INSTYTUT BADAŃ SYSTEMOWYCH
POLSKIEJ AKADEMII NAUK

TECHNIKI INFORMACYJNE TEORIA I ZASTOSOWANIA

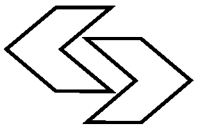
Wybrane problemy
Tom 1(13)

poprzednio

ANALIZA SYSTEMOWA W FINANSACH I ZARZĄDZANIU

Pod redakcją
Jerzego HOŁUBCA

Warszawa 2011



INSTYTUT BADAŃ SYSTEMOWYCH
POLSKIEJ AKADEMII NAUK

TECHNIKI INFORMACYJNE TEORIA I ZASTOSOWANIA

Wybrane problemy
Tom 1(13)

poprzednio

ANALIZA SYSTEMOWA W FINANSACH I ZARZĄDZANIU

Pod redakcją
Jerzego HOŁUBCA

Warszawa 2011

Wykaz opiniodawców artykułów zamieszczonych
w niniejszym tomie:

Dr hab. inż. Przemysław GRZEGORZEWSKI, prof. PAN

Prof. dr hab. inż. Jerzy HOŁUBIEC

Dr inż. Tatiana JAWORSKA

Dr hab. inż. Wiesław KRAJEWSKI, prof. PAN

Dr hab. inż. Maciej KRAWCZAK, prof. PAN

Dr hab. Michał MAJSTEREK

Dr hab. inż. Andrzej MYŚLIŃSKI, prof. PAN

Prof. dr hab. inż. Witold PEDRYCZ

Dr hab. inż. Ryszard SMARZEWSKI, prof. KUL

Prof. dr hab. inż. Andrzej STRASZAK

Dr Dominik ŚLĘZAK

Prof. dr hab. inż. Stanisław WALUKIEWICZ

ISBN 9788389475336

AUTOMATYCZNE GENEROWANIE SEKWENCJI WEJŚCIA-WYJŚCIA (UIO) ZA POMOCĄ ALGORYTMU PSO DO TESTÓW AUTOMATU SKOŃCZONEGO

Krzysztof Zaniewski
Studia Doktoranckie IBS PAN

Abstract: *Unique Input Output sequences (UIOs), generated from a formal specification, are used in conformance testing. Finding UIOs of minimal length is an NP-Hard problem. This study provides an approach to generate multiple UIOs automatically. The Particle Swarm Optimization (PSO) algorithm is used to optimize a multi-objective fitness function. The first objective concerns the uniqueness of the sequence while the second one captures its length. We discuss a way of encoding particles used in the PSO so that a combinatorial nature of the testing problem can be described. A set of tests comprises a number of finite state machines of different topology. The experimental evaluation shows that the proposed method outperforms the random search. We also report on the relationships between the topology of the finite state machine and the effectiveness of the PSO-generated tests. This paper shows optimization progress over time and examines how the proposed approach is affected by the topology and size of the finite state machine.*

Keywords*: Finite State Machine, Particle Swarm Optimization, Unique Input Output Sequences, Conformance Testing, Combinatorial optimization.*

1. Wstęp

Podczas testowania automatu skończonego najczęściej porównujemy specyfikacje automatu z jego implementacją. Błędy możemy podzielić na błędy wyjścia i błędy stanu. Błędy wyjścia występują, jeżeli wynik wyjściowy testu jest niezgodny ze specyfikacją automatu. Natomiast błędy stanu występują, kiedy po wykonaniu testu automat nie znajduje się w stanie końcowym.

W tej pracy skupiamy się na weryfikacji, czy po zakończeniu testu automat znajduje się w stanie końcowym. Do tego celu wykorzystana została metoda Unique Input Output sequences (UIOs), ponieważ jak pokazują dotychcza-

sowe badania [8], generuje ona najkrótsze przypadki testowe. Niestety generowanie minimalnych sekwencji UIO jest problemem NP-trudnym.

W dotychczasowych badaniach nad generowaniem minimalnych sekwencji UIO przedstawiano koncepcje wykorzystania algorytmów metaheurystycznych takich jak algorytmy genetyczne i symulowane wyżarzanie [1][7]. Jednak żadna z tych prac nie przedstawia tego problemu jako optymalizacji wielokryterialnej. Dotychczasowe prace wykorzystują funkcję przystosowania(fitness function), która pozwala generować krótkie sekwencje UIO, jednak nigdzie unikalność sekwencji i ich długość nie zostały od siebie jawnie oddzielone. Wygenerowanie długiej unikalnej sekwencji jest znacznie prostsze niż wygenerowanie minimalnej sekwencji UIO. Przeważnie, kiedy staramy się generować krótkie sekwencje sprawiamy, że nie jesteśmy w stanie znaleźć wszystkich unikalnych. Dlatego ważne jest znalezienie kompromisu pomiędzy unikalnością a długością sekwencji.

Niniejsza publikacja przedstawia przykład wykorzystania algorytmu Particle Swarm Optimization (PSO) w generowaniu sekwencji Unique Input-Output do testowania automatu skończonego. Pracę wyróżnia zastosowanie wielokryterialnej funkcji przystosowania, gdzie pierwszym kryterium jest unikalność sekwencji, natomiast drugim jest jej długość. Oryginalnym elementem jest również sposób kodowania (reprezentacji) sekwencji. Generowanie sekwencji UIO jest problemem kombinatorycznym, natomiast algorytm PSO służy do optymalizowania problemów numerycznych. Aby możliwe było zastosowanie algorytmu PSO w jego oryginalnej formie wykorzystany został innowacyjny sposób konwersji problemu kombinatorycznego do problemu numerycznego. Opisany sposób konwersji może zostać wykorzystany do rozwiązania wielu innych problemów kombinatorycznych za pomocą algorytmu numerycznego.

