AGH University of Science and Technology
Faculty of Computer Science, Electronics
and Telecommunication
Mickiewicza 30, 30-059 Kraków, POLAND; tel. (+4812)3283313, fax (+4812)6175172

Robert Schaefer, Professor, PhD
Department of Computer Science
Home: Batorego 5/4, 31-135 Kraków, POLAND
e-mail schaefer@agh.edu.pl

Kraków, 17 marzec 2022

**Recenzja rozprawy doktorskiej**
**Mgr inż. Iwo Błądka**
**zatytułowanej:**
**Machine Learning and Formal Verification for Acquisition of Knowledge in Heuristic Program Synthesis**

## 1. Problem badawczy i jego znaczenie

Zasadniczym problemem badawczym rozwiązanym w dysertacji jest opracowanie i zbadanie strategii wspieranych wiedzą stochastycznych algorytmów syntezy programów, w tym głównie algorytmów klasy Geneting Programming (GP). W taksonomii stochastycznej optymalizacji, strategie takie zaliczyć można do szeroko pojętej grupy strategii memetycznych.

Program podlegający syntezie jest rozumiany przez autora jako przygotowana wcześniej sekwencja akcji, które mają być wykonane przez maszynę. Zakłada się, że programy spełniają:

- Warunek stopu (program się zatrzymuje zawsze, dla każdych danych),
- Funkcjonalność, tj. output zawiera całość wyników obliczeń
- Nie ma pętli ani wywołań rekursywnych,
- Jest deterministyczny.

Ponadto autor przyjmuje następujące definicje:

*Zadanie syntezy programu* jest parą $(L, Correct)$ gdzie $L$ jest językiem programowania a $Correct: L \to B = \{0,1\}$ jest funkcją zwaną predykatem poprawności. Rozwiązanie tego zadania to program $p \in L$ spełniający predykat $Correct$.

*Problem syntezy programów* jest zbiorem wszystkich możliwych zadań syntezy programów dla ustalonego języka $L$ i ustalonej formy predykatów $Correct$.

W pracy rozważa się warianty problemów i zadań syntezy, w których poszukuje się programów spełniających predykat $Correct$, lub spełniający ten predykat oraz pewien warunek optymalności (np. minimalizuje pewną funkcje kosztu). Zakłada się ponadto, że każde z zadań posiada rozwiązanie (przynajmniej jedno).

Pierwszym rozwiązywanym zagadnieniem było wypełnianie dziur w ewoluującym programie, tj. ***Evolutionary Program Sketching*** **(EPS)**.

Motywacją dla podjęcia tego kierunku badań był brak wyników wykorzystujących w dostatecznym stopniu technik stochastycznej optymalizacji globalnej (algorytmów ewolucyjnych) dla rozwiązania tego problemu (np. prace grupy Armando Solar-Lorenza (2013)). Zaproponowana metoda polegająca na znajdowaniu optymalnych szkiców przy pomocy odpowiedniej modyfikacji GP stanowi oryginalny wkład autora w rozwiązanie tego zagadnienia.

Rozwiązywany problem można krótko scharakteryzować następująco: Dany jest $T = \{(in, out)\}$ – zbiór testów, $L$ – język programowania rozumiany jako zbiór programów, zdań poprawnych syntaktycznie w tym języku formalnym. Ponadto oznaczymy przez $b \in 2^L$ – uzupełnienie dziur, $p_b \in L$ – program z uzupełnionymi dziurami. Poszukujemy

$$\arg\max_{b \in A} \mathrm{card}\{(x, y) \in T; p_b(x) = y\}, \qquad A = \{b \in 2^L; p_b \in L\}.$$

W proponowanym rozwiązaniu uzupełnienia dziur $b$ jest obliczane są przez solwer Z3 posługujący się zmodyfikowaną techniką SMT (Satisfability Modulo Theories). Zakłada się, że uzupełnienia $b$ składają się tylko ze zbiorów stałych i zmiennych.

Zastosowany wariant obliczeń ewolucyjnych można traktować jako memetyczny, gdzie GP pełni rolę silnika ewolucyjnego zapewniającego odpowiedni poziom globalności poszukiwania powiązanego z deterministyczną metoda SMT jako metodą lokalną. W trakcie obliczeń stosowane były oba paradygmaty Baldwina i Lamarka dla oceny jakości oraz memetycznej modyfikacji elementów populacji GP.

