conjecture concerning synchronizing words in finite automata, EATCS Bulletin 73 (2001), 146.

[Kari 2002] Kari J., Synchronization and Stability of Finite Automata, JUCS, 8(2) (2002) 270–277.

[Kari 2003] Kari J., Synchronizing finite automata on Eulerian digraphs, Theoretical Computer Science 1-3 (2003), 223–232.

[Kisielewicz 2013] Kisielewicz A., Kowalski J., Szykuła M., A Fast Algorithm Finding the Shortest Reset Words, Lecture Notes in Computer Science 7936 (2013), 182–196.

[Martjugin 2006] Martyugin P.V., Lower bounds for length of carefully synchronizing words, International Computer Science Symposium, CSR 2006, St. Petersburg, Russia.

[Natarajan 1986] Natarajan B.K., An algorithmic Approach to the Automated Design of Parts Orienters, Proc. 27th Annual Symp. Foundations of Computer Science, IEEE, 1986, 132–142.

[Olschewski 2010] Olschewski J., Ummels M., The complexity of finding reset words in finite automata. Lecture Notes in Computer Science 6281 (2010), 568–579.

[Pin 1983] Pin J.-E., On two combinatorial problems arising from automata theory, Annals of Discrete Mathematics 17 (1983), 535–548.

[Podolak 2007] Podolak, I.T., Hierarchical rules for a hierarchical classifier, Lecture Notes in Computer Science 4431 (2007), 749– 757.

[Podolak 2008] Podolak I.T., Hierarchical classifier with overlapping class groups, Expert Systems with Applications, 34(1) (2008), 673–682.

[Pomeranz 1998] Pomeranz I., Reddy S.M., On synchronizing sequences and test sequence partitioning, Proc. IEEE 16th VLSI Test Symp., 1998, 158–167.

[Rystsov 1992] Rystsov I., The rank of finite automaton, Cybernetics and System An. 3 (1992), 3–10.

[Skvortsov 2011] Skvortsov E., Tipikin E., Experimental study of the shortest reset word of random automata, Lecture Notes in Computer Science 6807 (2011), 290–298.

[Skvortsov 2010] Skvortsov E., Zaks J., Synchronizing random automata, Discrete Mathematics and Theoretical Computer Science 12(4) (2010), 95–108.

[Trahtman 2003] Trahtman A.N., A package TESTAS for checking some kinds of testability. Lecture Notes in Computer Science 2608 (2003), 228–232.

[Trahtman 2004] Trahtman A.N., Černý Conjecture for DFA Accepting Star-Free Languages, Workshop on Synchronizing Automata, Turku, Finland, 2004.

[Trahtman 2009] Trahtman A.N., Road Coloring Problem, Israel Journal of Mathematics 172 (2009), 51–60.

[Trahtman 2006] Trahtman A.N., An Efficient Algorithm Finds Noticeable Trends and Examples Concerning the Černý Conjecture, Lecture Notes in Computer Science 4162 (2006), 789–800.

[Zhao 2010] Zhao Y., Liu Y., Guo X., Zhang C., Conformance testing for IS-IS protocol based on E-LOTOS, IEEE Int. Conf. on Information Theory and Information Security (ICITIS'10), 2010, 54–57.