Celem pracy jest analiza zaproponowanego rozwiązania w procesie automatycznego generowania minimalnych sekwencji UIO ze specyfikacji automatu skończonego.

Publikacja podzielona jest na 7 części. Sekcje 1-4 są głównie teoretyczne i podsumowują dotychczasowe osiągnięcia naukowe. Natomiast sekcje 5-7 opisują wkład autora i prowadzone przez niego badania. W kolejnych rozdziałach poruszone zostały następujące zagadnienia:

- Sekcja 2-ga: teoria automatu skończonego.
- Sekcja 3-cia: teoria sekwencji UIO i sposób ich wykorzystania w procesie testowania.

- Sekcja 4-ta: algorytm PSO.
- Sekcja 5-ta: szczegóły konwersji problemu kombinatorycznego do problemu numerycznego oraz sposób liczenia funkcji przystosowania(fitness function).
- Sekcja 6-ta: środowisko testowe, dobór parametrów algorytmu i wyniki eksperymentów. Porównany został algorytm PSO z losowym generowaniem sekwencji.
- Sekcja 7-ta: podsumowanie i wyznaczenie możliwych celów przyszłych badań.

2. Automat skończony

Automat skończony (Finite-State Machine) jest matematycznym modelem opisującym układy logiczne lub programy komputerowe. Jest podobny do diagramu stanów w języku UML z tym, że jest znacznie mniej złożony. Automat składa się ze skończonej liczny stanów, transformacji pomiędzy tymi stanami, zbioru akcji i wyjść.

Istnieją dwa popularne modele automatu skończonego:

- Automat Mealy'ego[5]
- Automat Moore'a [6]

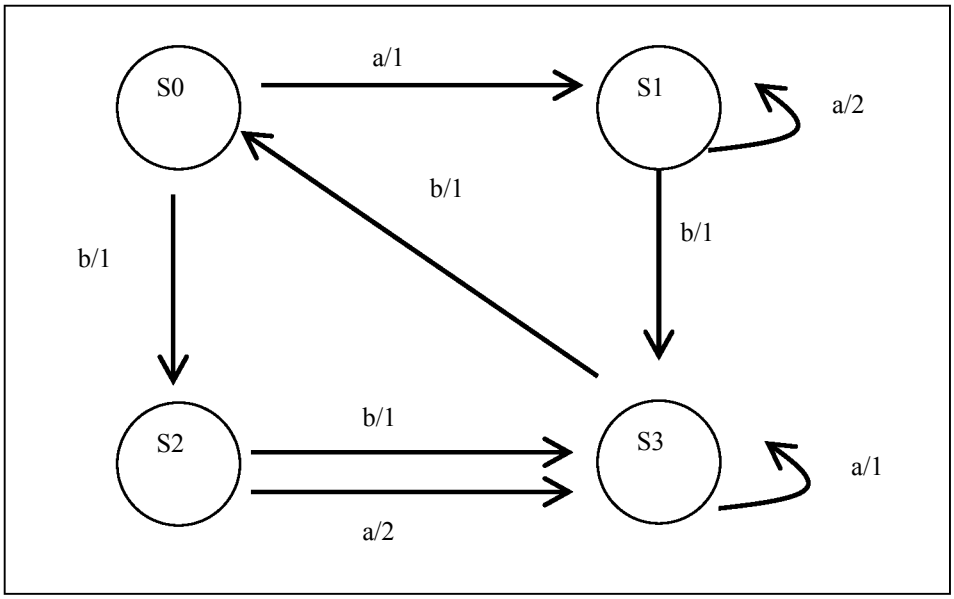
W tej pracy rozpatrujemy klasę automatów Mealy'ego. Każdy stan zawiera dostępną dla niego listę akcji, jakie można wykonać. Po wykonaniu akcji, automat przechodzi do następnego stanu i generuje wynik wyjściowy. Przejście z jednego stanu w inny nazywamy transformacją. Przykładowy automat skończony pokazany jest na Rys. 1. Przykładowo, rozpoczynając w stanie S0 wykonamy sekwencje akcji „ab": przejdziemy najpierw do stanu S1 a następnie do stanu S3 i otrzymamy wynik „11".

3. Sekwencje UIO

Sekwencje Unique Input Output wykorzystywane są w testowaniu protokołów sieciowych[23] oraz układów sekwencyjnych[24].

Podczas testowania automatu skończonego porównujemy specyfikacje automatu z jego implementacją. Błędy dzielimy na:

- błędy wyjścia
- błędy stanu.



Rys. 1: Przykładowy automat skończony Mealy'ego.
Stany (S1, S2, S3, S4), akcje (a, b), wyjścia (1, 2).

Błędy wyjścia występują, jeżeli wynik wyjściowy testu jest niezgodny ze specyfikacją automatu. Natomiast błędy stanu występują, kiedy po wykonaniu testu automat znajduje się w innym stanie niż powinien. Błędy wyjścia mogą zostać dosyć łatwo znalezione poprzez porównywanie wyniku wyjściowego testu z oczekiwanym. Błędy stanu mogą być wykryte poprzez sprawdzenie czy po wykonaniu testu znajdujemy się w stanie końcowym. Istnieją trzy sposoby weryfikacji stanu(ang. state verification):

- Unique input/output sequence (UIO) [1][7]
- Distinguishing sequence (DS) [10]
- Characterization set (W-set) [9]

W tej pracy skupiamy się na generowaniu sekwencji UIO (tak zwana U-method) ponieważ metoda ta generuje najkrótsze przypadki testowe [8]. Aby dodatkowo ograniczyć długość przypadków testowych generuje się jak najkrótsze sekwencje a problem generowania minimalnych sekwencji UIO jest NP-

trudny. Do generowania UIO używane były już techniki metaheurystyczne takie jak algorytmy genetyczne i symulowane wyżarzanie [1][7].