W trakcie testów dla tworzenia optymalnych skeczów stosowano GP o dość dużej populacji 250. Stosowano warunek stopu zatrzymujący obliczenia po 100 epokach. Stosowano selekcję turniejową o wielkości maty 7 oraz mutację o bardzo wysokim prawdopodobieństwie 0.5, oraz prawdopodobieństwo krzyżowania również 0.5. Osobniki dopuszczalne generowane były za pomocą gramatyki NIA języka $L$.

Testowane były cztery warianty tej strategii EPS-Lc, EPS-Lv, EPS-Bc, EPS-Bv , oraz EPS-Lcv i EPS-Bcv odpowiadające technice Lamarka i Balwina z dziurami wypełnionymi stałymi lub zmiennymi oraz stałymi i zmiennymi.

Wyniki testów pokazują generalnie wydajniejsze działanie wariantów EPS-$B_{cv}$ i EPS-$B_c$ niż pozostałych. Wymienione warianty EPS wygrywają także z bazowymi technikami $GP_T$ i $GP_{500}$.

Powyższe wyniki oraz obserwacja wykresów pudełkowych dla tych eksperymentów pokazują dominujące znaczenie wysokiego poziomu chaosu w globalnej fazie proponowanej strategii. Przyjęta konfiguracja fazy globalnej skutkuje bardzo elastycznym i efektywnym przeszukiwaniem całej dziedziny..

Kolejnym rozwiązanym problemem było programowanie genetyczne sterowane kontrprzykładami ***Counterexample-Driven Genetic Programming*** **(CDGP).**

Programowanie genetyczne GP służące do syntezy programów wykorzystuje numeryczną funkcję przystosowania opartą głównie na wykorzystaniu wyników testów i porównaniu ich z zbiorem elementów typu $T = \{(in, out)\}$ zawierającym dane wejściowe i odpowiadające im poprawne dane wyjściowe. Celem proponowanej strategii jest umożliwienie sterowania ewolucją GP przy pomocy formalnych specyfikacji syntetyzowanego programu. Zadanie takie nie było dotąd podejmowane.

Autor zakłada, że specyfikacje formalne spełniają poniższe własności:

- Są jedynym źródłem informacji o poprawności działania programu, czyli o celu syntezy,
- Mogą mieć dowolną formę akceptowalną przez weryfikator, tj. moduł sprawdzający spełnienie specyfikacji przez ewoluujący program.

Proponowana strategia tworzy iteracyjnie zbiór testów na podstawie specyfikacji, które posiadają formę predykatów. Ważnym elementem przyjętej koncepcji jest „miękki" sposób weryfikacji zakładający progowy stopień spełnienia predykatów, nie mniejszy niż pewne $\alpha \in (0,1)$. Taka koncepcja umożliwia znaczne obniżenie kosztu obliczeniowego całej strategii.

Wykorzystywany silnik GP posługuje się następującymi konstrukcjami i operatorami:

- Wyrażenia typu SMT-LIB kodujące drzewa wywodów programów.
- Inicjalizacja osobników odbywa się poprzez wywód z symbolu startowego poprzez losowe, osadzalne produkcje.
- Operatory „mieszania" są typu produkcji gramatyki definiującej syntaktykę programów. Mutacja polega na wymianie poddrzew na losowo generowane, natomiast krzyżowanie na wymianie poddrzew o takim samym sposobie osadzenia. Znajdowanie takich poddrzew odbywa się na podstawie złożonego, wielokrotnego losowania głównie z zastosowaniem rozkładów równomiernych.

Stosowane są dwa typy selekcji rodziców: turniejowa o macie wielkości 7 gdzie w pojedynczym turnieju wygrywa osobnik o mniejszej ilości testów negatywnych, oraz selekcji lexicase dla zbioru benchmarków.

Stosowana jest sukcesja typu $(\mu, \lambda)$. Stosowany jest również schemat $(\mu + 1)$ gdzie dziecko podmienia jednego z rodziców. Na tym etapie stosuje się też operator deselekcji, polegający na wyrzucaniu najgorszych osobników. Może ona mieć postać negatywnego turnieju (wygrywa najgorszy).

