Politechnika Poznańska

Wydział Informatyki
Instytut Informatyki

Praca zaliczeniowa

Inżynieria Kreatywności

Problem: **Efektywna metoda automatycznego programowania**

Prowadzący:
Prof. dr hab. Czesław CEMPEL

Doktorant:
mgr inż. Tomasz PAWLAK
tomasz.pawlak@cs.put.poznan.pl

Opiekun:
dr hab. inż. Krzysztof KRAWIEC, prof. PP
krzysztof.krawiec@cs.put.poznan.pl

Poznań, czerwiec 2012r.

Spis treści

1. Wprowadzenie do problemu

Od czasu zbudowania pierwszej maszyny programowalnej, złożoność tworzonego oprogramowania nieustannie rośnie. Jest to efekt wprowadzania nowych technologii, umożliwiających łatwe łączenie w spójną całość gotowych modułów programowych, jak i długiego czasu życia oprogramowania, sięgającego obecnie już kilkudziesięciu lat w wielu przypadkach (np.: LaTeX [1], Windows [2] , Unix [3] i wiele innych). Złożoność ta powoduje znaczne wydłużenie czasu wytwarzania i utrzymywania oprogramowania. W efekcie, w drugiej połowie XX wieku, w świecie akademickim, pojawiły się pierwsze próby badań nad metodami automatycznego wytwarzania oprogramowania. Jednak nadal, osiągi automatycznych metod rzadko przewyższają programy stworzone przez człowieka, i co ważniejsze efekty ich pracy nie są wystarczająco zadowalające w przypadku dużych zadań.

W ogólności metody automatycznego programowania, w tym m.in. programowanie genetyczne [4, 5, 6], automatyczna synteza programów deklaratywnych [7, 8, 9] i programów imperatywnych [10] mają jedną wspólną cechę – operują na kodzie programu, a zadanie stawiają w kategoriach oczekiwanego wyjścia programu. Są to dwie zupełnie różne dziedziny, pomiędzy którymi nie istnieje prosta relacja odwzorowująca oczekiwane wyjście w kod programu.

W ostatnich latach, w tej dziedzinie pojawił się nowy trend semantycznego programowania genetycznego [11, 12, 13, 14]. Jest to zbiór metod operujący na częściowych lub końcowych wynikach pracy programu i uzależniający kierunek poszukiwań optimum właśnie w zależności od tych wyników. Zauważalnym jest wyraźne skupienie prac badaczy tej dziedziny nad operatorami krzyżowania semantycznego, tj. metody łączenia dwóch programów w inny, z uwzględnieniem ich wyjścia lub wyników częściowych obliczeń. Wstępne efekty opisywanego podejścia są bardzo obiecujące i wyraźnie lepsze od wcześniejszych rozwiązań.

2. Definicja problemu i jego wielostronne określenie

Problem związany jest ze stworzeniem efektywnej metody automatycznego programowania. Wyniki ostatnich badań kładą nadzieję w metodach semantycznych. Jednak metody te związane są z nietrywialną i nieprzewidywalną relacją pomiędzy semantyką programu, a jego kodem źródłowym. W szczególności trudne do przewidzenia jest, jak drobna zmiana syntaktyczna wpłynie na semantykę całego programu, oraz, co trudniejsze, problem odwrotny, czyli odpowiedź na pytanie, jaki kod programu generuje zadane wyjście lub jego element.

2.1. Analiza problemu metodą SIMPLEX

Metoda SIMPLEX [15] jest ośmioetapową procedurą wspomagającą wszystkie fazy twórczego myślenia – od odkrycia problemu, poprzez jego zdefiniowanie, stworzenie pomysłów rozwiązania problemu, a kończąc na jego wdrożeniu.

2.1.1. Etap 1: Odkrycie problemu

Problemem jest brak dostatecznie efektywnej metody automatycznego programowania, tzn. metody, która spełnia w sposób zadowalający, wymagania jakości generowanych rozwiązań i jednocześnie minimalizuje swój czas pracy.