Sekwencje UIO służą do jednoznacznego identyfikowania stanu, w którym się znajdujemy. Dla każdego stanu generowana jest taka sekwencja akcji, aby wykonana na innym stanie dała inny wynik.

Przykładowe sekwencje UIO dla Rys. 1 to:

- Dla stanu S0 - ba
- Dla stanu S1 - aa
- Dla stanu S2 - aa
- Dla stanu S3 - aa

Na przykład, jeżeli wykonamy sekwencję „ba" na stanie S0, otrzymamy wynik „11". Wykonując tą samą sekwencje na innym stanie otrzymamy inny wynik. Oznacza to, że sekwencja „ba" jednoznacznie identyfikuje stan S0.

4. PSO

Algorytm został wynaleziony przez Kennedy James, Eberhart Russell i Shi Yuhui. Na początku celem algorytmu było symulowanie zachowań społecznych, później został on uproszczony i zastosowany do optymalizacji [2][3][4].

Algorytm zakłada istnienie populacji złożonej z cząstek, które poruszają się z pewną prędkością po wielowymiarowej przestrzeni współzawodnicząc ze sobą wzajemnie. Podczas doboru kierunku i prędkości uwzględniana jest najlepsza lokalizacja znaleziona przez dana cząstkę oraz najlepsza lokalizacja znaleziona w całej populacji tak, żeby cząstki przeszukiwały lokalizacje w pobliżu najlepszych rozwiązań. Algorytm jest sterowany czterema parametrami:

- Licznością populacji
- Współczynnikiem ω, który określa zmianę aktualnej prędkości cząstki (jak szybko maleje lub rośnie).
- Współczynnik φp, który określa wagę najlepszego znalezionego położenia przez daną cząstkę
- Współczynnik φg, który określa wagę najlepszego znalezionego położenia w całej populacji.

W każdej iteracji położenie i prędkość cząstek korygowane są wzorami:

$$\underline{v_i} = \omega * \underline{v_i} + \varphi p * rand() * (\underline{xbest_i} - \underline{x_i}) + \varphi g * rand() * (\underline{gbest} - \underline{x_i}) \quad ...(1)$$

$$\underline{x_i} = \underline{x_i} + \underline{v_i} \quad (2)$$

$$x[i] = x[i] + v[i]$$

gdzie:

i - numer cząstki w populacji,

$\underline{x_i}$ - współrzędne i-tej cząstki.

v_i - wektor prędkości i-tej cząstki.

$xbest_i$ - najlepsze znalezione położenie i-tej cząstki.

$gbest$ - najlepsze znalezione położenie w całej populacji.

$rand()$ – losuje wektor składający się z liczb w zakresie [0.0; 1.0] o rozkładzie równomiernym.

Schemat algorytmu wygląda następująco:

1. Inicjalizacja
   1.1. Określamy granice obszaru, po którym poruszają się cząsteczki.
   1.2. Losujemy pozycję startowe $\underline{x_i}$ i prędkości v_i każdej cząstki.
   1.3. Wybieramy najlepszą lokalizacje z całej populacji i przypisujemy do zmiennej $gbest$.
2. W każdej iteracji wykonujemy następujące operacje:
   2.1. Dla każdej cząstki.
      2.1.1. Uaktualniamy pozycję cząstki $\underline{x_i}$ wzorem ...(1).
      2.1.2. Uaktualniamy wektor prędkości v_i wzorem (2).
      2.1.3. Aktualizujemy zmienną $xbest_i$.
   2.2. Aktualizujemy zmienną $gbest$.

5. Proponowane rozwiązanie

Do wyszukiwania sekwencji UIO wykorzystany został standardowy algorytm PSO. W naszym przypadku cząstki poruszają się w przestrzeni n wymiarowej, gdzie n jest liczbą stanów maszyny. Generujemy po jednej sekwencji dla każdego stanu.

5.1 Reprezentacja problemu

Wyszukiwanie sekwencji UIO jest problemem kombinatorycznym, natomiast algorytm PSO służy do rozwiązywania problemów numerycznych. Z tego powodu potrzebne były pewne modyfikacje sposobu kodowania rozwiązań.

Bardzo ważne jest, aby po zakodowaniu, wykres funkcji przystosowania (fitness function) był jak najbardziej gładki, z jak najmniejszą liczbą minimów lokalnych, bez tak zwanych igieł(czyli bardzo dobrych wyników w bezpośrednim sąsiedztwie znacznie słabszych). Aby to osiągnąć należy określić odpowiednie reguły sąsiedztwa, mówiące, które kombinacje elementów powinny być sąsiadami.

Rozpatrzone zostały dwa sposoby kodowania. Oba sposoby kodowania dokonują konwersji liczby rzeczywistej będącej w przedziale [0.0; 1.0] do sekwencji akcji. W pierwszym etapie liczba z przedziału [0.0; 1.0] skalowana jest na liczbę całkowitą z przedziału [0; n-1], gdzie n to liczba wszystkich kombinacji. Przeskalowana liczba całkowita identyfikuje jednoznacznie daną kombinację. W przykładach z Tabeli 1 i 2 przedział liczbowy od 0.142 do 0.214 odpowiada kombinacji numer 3. W drugim etapie otrzymana liczba całkowita zamieniana jest na sekwencje akcji.