Dla badań testowych CDGP wykorzystano benchmarki z repozytoriów sieciowych typu LIA (Linear Integer Arithmetic) oraz SLIA (String with LIA). Jako algorytm referencyjny wykorzystano algorytm GRP (Random GP). W obliczeniach przyjęto parametr progowy spełnialności $\alpha = 0.75$ który dawał najlepszą wydajność.

CDGP okazało się przydatne do rozwiązywania problemów syntezy dla obliczeń całkowitoliczbowych i obliczeń na stringach, bazujących na specyfikacjach formalnych. Porównanie z GRP wypada tutaj zdecydowanie korzystnie.

Porównanie z metodami dokładnymi EU i CVC4 pokazuje większy koszt obliczeniowy proponowanej strategii dla testów LIA. Natomiast CDGP generowała krótsze programy, ponadto na testach SLIA w porównaniu z CVC4 rozwiązywała poprawnie więcej benchmarków.

Kolejnym rozwiązywanym problemem była próba zastosowania CDGP do syntezy programów realizujących zadanie symbolicznej regresji rzeczywistoliczbowej ***Counterexample-Driven Symbolic Regresion*** **(CDSR).**

Aktywność ta była motywowana chęcią zbadania jak i czy można wykorzystać CDGP do problemów regresji symbolicznej, w której wprowadzamy formalne ograniczenia na funkcję (model) regresji.

Rozważany jest rodzaj regresji, dla której zmiennymi niezależnymi (regresorami) są wykonywalne artefakty z pewnego zbioru $D$. Ponadto mamy dany zbiór uczący $T = \{(x_1, y_1), \dots, (x_n, y_n)\}$ charakteryzujący nieznaną funkcję, tj. $f: D \to \mathbb{R}; f(x_i) = y_i, i = 1, \dots, n$ oraz zbiór ograniczeń $C$ mających postać wyrażeń logicznych 1-go rzędu dotyczących wartości funkcji $f$.

Rozważmy również pewien zbiór modeli $H = \{h: D \to \mathbb{R}\}$ to znaczy obiektów, przy pomocy których możemy przybliżać funkcje $f$. W analizowanym przez autora przypadku zbiór ten będzie językiem, którego zdania są wyrażeniami arytmetycznymi, poprawnymi syntaktycznie w pewnej gramatyce.

Problem regresji polega na znalezieniu $h \in H$ spełniającego w możliwie największym stopniu testy zawarte w zbiorze uczącym $T$ oraz ograniczenia $C$. Jakość przybliżenia $f$ przez $h$ określona jest w specjalny sposób, poprzez stopień spełnienia warunków ze zbioru $T$ oraz ograniczeń ze zbioru $C$.

Tak zdefiniowana regresja z formalnymi ograniczeniami (SRFC) jest szczególnym przypadkiem uczenia nadzorowanego z ograniczeniami.

Przebieg obliczeń jest podobny do CSGP, w trakcie ewolucji GP namnażany jest zbiór uczący na podstawie przekształconych ograniczeń. CDSR jest częściowo zrelaksowaną wersją CSGP, gdzie dopuszcza się występowanie szumu w zbiorze uczącym $T$, kryteria formalne spełnione są w stopniu $\alpha < 1$ oraz dopuszcza się wczesne stopowanie GP, jeżeli nie ma poprawy jakości regresji.

Oprócz podstawowego wariantu CDSR opracowano wariant $\text{CDRS}_\text{p}$ w którym stopień spełnienia ograniczeń dodaje się jako dodatkowe kryterium stosowane przez selekcję lexicase.

Pierwszym, przygotowawczym etapem testów było porównanie 10 algorytmów regresji state of the art dla wyłonienia najbardziej odpowiednich algorytmów referencyjnych. Wykorzystywano 11 funkcji jednej zmiennej które identyfikowano oraz zbiory uczące generowane z rozkładem równomiernym z przyjętej dziedziny (przedziału). Wykorzystywano również 3 – 5 formalnych ograniczeń. Każdy z utworzonych w ten sposób benchmarków składał się z:

- ograniczeń na parametry funkcji,
- 500 par $(in, out)$ spełniających ograniczenia formalne.

Na tym etapie najlepiej wypadły algorytmy AdaBoost i KernelRidge, które posłużyły do dalszego etapu testów.