2.1.2. Etap 2: Odkrywanie faktów

- Duże koszty obliczeniowe metod automatycznego programowania,
- nieograniczona przestrzeń możliwych do stworzenia programów,

- brak metody gwarantującej znalezienie optymalnego rozwiązania w ogólności,
- (obecnie) brak metody gwarantującej znalezienie optymalnego rozwiązania w czasie wielomianowym przy założeniu ograniczeń na przestrzeń programów,
- relatywnie dobre wyniki metod semantycznych w stosunku do syntaktycznych,
- słabe rozumienie zależności między syntaktyką, a semantyką programu,
- ograniczenia na dostępność sprzętu badawczego.

2.1.3. Etap 3: Definicja problemu

Dlaczego koszty obliczeniowe metod automatycznego programowania są tak wysokie?
Ponieważ każdy wygenerowany program musi zostać uruchomiony w celu oceny. W zależności od przyjętych założeń, program może wykonywać się bardzo długo lub nawet nie gwarantować swojego zakończenia (problem stopu jest nierozstrzygalny [16]).

Dlaczego przestrzeń możliwych programów jest nieograniczona?
Wynika to wprost z możliwej długości programu. Rozmiar przestrzeni programów jest zdefiniowany przez zbiór dostępnych instrukcji i maksymalną długość programu. Jeśli długość ta nie jest określona, przestrzeń programów jest nieograniczona.

Dlaczego nie ma metody gwarantującej znalezienie optymalnego programu?
Jeśli przestrzeń programów jest nieograniczona i aktualnie znalezione najlepsze rozwiązanie ma wartość funkcji celu inną niż teoretyczne optimum, to nie można określić, czy jest ono globalnie najlepsze. W przeciwnym wypadku jeśli przestrzeń programów jest ograniczona, nie istnieje algorytm znajdujący lub konstruujący rozwiązanie optymalne w czasie wielomianowym. Przeszukanie całej przestrzeni programów ma z definicji złożoność wykładniczą od wielkości instancji problemu.

Dlaczego metody semantyczne sprawują się lepiej od syntaktycznych?
Metody semantyczne operują w mniejszym lub większym stopniu na wyjściu programu, a więc w dziedzinie, w której postawione jest zadanie indukcji programu. Metody syntaktyczne natomiast działają bardziej ślepo, nie są one wstanie przewidzieć jak drobna zmiana składniowa wpłynie na wyniki pracy programu, a zasada ich działania zwykle sprowadza się do minimalizowania wielkości zmian syntaktycznych w pojedynczym ruchu przeglądania przestrzeni rozwiązań.

Co wpływa na ograniczenie dostępu do sprzętu badawczego?
Nieprzewidywalność efektów zmian jakościowych powoduje, że wszelkie, nawet najdrobniejsze modyfikacje wymagają przeprowadzenia eksperymentów obliczeniowych w celu ich oceny i analizy. Z powodu stochastycznego charakteru wielu metod automatycznego programowania, analiza ta często ma charakter czysto statystyczny, a więc wymaga wielokrotnego powtarzania obliczeń. Wszystko to przekłada się na duże czasy obliczeń, które można zredukować jedynie poprzez zrównoleglenie pracy pomiędzy wiele komputerów. Niestety dostępne zasoby sprzętowe są często ograniczone.

Czy można jednocześnie zredukować koszty obliczeń i wykorzystać potęgę metod semantycznych?
Wyniki ostatnich badań [niepublikowane] pokazują, że tak, jednak spadek kosztów obliczeniowych jest nadal niewielki.

2.1.4. Etap 4: Znalezienie pomysłów

Pomysł 1: Stworzenie operatora krzyżowania geometrycznego
Pomysł polega na zbudowaniu operatora krzyżowania wykorzystywanego przez programowanie genetyczne. Operator ten otrzymywałby na wejściu parę istniejących programów –

rodziców, a jego zadaniem byłoby wyprodukować dwa semantycznie pośrednie pomiędzy nimi programy – dzieci. O operatorze krzyżowania mówi się, że jest geometryczny jeśli każdy stworzony przez niego potomek znajduje się na linii geodezyjnej (ang. *geodesic*, dla przestrzeni Euklidesowej – odcinek) łączącej obu rodziców. Obecnie został już zaproponowany taki operator [14], a jego właściwości zostały przeanalizowane [17].