W przypadku kodowania pokazanego w Tabeli 1 reguła sąsiedztwa mówi, że sekwencje o takim samym początku, różniące się końcówką będą blisko siebie. Według tego kodowania sekwencje zaczynające się od innego elementu będą na przeciwległych biegunach.

W przypadku kodowania pokazanego w Tabeli 2 zastosowane zostały dwie reguły sąsiedztwa. Pierwsza reguła dotyczy długości sekwencji (sekwencje o tej samej długości będą bardzo blisko siebie). Druga reguła sąsiedztwa określa, że sekwencje o takim samym początku, różniące się końcówką będą blisko siebie.

Tabela 1: Przykład kodowania 1. Sekwencje o długości nie większej niż 3.

Sekwencja akcji	Numer sekwencji	Przedział liczbowy z zakresu od 0.0 do 1.0
a	1	(0.00; 0.07]
aa	2	(0.07; 0.14]
aaa	3	(0.14; 0.21]
aab	4	(0.21; 0.29]
ab	5	(0.29; 0.36]
aba	6	(0.36; 0.43]
abb	7	(0.43; 0.50]
b	8	(0.50; 0.57]
ba	9	(0.57; 0.64]
baa	10	(0.64; 0.71]
bab	11	(0.71; 0.79]
bb	12	(0.79; 0.86]
bba	13	(0.86; 0.93]
bbb	14	(0.93; 1.00)

Tabela 2: Przykład kodowania 2. Sekwencje o długości nie większej niż 3.

Sekwencja akcji	Numer sekwencji	Przedział liczbowy z zakresu od 0.0 do 1.0
a	1	(0.00; 0.07]
b	2	(0.07; 0.14]
aa	3	(0.14; 0.21]
ab	4	(0.21; 0.29]
ba	5	(0.29; 0.36]
bb	6	(0.36; 0.43]
aaa	7	(0.43; 0.50]
aab	8	(0.50; 0.57]
aba	9	(0.57; 0.64]
abb	10	(0.64; 0.71]
baa	11	(0.71; 0.79]
bab	12	(0.79; 0.86]
bba	13	(0.86; 0.93]
bbb	14	(0.93; 1.00)

Pokazane algorytmy konwersji można zastosować do wielu problemów kombinatorycznych. Co nie wyklucza, że istnieją inne sposoby konwersji bardziej efektywne dla konkretnego problemu optymalizacyjnego.

5.2 Funkcja przystosowania

Zastosowana została funkcja przystosowania(fitness function), która uwzględnia unikalność sekwencji UIO oraz kara za długie sekwencje. Liczona funkcja przystosowania (3) posiada wagi unikalności i kary za długie sekwencje. Odpowiadają za to parametry:

- w1 – waga unikalności sekwencji UIO
- w2 – waga kary za długie sekwencje

W przypadku, kiedy oba parametry są równe wtedy funkcja przystosowania uwzględnia unikalność sekwencji w takim samym stopniu jak jej długość. Innymi słowy bardzo długa, unikalna sekwencja jest równoważna bardzo krótkiej, zupełnie nieunikalnej sekwencji. Jest to spowodowane tym, że oba elementy funkcji przystosowania posiadają ten sam zakres wartości.

Funkcja przystosowania określona jest następująco:

$$\sum_{i=1}^{n}(w1 * CalculateUniquiness(uio_i) + w2 * Length(uio_i)) \qquad (3)$$

.....

gdzie,

$_i$ – numer sekwencji
n – liczba stanów maszyny
uio_i – i-ta sekwencja
$CalculateUniquiness(x)$ – określna unikalność sekwencji x.
$Length(x)$ – określa długość sekwencji x.

Funkcja CalculateUniquiness - zwraca liczbę całkowitą określającą ile razy wynik sekwencji powtarza się, kiedy sekwencję uruchomimy na wszystkich stanach. Jeżeli funkcja zwróci wynik równy 0, oznacza to, że jest to sekwencja unikalna. Jeżeli funkcja zwróci wynik 1 oznacza to, że sekwencja daje taki sam wynik na 2-ch stanach. Zero jest wartością minimalną i oznacza znalezienie sekwencji UIO. Funkcja CalculateUniquiness zwraca wartości z zakresu (0; stateCount -1), gdzie stateCount to liczba stanów automatu.

Funkcja Length - zwraca liczbę całkowita określającą długośc sekwencji. Funkcja Length zwraca wartości z zakresu (0; stateCount -1), gdzie stateCount jest liczbą stanów automatu.

6. Testy i wyniki badań

Testy algorytmu zostały wykonane na automatach skończonych, które przedstawione są w postaci grafów. Podczas losowania grafu, pewne jego parametry były określane przez autora tak, aby przetestować automaty o najróżniejszej topologii. Dotyczy to następujących parametrów:

- Liczba stanów
- Liczba akcji
- Liczba wyjść
- Gęstość grafu

Na każdym grafie test został uruchomiony 10 razy. Wybrane topologie są przedstawione w Tabela 3. Gęstość grafu jest podana w procentach. 100% określa maksymalną gęstość dla maszyny o podanej ilości stanów i akcji. Na przykład dla maszyny nr 11 maksymalna liczba przejść wynosi 64 (16stanów* 4akcje = 64 przejścia). Współczynnik gęstości 50% oznacza, że są 32 przejścia.