W kolejnym etapie porównywano różne warianty CDSR. Zasadnicze wnioski z tego etapu można podsumować następująco:

- Średnio najlepszy ze względu na stopień spełnienia ograniczeń był wariant $\text{CDRS}_\text{p}$ . Wypadł on nieco słabiej w ocenie wyników na zbiorze uczącym $T$. Przewaga tego wariantu była tym silniejsza, im więcej było ograniczeń.
- Wariant z selekcją turniejową daje lepsze wyniki na zbiorze uczącym $T$.

Wynikiem tego etapu było wyselekcjonowanie pojedynczych najkorzystniejszych instancji CDSR i $\text{CDRS}_\text{p}$. Ostatnim etapem było porównanie tych algorytmów z wytypowanymi wcześniej algorytmami state of the art. Algorytmy klasy CDSR wykazały większą elastyczność niż klasyczne metody uczenia maszynowego, które nie wykorzystywały w pełni wiedzy zgromadzonej w ograniczeniach. Znalezione programy realizujące funkcje matematyczne spełniały więcej wprowadzonych ograniczeń. Koszt obliczeniowy algorytmów

CDSR był jednak istotnie wyższy. Głównym komponentem decydującym o dużym koszcie była implementacja selekcji lexicase. W eksperymentach stosowano bardzo duże populacje GP rzędu 1000 osobników.

Ostatnim rozwiązaniem opisanym w dysertacji jest synteza programów sterowana siecią neuronową ***Neuro-Guided Genetic Programming*** **(NGGP).**

Jest co całkowicie nowatorski pomysł sterowania nauczoną siecią ANN procesu syntezy prowadzonego przy pomocy GP. Sieć neuronowa stosowana jest do sterowania operatorem mutacji oraz inicjalizacją populacji (generacja populacji początkowej). Pomysł jest w pewnym stopniu inspirowany pracą Mateja Baloga i innych (2017), zawierającą ideę uwzględniania wiedzy domenowej w syntezie programów *Learning Inductive Program Synthesis* (LIPS). W proponowanym rozwiązaniu wykorzystano język specyfikacji domenowych podobny do DSL tego autora.

NGGP składa się z dwóch faz:

- Fazy nauki offline. Jej efektem jest sieć ANN służąca dalej do sterowania ewolucją GP.
- Faza online, w której nauczona ANN zwraca rozkład prawdopodobieństwa na zbiorze $L$ wyrażeń leksykalnych języka DSL który służy do wyboru instrukcji przez odpowiedni operator GP.

Dla przeprowadzenia testów autor skoncentrował się na rozwinięciu pewnej grupy instancji metodyki NGGP w której istotnie ograniczono długość syntetyzowanych programów oraz rodzaj instrukcji języka DSL, co pozwoliło ograniczyć ilość elementów języka do 34 pozycji. W konsekwencji trenowano na pewnym zbiorze uczącym sieć o 34 neuronach wyjściowych, zwracających nieznormalizowany wektor probabilistyczny. Architektura ANN jest zgodna z używaną w DeepCoder architekturą warstwową.

W obliczeniach GP sterowanych przez ANN stosowano:

- Stopowanie obliczeń po 200 generacjach lub gdy uzyskano program przechodzący przez wszystkie testy.
- Elementy populacji początkowej były programami zaczynającymi się od zmiennych wejściowych, a dalsze instrukcje były wybierane losowo, zgodnie z pewnym rozkładem prawdopodobieństwa.
- Mutacja poległa na wymianie losowej funkcji z programu na inną syntaktycznie zgodną z DSL w oparciu o pewien rozkład prawdopodobieństwa.
- Dla krzyżowania tworzymy multizbiory funkcji o takiej samej sygnaturze w każdym z rodziców. Później wybieramy losowo parę funkcji z multizbiorów o tej samej sygnaturze, a następnie wymieniamy funkcję pomiędzy rodzicami, pozostawiając wartości ich argumentów.

Dla testowania proponowanego rozwiązania posłużono się benchmarkami stosowanymi w pracy Mateja Baloga i innych (2017). Porównywano cztery warianty NGGP w różnym stopniu wykorzystujące wiedzę domenową:

- U – zwykłe GP z mutacją i inicjalizacją stosująca rozkład równomierny.
- P – poszukiwanie i inicjalizacja wykorzystują pewien ustalony rozkład prawdopobieństwa.