Pomysł 2: Stworzenie operatora mutacji geometrycznej

Pomysł polega na stworzeniu operatora mutacji dla programowania genetycznego. Operator mutacji otrzymuje jako wejście program, a jego zadaniem jest wprowadzenie niewielkich zmian w programie, tak aby jednocześnie zbadać nowy fragment przestrzeni poszukiwań rozwiązania i zapewnić dywersyfikację populacji rozwiązań w przestrzeni rozwiązań. O operatorze mutacji mówi się, że jest geometryczny jeśli programy, które produkuje znajdują się w pewnym niewielkim sąsiedztwie programu wejściowego, ograniczonym przez kulę o zadanym promieniu. Należy zwrócić uwagę, że „kształt" kuli jest zależny od przyjętej metryki przestrzeni. Obecnie został już zaproponowany taki operator [niepublikowane], jednak jego analiza nie jest kompletna.

Pomysł 3: Wykorzystanie algorytmów zachłannych wyposażonych w relację sąsiedztwa zdefiniowaną w przestrzeni semantyk

Pomysł polega na wykorzystaniu algorytmów lokalnego przeszukiwania np.: *greedy* [18] i *steepest* [18] lub algorytmów metaheurystycznych takich jak *przeszukiwanie tabu* [18] lub *symulowane wyżarzanie* [18] do przeszukiwania przestrzeni semantyk programów oraz wykorzystania biblioteki procedur odwzorowującej semantyki programów na wcześniej wygenerowane procedury.

2.1.5. Etap 5: Selekcja i ocena pomysłów

Pierwszy i drugi pomysł opierają się o wykorzystanie obecnie najefektywniejszego narzędzia automatycznego programowania – programowania genetycznego. Programowanie genetyczne jest bardzo ogólnym schematem postępowania opartym o mechanizm obliczeń ewolucyjnych, w którym istnieje wiele modyfikowalnych elementów. Takimi elementami są np.: operatory krzyżowania i mutacji rozwiązań.

Pomysł zaprojektowania operatora krzyżowania wynika wprost z ogólnego przeświadczenia, że to ten operator głównie jest odpowiedzialny za postęp algorytmu. W ogólności jest to prawda, jednak istnieją prace pokazujące, że w pewnych sytuacjach może zostać wyparty przez mutację [19]. Istniejące wyniki eksperymentów [14, 17] potwierdzają, że taki operator ma zastosowanie i przewyższa tradycyjne metody.

Drugi pomysł jest związany ściśle z operatorem mutacji. Operator ten, w wydaniu semantycznym, operuje nie na składni programu, ale na wektorach jego pośrednich i końcowych semantyk, efektywnie zmieniając wyjście programu w niewielki sposób. Tradycyjne operatory działające syntaktycznie mogą powodować ogromne i nieprzewidywalne zmiany wyjścia programu, przy nawet niewielkich zmianach w jego wnętrzu. Z tej obserwacji wynika spodziewana i sprawdzona eksperymentalnie skuteczność proponowanego rozwiązania [niepublikowane] dorównująca wcześniej opisanemu operatorowi krzyżowania.

Ostatni pomysł jest efektem wyekstrahowania operatora mutacji poza proces ewolucyjny. Operator ten w swoim działaniu jest bardzo podobny do algorytmów lokalnego przeszukiwania – greedy i steepest, jednak działa bardziej losowo. Pomysł polega nie na zastosowaniu mechanizmu ograniczenia odległości semantycznej programu wyjściowego od programu wejściowego mutacji, ale do zdefiniowania relacji sąsiedztwa we wspomnianych algorytmach. Oba te algorytmy startują z pewnego, danego z góry lub wylosowanego, roz-

wiązania (programu), a następnie iteracyjnie przeglądają jego sąsiedztwo, przechodząc do lepszego rozwiązania. Sposób wyboru rozwiązania z sąsiedztwa jest różny w obu algorytmach. Algorytm kończy się, kiedy w sąsiedztwie nie istnieje lepsze rozwiązanie. Należy zwrócić uwagę, że taki algorytm nie gwarantuje znalezienia globalnego optimum.