Tabela 3: testowane topologie

Numer grafu (nazwa)	Liczba stanów (state)	Liczba akcji (action)	Liczba wyjść (output)	Gęstość grafu (%).
1 (8-2-2-100)	8	2	2	100%
2 (8-2-3-100)	8	2	3	100%
3 (8-3-2-100)	8	3	2	100%
4 (8-3-3-100)	8	3	3	100%
5 (16-2-2-100)	16	2	2	100%
6 (16-2-4-100)	16	2	4	100%
7 (16-4-2-100)	16	4	2	100%
8 (16-4-4-100)	16	4	4	100%
9 (16-2-2-75)	16	2	2	75%

10 (16-2-4-75)	16	2	4	75%
11 (16-4-2-50)	16	4	2	50%
12 (16-4-4-50)	16	4	4	50%
13 (16-4-2-33)	16	4	2	33%
14 (16-4-4-33)	16	4	4	33%
15 (24-2-2-100)	24	2	2	100%
16 (24-2-4-100)	24	2	4	100%
17 (24-4-2-100)	24	4	2	100%
18 (24-4-4-100)	24	4	4	100%
19 (24-2-2-75)	24	2	2	75%
20 (24-2-4-75)	24	2	4	75%
21 (24-4-2-50)	24	4	2	50%
22 (24-4-4-50)	24	4	4	50%
23 (24-4-2-33)	24	4	2	33%
24 (24-4-4-33)	24	4	4	33%

Algorytm PSO został porównany z losowym generowaniem sekwencji UIO.

Sekwencje losowe są generowane w następujący sposób:

1. Losujemy liczbę n, określającą długość sekwencji s.
2. Następne kroki wykonujemy dla i = 1 do n.
    a. Losujemy akcję a, stosując rozkład równomierny.
    b. s_i = a, gdzie s_i to i-ty element sekwencji s.

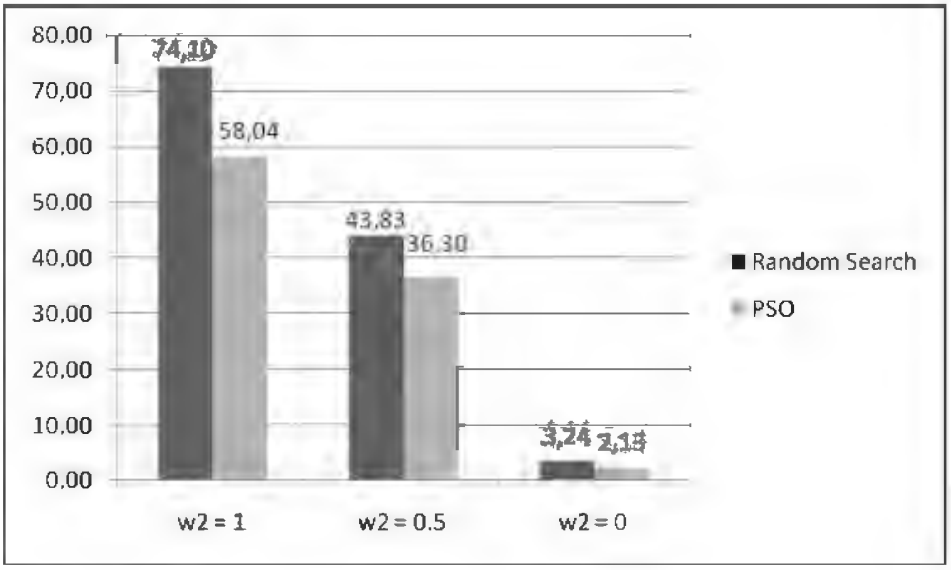
Parametry algorytmu PSO były dobierane eksperymentalnie. Skuteczność algorytmu jest bardzo mocno uzależniona od dobranych parametrów. Parametry algorytmu PSO zalecane przez innych autorów [11] dawały rezultaty podobne do losowego generowania sekwencji i konieczne było indywidualne dostrojenie algorytmu dla tego konkretnego problemu. Największe znaczenia ma parametr ω, natomiast φp, φg i liczebność populacji miały już nieco mniejsze znaczenie. Drobna zmiana sposobu liczenia funkcji przystosowania powodowała konieczność ponownego dobrania parametrów algorytmu. Podczas strojenia algorytmu ograniczona została początkowa prędkość, zamiast standardowych wartości <-1;1>, prędkość losowana jest z zakresu <-0.2, 02>, W przypadku, kiedy inicjowana prędkość była zbyt duża, cząstki zanim wytraciły prędkość

dużo czasu poświęcały na badanie wartości granicznych -1 lub 1, co jest zupełnie niepotrzebne i skutkuje wzrostem czasu obliczeń.

Przeprowadzone zostały trzy rodzaje testów, dla różnych wartości wag funkcji przystosowania:

1. w1 = 1, w2 = 1
2. w1 = 1, w2 = 0.5
3. w1 = 1, w2 = 0

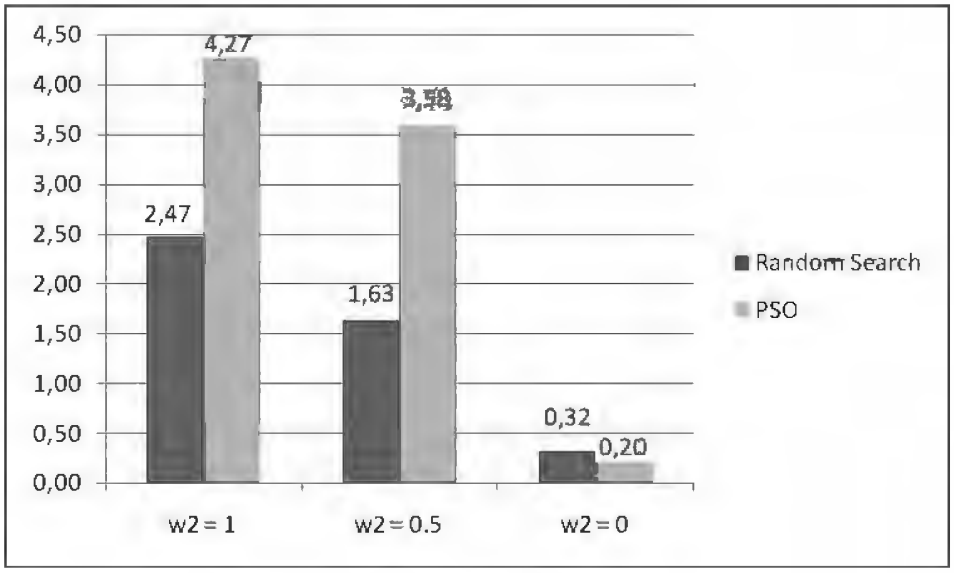
Wartości wag są kwestią indywidualną, zależną od potrzeb. Wybrane zostały różne parametry, żeby pokazać skuteczność algorytmu w przypadku, kiedy zależy nam na krótkich sekwencjach jak i w przypadku, kiedy zależy nam wyłącznie na unikalności.



Rys. 2: Uśrednione wyniki w zależności od wartości funkcji przystosowania.

W każdym przypadku wyniki (Rys. 2) pokazują, że algorytm PSO prowadzi do generowania lepszych sekwencji niż te generowane losowo. Wykres odchyleń standardowych (Rys. 3) pokazuje, że dla w2 = 1 i w2 = 0.5 odchylenia standardowe są niemal dwukrotnie większe dla algorytmu PSO. Sugeruje to, zależność wyniku od punktu startowego i możliwość zwiększenia skuteczności

działania algorytmu PSO poprzez wprowadzenie populacji konkurujących ze sobą. Z analizy odchyleń standardowych wynika, że jeżeli, zamiast uruchamiać test 10 razy, mielibyśmy 10 konkurujących ze sobą populacji różnica pomiędzy losowym generowaniem sekwencji a PSO powiększyłaby się na korzyść algorytmu PSO. Dla w2 = 0, odchylenie standardowe jest dosyć małe, ponieważ znalezienie optymalnego rozwiązania w przypadku, kiedy nie karamy za długość sekwencji jest łatwiejsze i algorytm PSO jest bardzo blisko optymalnego rozwiązania.



Rys. 3: Uśrednione odchylenia standardowe w zależności od wartości funkcji przystosowania.

Dla każdej wersji funkcji przystosowania dobrane zostały następujące parametry algorytmów:

1. kodowanie 2, ω=0.5, φp = φg = 2, populacja = 100
2. kodowanie 2, ω=0.3, φp = φg = 2, populacja = 100
3. kodowanie 1, ω=0.7, φp = φg = 2, populacja = 100

Warunkiem stopu było wykonanie 200 iteracji dla algorytmu PSO. Ze względu na to, że algorytm losowy nie posiada populacji, kryterium stopu było dla niego 20.000 iteracji.

Kodowanie 1 sprawdza się najlepiej w sytuacji, kiedy kara za długość sekwencji jest mniejsza lub wcale jej nie ma. Natomiast w sytuacji, gdy duży nacisk jest kładziony na to, aby sekwencje były jak najkrótsze, lepiej sprawdza się kodowanie 2. Jest to spowodowane innym zdefiniowaniem sąsiedztwa w poszczególnych kodowaniach.

Tabela 4: Wyniki optymalizacji wraz z odchyleniem standardowym dla funkcji przystosowania o wagach w1 = 1.0, w2 = 0.5

Numer grafu (nazwa)	Wynik ± Odchylenie standardowe	
	Random Search	PSO
1 (8-2-2-100)	11,80 ± 0,71	11,60 ± 1,36
2 (8-2-3-100)	7,75 ± 0,40	6,75 ± 0,81
3 (8-3-2-100)	12,90 ± 0,73	12,45 ± 0,96
4 (8-3-3-100)	6,75 ± 0,46	5,60 ± 0,92
5 (16-2-2-100)	48,90 ± 2,13	44,90 ± 3,86
6 (16-2-4-100)	38,15 ± 1,53	32,15 ± 5,22
7 (16-4-2-100)	52,65 ± 1,75	51,10 ± 4,89
8 (16-4-4-100)	35,50 ± 1,75	27,95 ± 4,34
9 (16-2-2-75)	35,05 ± 0,72	25,15 ± 1,16
10 (16-2-4-75)	26,10 ± 1,07	18,90 ± 1,39
11 (16-4-2-50)	28,65 ± 0,59	20,40 ± 1,73
12 (16-4-4-50)	14,50 ± 1,24	8,20 ± 1,73
13 (16-4-2-33)	14,45 ± 0,61	12,70 ± 1,60
14 (16-4-4-33)	10,95 ± 0,35	8,40 ± 0,70
15 (24-2-2-100)	115,75 ± 4,50	111,75 ± 7,54
16 (24-2-4-100)	83,10 ± 5,09	74,05 ± 5,57
17 (24-4-2-100)	118,40 ± 3,08	114,20 ± 10,36
18 (24-4-4-100)	87,95 ± 3,80	69,55 ± 10,62
19 (24-2-2-75)	70,25 ± 1,90	50,00 ± 3,49
20 (24-2-4-75)	67,70 ± 1,36	46,50 ± 7,14
21 (24-4-2-50)	57,20 ± 2,29	40,70 ± 2,43
22 (24-4-4-50)	39,10 ± 1,07	26,30 ± 1,66
23 (24-4-2-33)	44,80 ± 0,75	34,75 ± 4,12
24 (24-4-4-33)	23,60 ± 1,32	17,05 ± 2,37
Średnia	43,83 ± 1,63	36,30 ± 3,58

Wyniki testów dla funkcji przystosowania o wagach w1=1, w2= 0.5 są przedstawione w Tabeli 4. Zostały one poddane bardziej szczegółowej analizie. Ta wersja funkcji przystosowania została wybrana, ponieważ kara za długość sekwencji wydaje się być najbardziej odpowiednia. Dla w2=1, kara za długość sekwencji jest bardzo duża, natomiast dla w2=0 nie ma żadnej kary, co prowadzi do generowania bardzo długich sekwencji.