- S – rozkład zwracany przez nauczoną ANN jest stosowany tylko dla sterowania mutacją, a inicjalizacja wykorzystuje rozkład równomierny.
- IS – rozkład zwracany przez nauczoną ANN jest stosowana na wszystkich etapach.

Wyniki wariantów S i IS były systematycznie lepsze od pozostałych. Najlepsze był wariant IS. Wyniki wariantu U przewyższały wyniki wariantu P, co oznacza, że poszukiwanie powinno być dostatecznie chaotyczne. Wykorzystanie wiedzy domenowej wydaje najbardziej korzystne dla wsparcia poszukiwania stochastycznego syntezy.

Przedstawione w dysertacji wyniki skłaniają do sformułowania pytań, na których odpowiedzi mogą stanowić interesujące uzupełnienie treści dysertacji:

A. Autor zakłada, że każdy z rozważanych problemów syntezy posiada przynajmniej jedno rozwiązanie. Spełnienie tego warunku ma umożliwić odpowiednio duża moc opisowa użytego języka programowania.
Czy rozważane problemy mogą posiadać więcej niż jedno rozwiązanie?
Czy rozważane strategie syntezy dają możliwość znalezienia więcej niż jednego rozwiązania lub wszystkich rozwiązań w takim przypadku?
Czy znajdowanie większej ilości rozwiązań takich problemów może być umotywowane praktycznie?

B. Jaki może być stosunek kosztu (czasu CPU, RAM, etc.) etapów wstępnych, każdej z proponowanych metod do uzyskanego przyspieszenia (lub innej formy poprawy działania) dla pojedynczej syntezy w stosunku do GP bez uzupełniania wiedzą?

## 2. Wkład autora

Według mojej najlepszej wiedzy, wszystkie zawarte w rozprawie doktorskiej wyniki naukowe, streszczone krótko w Sekcji 1 niniejszej recenzji stanowią oryginalny dorobek jej autora.

Dorobek ten jest bardzo obszerny, znacznie przekraczający typowy zakres rozprawy doktorskiej. Rozwiązane problemy były trudne, wymagające znajomości zaawansowanych metod sztucznej inteligencji i ogólnej wiedzy informatycznej.

W szczególności, za wartościowe elementy i aspekty prezentowanych w dysertacji wyników autora uznać należy:

- Inwencje algorytmów stochastycznych wspieranych wiedzą, które mnożna klasyfikować jako memetyczne w szeroko pojętym znaczeniu. Ich zasadniczą ideą jest uwzględnianie dodatkowej wiedzy w różnej postaci w ewolucyjnych algorytmach syntezy programów.
- Zastosowanie uzyskanych rozwiązań do syntezy programów.
- Rygorystyczny i obszerny sposób weryfikacji inwencji algorytmicznych, głównie na drodze złożonych testów porównawczych.
- Duża wiedza, doświadczenie oraz intuicja w dziedzinie realizowanych badań.
- Dobrze dobrany wstęp do złożonej tematyki badań zawarty w rozdziałach 1 – 4 motywujący rozwiązywane zagadnienia oraz budujące ścieżkę przez state of the art wybranych zagadnień syntezy programów, ich formalnej weryfikacji oraz metod stochastycznych rozwiązywania problemów syntezy programów, w tym GP.

## 3. Poprawność

Sposób prezentacji wyników state of the art oraz oryginalnych wyników autora cechuje precyzja i kompletność. Wywody prowadzące do konstrukcji nowych heurystyk są logicznie poprawne i przekonywujące.

Materiał eksperymentalny jest opracowany rzetelnie, zgodnie z zasadami i metodyką planowania i opracowywania wyników metodami statystyki matematycznej. Wyniki są przedstawione w dobrze opracowanych zestawieniach tabelarycznych wraz z odpowiednimi syntetycznymi komentarzami.

W recenzowanej rozprawie nie znalazłem błędów merytorycznych które mogły by mieć wpływ na ocenę jej wyników.