Tabela 1. Macierz selekcji i oceny pomysłów wg metody SIMPLEX. Skala: 0 – najgorsza ocena, 10 – najlepsza.

	Pomysł 1 Op. krzyżowania	**Pomysł 2** Op. mutacji	**Pomysł 3** Alg. zachłanne
Łatwość implementacji	6	8	10
Koszty obliczeniowe uruchomienia	2	2	7
Dostosowanie do istniejących rozwiązań	10	10	4
Przewidywana jakość rozwiązań	8	9	9
SUMA:	**26**	**29**	**30**

Z podsumowania punktowego wynika, że najlepszym rozwiązaniem jest zastosowanie algorytmu zachłannego. Świadczą o tym przede wszystkim przewidywane dobrej, co najmniej takiej jak w przypadku mutacji, jakości wynikowe programy oraz prostota implementacji podejścia. Drugie miejsce, z wynikiem zaledwie o 1 punkt gorszym od pierwszego, otrzymał pomysł 2, związany z operatorem mutacji geometrycznej. Rozwiązanie to tworzy programy bardzo dobrej jakości [niepublikowane] oraz dobrze wpisuje się w istniejącą, wiodącą, metodologię programowania genetycznego.

2.1.6. Etap 6: Planowanie wdrożenia

- Projekt – zastosowanie algorytmów zachłannych wyposażonych w semantyczną relację sąsiedztwa,
- implementacja – łatwa implementacja w języku Java z wykorzystaniem istniejącego mechanizmu indeksowania i przeszukiwania przestrzeni semantyk,
- eksperyment obliczeniowy – porównanie z istniejącymi rozwiązaniami semantycznymi (krzyżowanie i mutacja), algorytmami zachłannymi nieposiadającymi informacji semantycznej o problemie oraz ze standardowym podejściem – kanonicznym programowaniem genetycznym,
- analiza wyników – analiza statystyczna, porównawcze testy statystyczne,
- publikacja wyników – publikacja wyników w artykule naukowym.

2.1.7. Etap 7: Sprzedanie pomysłu

Zastosowanie podejścia zachłannego ma wiele plusów. Przede wszystkim metody zachłanne są relatywnie szybkie i zwykle już po kilku sekundach pracy znajdują lokalne optimum w przestrzeni rozwiązań problemu. Jeśli rozbudować takie metody o rozszerzenia metaheurystyczne, to kosztem niewielkiego wydłużenia czasu obliczeń można uzyskać jeszcze lepsze rezultaty. Pewnym specyficznym rozszerzeniem jest algorytm symulowanego wyżarzania, dla którego zostało udowodnione, że przy pewnej wartości temperatury początkowej i funkcji ochładzania metoda zbiega się do globalnego optimum [20].

Uzupełnienie metody zachłannej o semantyczną relację sąsiedztwa ma dobre uzasadnienie w postaci wyników eksperymentalnych, wcześniej zweryfikowanych podczas prac nad operatorem krzyżowania geometrycznego i mutacji geometrycznej.

W efekcie zastosowania zaproponowanej metody, stałoby się możliwe rozwiązywanie jeszcze trudniejszych instancji zadań automatycznego programowania niż do tej pory. Głów-

nie ze względu na zmniejszenie czasu trwania obliczeń, jak i przypuszczalne poprawienie zbieżności całego algorytmu.

2.1.8. Etap 8: Działanie

Cykl metody SIMPLEX został zamknięty, wybrane rozwiązanie oczekuje na swoją implementację i próbę wdrożenia. W przypadku powodzenia, stanie się ono podstawą do przyszłych rozważań i być może rozpocznie kolejny cykl pracy metody SIMPLEX.