Wyniki przedstawione Tabeli 4 pokazują, skuteczność działania algorytmu PSO w zależności od topologii grafu. Niezależnie od topologii grafu wynik algorytmu PSO zawsze jest lepszy niż algorytmu Random Search. Na podstawie analizy Tabeli 4 można zaobserwować następujące tendencje:

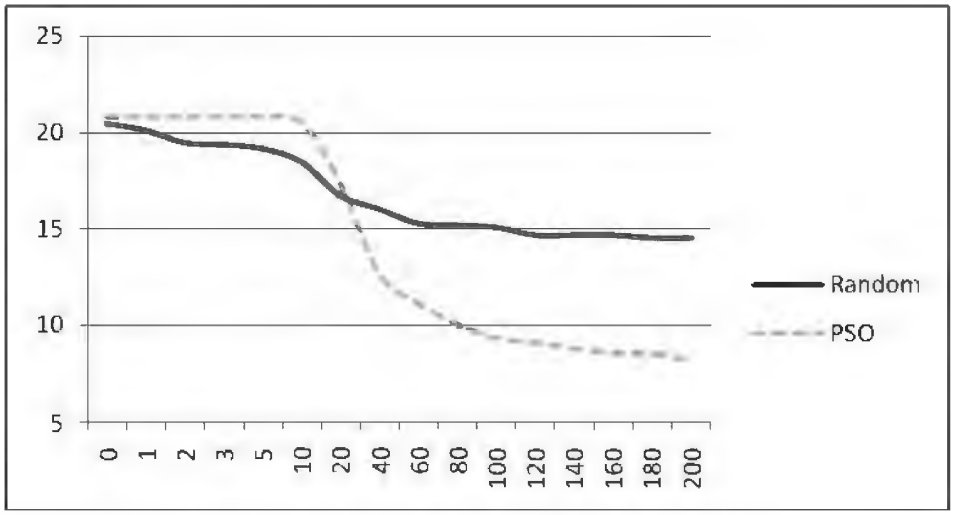
- Odchylenie standardowe algorytmu PSO jest większe w każdym przypadku.
- Grafy o mniejszej ilości stanów, charakteryzują się lepszym wynikiem.
- Grafy o mniejszej gęstości, charakteryzują się lepszym wynikiem.
- Grafy o większej liczbie wyjść (output), charakteryzują się lepszym wynikiem.
- Ilość akcji nie ma tak dużego znaczenia jak ilość stanów, gęstość grafu czy ilość wyjść.
- Różnica wyników pomiędzy algorytmem PSO a Random Search powiększa się wraz ze zmniejszaniem sie gęstości grafu. Tabela 5 pokazuje, że największa różnica 45,8% na korzyść algorytmu PSO jest na grafach o gęstości 50%. Natomiast dla grafów o maksymalnej gęstości różnica ta wynosi już tylko 10,2%.

Tabela 5: Średnie wyniki w zależności od gęstości grafu

Gęstość grafu	Random Search	PSO	Różnica
100%	51,63	46,84	10,24%
75%	49,78	35,14	41,66%
50%	34,86	23,90	45,87%
33%	23,45	18,23	28,67%

Bardzo interesująco wyglądają wyniki w zależności od czasu (liczby generacji). W początkowej fazie algorytm PSO jest słabszy niż losowe generowanie sekwencji. Natomiast z upływem czasu PSO poprawia wyniki, przewyższa-

jąc losowe generowanie sekwencji. Spowodowane jest to, początkową prędkością cząstek, które po jakimś czasie kierują się w stronę minimów lokalnych. Rys. 4 przedstawia przykładowy postęp optymalizacji na grafie 12 (16-4-4-50). Na wykresie widać, że do 10-ej iteracji algorytm PSO nie ma większych postępów. Dopiero około 10-ej iteracji algorytm PSO gwałtownie przyśpiesza. W okolicach 20-ej iteracji równa się z losowym optymalizatorem i w końcu go deklasuje.



Rys. 4: Postęp optymalizacji na grafie 12 (16-4-4-50). Algorytm PSO na tle Random Search. Oś pionowa to wynik funkcji przystosowania, natomiast oś pozioma to czas (numer iteracji).

7. Podsumowanie i przyszłe badania

W pracy przedstawiony został uniwersalny sposób konwersji problemu kombinatorycznego do problemu numerycznego na przykładzie algorytmu Particle Swarm Optimization. Algorytm PSO został wykorzystany w generowaniu sekwencji Unique Input-Output do testów automatu skończonego. Opisany sposób konwersji może zostać wykorzystany do rozwiązania wielu innych problemów kombinatorycznych za pomocą dowolnego algorytmu do optymalizacji numerycznej.

Badania wykazały skuteczność zaprezentowanego rozwiązania. Algorytm PSO deklasował losowe generowanie sekwencji, dając lepsze wyniki na każ-

dym z testowanych automatów, niezależnie od użytej wersji funkcji przystosowania.