## 4. Wiedza kandydata

Opiniowana dysertacja w Rozdziałach 2 – 4 zawiera obszerne wprowadzenie do tematyki badań zawierające równocześnie state of the art. przedmiotowej dziedziny. W Rozdziale 2 omawia się podstawowe definicje i własności syntezy programów, a także metody syntezy sterowanej syntaktyką języka programowania (SyntGuS). W Rozdziale 3 omówione zostały metody formalnej weryfikacji programów, w szczególności metody testów oraz metody klasy SAT i SMT oraz związane z nimi narzędzia software. W Rozdziale 4 omówione zostały typy stochastycznych algorytmów populacyjnych optymalizacji globalnej które stosowane są do syntezy programów, głównie wykorzystujące mechanizmy ewolucyjne, dokładniej warianty Genetic Programming (GP). Omówiono także reprezentacje programów w postaci drzew składniowych, operatory mieszania (mutacje, krzyżowania) oraz typy selekcji dla zadań jedno- i wielokryterialnych dedykowane do rozwiązywania zadań syntezy programów (w tym selekcji leksykalnej).

Dodatkowo we wstępach do Rozdziałów 5 – 8 podano bardziej szczegółowy state of the art dla każdej z opracowanych przez autora metod.

Obszerny wstęp merytoryczny opisany powyżej oraz kompetentny opis inwencji algorytmicznych oraz wyników ich badania świadczą o szerokiej, ugruntowanej wiedzy doktoranta z zakresu automatycznej syntezy programów, metod ewolucyjnych tej syntezy oraz bardzo wielu powiązanych obszarów sztucznej inteligencji jak również informatyki teoretycznej (np. formalna weryfikacja programów, teoria języków formalnych).

Wszystkie wymieniane powyżej wypowiedzi autora opierają się o bardzo bogatą, aktualna literaturę z dziedziny cytowaną w rozprawie, obejmującą 200 pozycji.

## 5. Inne uwagi[1]

Brak.

---

[1] Opcjonalnie

## 6. Podsumowanie

Biorąc pod uwagę opinie zaprezentowane w poprzednich punktach recenzji i wymagania zdefiniowane przez artykuł 13 Ustawy z dnia 14 marca 2003 r. o stopniach naukowych i tytule naukowym (z późniejszymi zmianami)[2] moja ocena rozprawy pod względem trzech podstawowych kryteriów jest następująca:

**A.** Czy rozprawa zawiera oryginalne rozwiązanie problem naukowego? (wybierz jedną opcję stawiając znak **X**)

| [X] | [ ] | [ ] | [ ] | [ ] |
|---|---|---|---|---|
| *Zdecydowanie TAK* | *Raczej TAK* | *Trudno powiedzieć* | *Raczej NIE* | *Zdecydowanie NIE* |

**B.** Czy po przeczytaniu rozprawy zgadzasz się, że kandydat posiada ogólną wiedzę teoretyczną w dyscyplinie Informatyka techniczna i telekomunikacja?

| [X] | [ ] | [ ] | [ ] | [ ] |
|---|---|---|---|---|
| *Zdecydowanie TAK* | *Raczej TAK* | *Trudno powiedzieć* | *Raczej NIE* | *Zdecydowanie NIE* |

**C.** Czy kandydat posiada umiejętność samodzielnego prowadzenia pracy naukowej?

| [X] | [ ] | [ ] | [ ] | [ ] |
|---|---|---|---|---|
| *Zdecydowanie TAK* | *Raczej TAK* | *Trudno powiedzieć* | *Raczej NIE* | *Zdecydowanie NIE* |

Ponadto, biorąc pod uwagę:

- Stopień zaawansowania, trudności oraz duży zakres podjętego tematu rozprawy.
- Bardzo wysoki poziom uzyskanych wyników naukowych oraz świeże, ciekawe pomysły idące w kierunku algorytmiki memetycznej.
- Doskonałe wyniki publikacyjne. Publikacje w jednym z najważniejszych czasopism z dziedziny obliczeń ewolucyjnych (*Evolutionary Computation*, 2018, IF=3.469) oraz w materiałach pięciu wysoko ocenianych konferencji (IJCAI'18 CORE A*, GECCO 2017, 2018, 2019 CORE A, EuroGP 2017 CORE B).

rekomenduję wyróżnienie rozprawy doktorskiej[3].

*Podpis*

[2] http://www.nauka.gov.pl/g2/oryginal/2013_05/b26ba540a5785d48bee41aec63403b2c.pdf

[3] Oczywiście to zdanie jest opcjonalne.