2.2. Analiza problemu metodą PMI

Metoda PMI [21] zakłada rozpatrzenie problemu z wielu punktów widzenia i w każdym przypadku określenie plusów i minusów możliwych rozwiązań oraz implikacji zastosowania danego rozwiązania. Dla uproszczenia zakłada się, że istnieją 3 możliwe rozwiązania problemu, takie same, jak wcześniej wypracowane metodą SIMPLEX, tj.:

- krzyżowanie geometryczne,
- mutacja geometryczna,
- algorytmy zachłanne wyposażone w semantyczną relację sąsiedztwa.

2.2.1. Naukowiec – twórca metody

1. Krzyżowanie geometryczne
   a. ***Dobrze:*** dobrze ugruntowana metoda, powszechne przeświadczenie o słuszności, obiecujące wyniki eksperymentalne,
   b. ***Źle:*** duża złożoność implementacji, wysokie koszty oceny implementacji,
   c. ***Implikacje:*** dobre efekty spowodują skupienie prac nad metodami semantycznymi.
2. Mutacja geometryczna
   a. ***Dobrze:*** obiecujące wyniki eksperymentalne,
   b. ***Źle:*** mutacja nie jest traktowana jako główny mechanizm napędowy programowania genetycznego,
   c. ***Implikacje:*** kontrowersyjne wyniki, trudność w przekonaniu innych.
3. Algorytm zachłanny
   a. ***Dobrze:*** spodziewane dobre wyniki eksperymentalne,
   b. ***Źle:*** oderwanie od dominujących trendów,
   c. ***Implikacje:*** kontrowersyjne wyniki, trudność w przekonaniu innych.

2.2.2. Naukowiec – weryfikator metody

1. Krzyżowanie geometryczne
   a. ***Dobrze:*** łatwość zrozumienia, powszechne przeświadczenie o słuszności zastosowania krzyżowania, obiecujące wyniki eksperymentalne,
   b. ***Źle:*** duża złożoność implementacji, wysokie koszty oceny implementacji,
   c. ***Implikacje:*** może postawić w złym świetle dotychczasowe rozwiązania.
2. Mutacja geometryczna
   a. ***Dobrze:*** łatwość zrozumienia, obiecujące wyniki eksperymentalne,
   b. ***Źle:*** kontrowersyjne wyniki w porównaniu z krzyżowaniem,
   c. ***Implikacje:*** konieczność weryfikacji eksperymentalnej.
3. Algorytm zachłanny
   a. ***Dobrze:*** łatwość weryfikacji,
   b. ***Źle:*** kontrowersyjne wyniki w porównaniu z programowanie genetycznym,

c. ***Implikacje:*** konieczność weryfikacji eksperymentalnej.

2.2.3. Zewnętrzny programista

1. Krzyżowanie geometryczne
   a. ***Dobrze:*** dobre wyniki, wpisuje się w istniejące środowiska pracy,
   b. ***Źle:*** duża złożoność implementacji, trudność weryfikacji poprawności,
   c. ***Implikacje:*** osiągnięcie zadowalających wyników w większości typowych zadań.
2. Mutacja geometryczna
   a. ***Dobrze:*** dobre wyniki, wpisuje się w istniejące środowiska pracy,
   b. ***Źle:*** trudność weryfikacji poprawności implementacji,
   c. ***Implikacje:*** brak zrozumienia, dlaczego mutacja zachowuje się tak dobrze, potencjalnie marnotrawstwo mocy obliczeniowej.
3. Algorytm zachłanny
   a. ***Dobrze:*** łatwość implementacji,
   b. ***Źle:*** *brak*,
   c. ***Implikacje:*** stworzenie szybkiej i efektywnej implementacji oraz upublicznienie jej.