Przyszłe badania mogą być prowadzone w następujących kierunkach:

- Modyfikacja algorytmu PSO poprzez wprowadzenie konkurujących ze sobą populacji.
- Użycie logiki rozmytej [12][22] w celu doboru parametrów algorytmu, w zależności od topologii maszyny.
- Sprawdzenie innych algorytmów: genetyczny[17], symulowane wyżarzanie[13][14], differential evolution[15][16] i porównanie ich z PSO.
- Zastosowanie algorytmu 2-u etapowego, gdzie wyniki wygenerowane przez pierwszy algorytm heurystyczny będą ulepszane przez algorytm typu Hill Climbing[18] w celu znalezienia minimum lokalnego.
- Zastosowanie meta optymalizacji w celu optymalnego dostrojenia parametrów[19][20].
- Zastosowanie omówionej techniki w testowaniu aplikacji poprzez graficzny interfejs użytkownika [21].
- Uproszczenie sposobu liczenia funkcji przystosowania lub liczenie jej przybliżonej wersji w celu zwiększenia wydajności [1].

Literatura

[1] Karnig Derderian, Robert M. Hierons, Mark Harman i Qiang Guo (2006), Automated Unique Input Output sequence generation for conformance testing of FSMs. *The Computer Journal*, Vol. 00, No. 0, 2006, Oxford University Press.

[2] Kennedy James, Eberhart Russell (1995), Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks*, IV:1942-1948, Piscataway, NJ.

[3] Shi Yuhui, Eberhart Russell (1998), A modified particle swarm optimizer. *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 69–73, Anchorage, AK , USA.

[4] Kennedy James (1997), The particle swarm: social adaptation of knowledge. Proceedings of IEEE International Conference on Evolutionary Computation, pp. 303–308, Indianapolis, IN.

[5] Mealy George (1955), A Method to Synthesizing Sequential Circuits. *Bell Systems Technical Journal*, pp. 1045–1079.

[6] Moore Edward (1956), Gedanken-experiments on Sequential Machines. *Automata Studies, Annals of Mathematical Studies*, 34, 129–153, Princeton University Press, Princeton, N.J.

[7] Qiang Guo, Robert Hierons, Mark Harman i Karnig Derderian (2005), Constructing Multiple Unique Input/Output Sequences Using Metaheuristic Optimisation Techniques. Software, *IEE Proceedings,* Vol. 152, No. 3, pp. 127-140, IET

[8] Dorofeeva Rita, El-Fakih Khaled, Maag Stephane, Cavalli Ana i Yevtushenko Nina (2005), *Experimental evaluation of FSM-based testing method*. Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods, pp. 23-32.

[9] Huaikou Miao, Pan Liu i Jia Mei (2010), *An Improved Algorithm for Building the Characterizing Set.* 4th IEEE International Symposium on Theoretical Aspects of Software Engineering, pp.67-74, Taipei, Taiwan.

[10] Robinson-Mallett Christopher, Liggesmeyer Peter, Mucke Tilo i Goltz Ursula (2005), *Generating optimal distinguishing sequences with a model checker*. Proceedings of the 1st international workshop on Advances in model-based testing, ACM New York, NY, USA.

[11] Pedersen Magnus (2010), Good Parameters for Particle Swarm Optimization. Hvass Laboratories Technical Report no. HL1001.

[12] Witold Pedrycz, Fernando Gomide (2007), *Fuzzy Systems Engineering: Toward Human-Centric Computing*. Wiley-IEEE Press

[13] Kirkpatrick S., Gelatt C. D., Vecchi M. P. (1983), Optimization by Simulated Annealing. *Science*, Vol. 220, pp. 671-680.

[14] Cerny V. (1985), Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, Vol. 45, No. 1, pp. 41-51.

[15] Storn Rainer (1996), *On the usage of differential evolution for function optimization.* Biennial Conference of the North American Fuzzy Information Processing Society, pp. 519–523.

[16] Storn Rainer, Price Kenneth (1997), Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of*

Global Optimization, Vol 11 Issue 4, Kluwer Academic Publishers Hingham, MA, USA.

[17] Fraser Alex (1957), Simulation of genetic systems by automatic digital computers. I. Introduction. *Australian Journal of Biological Science*, Vol. 10, pp. 484–491.

[18] Cormen Thomas, Leiserson Charles, Rivest Ronald, Stein Clifford (2001), *Introduction to Algorithms*, Chapter 16: Greedy Algorithms. The MIT Press.

[19] Pedersen Magnus, Chipperfield Andrew (2010), Simplifying Particle Swarm Optimization. *Journal Applied Soft Computing*, Vol 10, Issue 2, March, 2010, Elsevier Science Publishers B. V. Amsterdam, The Netherlands.

[20] Pedersen Magnus (2010), PhD thesis: Tuning & Simplifying Heuristical Optimization. University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.

[21] Krzysztof Zaniewski (2008), Praca magisterska: Automatyzacja testów aplikacji na platformie MS Windows. Warsaw Information Technology.

[22] Zadeh Lotfi (1965), Fuzzy sets. *Information and Control*, Vol. 8, pp. 338–353.

[23] B. Yang, H. Ural (1990), *Protocol conformance test generation using multiple UIO sequences with overlapping*. Proceeding SIGCOMM '90 Proceedings of the ACM symposium on Communications architectures & protocols, ACM New York, NY, USA

[24] Tetsu Hasegawa, Kyoko Miura, Toshiaki Ohmameuda, Hideo Ito (2001), Test generation for sequential circuits using state transition diagram and test generation for combinatorial circuit part. *Electronics and Communications in Japan* (Part II: Electronics), Vol. 84, pp. 20–28.

ISBN 9788389475336

J. Hołubiec, red. – TECHNIKI INFORMACYJNE-TEORIA I ZASTOSOWANIA, tom 1 (13)