2.2.4. Wdrożeniowiec

1. Krzyżowanie geometryczne
   a. ***Dobrze:*** dobre wyniki, łatwość połączenia z istniejącymi rozwiązaniami,
   b. ***Źle:*** *brak*,
   c. ***Implikacje:*** osiągnięcie zadowalających klienta rozwiązań w typowych zadaniach.
2. Mutacja geometryczna
   a. ***Dobrze:*** dobre wyniki, łatwość połączenia z istniejącymi rozwiązaniami,
   b. ***Źle:*** *brak*,
   c. ***Implikacje:*** osiągnięcie zadowalających klienta rozwiązań w typowych zadaniach.
3. Algorytm zachłanny
   a. ***Dobrze:*** niezależny moduł,
   b. ***Źle:*** brak powiązania z istniejącymi środowiskami,
   c. ***Implikacje:*** konieczność stworzenia powiązań z innymi modułami oraz ścieżki przepływu informacji między nimi.

2.2.5. PMI – podsumowanie

Operator krzyżowania ma sporo zalet, do największych z nich należy dobre powiązanie z istniejącymi środowiskami pracy oraz ogólne przeświadczenie o skuteczności tego operatora w zadaniach programowania genetycznego. Nie jest on jednak bez wad, do największych z nich należy niewątpliwie trudność implementacji. Ponadto nie gorsze wyniki są osiągane przez operator mutacji, którego innymi zaletami jest prostsza implementacja niż w przypadku krzyżowania oraz zgodność z istniejącymi środowiskami.

Z drugiej strony pojawia się rozwiązanie bazujące na algorytmach zachłannych, które stanowi niezależny moduł, prosty do wdrożenia, jednak wymagający integracji z innymi systemami. Rozwiązanie to ponadto charakteryzuje się wysoką skutecznością i niskimi kosztami obliczeniowymi, a jego implementacja jest łatwa. Wydaje się, że zbiór zalet tej metody zdecydowanie przewyższa zalety wymienionych operatorów programowania genetycznego, a jej wady wydają się nieistotne. Dlatego ostatnie rozwiązanie zostało wybrane jako proponowane do wdrożenia.

3. Podsumowanie

W pracy przedstawiono problem słabej zbieżności metod programowania automatycznego, ograniczającej bezpośrednio możliwe jego aplikacje do zdecydowanie prostszych zadań niż oprogramowanie tworzone dziś przez człowieka. W efekcie przeprowadzonej analizy metodą SIMPLEX i później zweryfikowanej przy użyciu metody PMI, zostało wypracowane i wybrane rozwiązanie oparte o algorytm zachłanny, wyposażony w relację sąsiedztwa uwzględniającą różnice semantyczne pomiędzy programami.

Oczekuje się, że w efekcie prowadzonych badań uda się wypracować metodę automatycznego programowania, gwarantującą zbieżność do globalnego optimum w z góry ograniczonym czasie. Lepsza zbieżność ma bezpośredni wpływ na zmniejszenie kosztów obliczeniowych algorytmu, co wiąże się z możliwością rozwiązania bardziej złożonych problemów, niż dotychczas oraz syntezą potencjalnie większych programów. Ponadto należy zauważyć, że istniejące rozwiązania, w tym programowanie genetyczne, często znajdują rozwiązanie suboptymalne. Oczekuje się, że wybrane podejście semantyczne pozwoli łatwiej odkrywać algorytmowi rozwiązania optymalne.

Należy spodziewać się, że przestrzeń semantyk programów, wraz ze zdefiniowaną na niej metryką i relacją sąsiedztwa gwarantuje istnienie globalnej wypukłości krajobrazu dopasowania rozwiązań (ang. *fitness landscape*), wynikającego z konkretnej instancji zadania automatycznego programowania. Istnienie (idealnej) globalnej wypukłości daje gwarancję zbieżności do globalnego optimum większości istniejących algorytmów przeszukiwania.

W kontekście sukcesu prowadzonych badań, można w dalekiej przyszłości rozważyć tak szeroki rozwój metod automatycznego programowania, że będą one wstanie samodzielnie tworzyć oprogramowanie, nad którym obecnie musi pracować człowiek. W efekcie zawód programisty może zostać w dłuższej perspektywie wyparty przez operatora procesu programistycznego.

4. Literatura

[1] L. Lamport, LaTeX: A Document Preparation System, Massachusetts: Addison-Wesley, Reading, 1986.

[2] „Wikipedia - Microsoft Windows,” Wikimedia Foundation, [Online]. Available: http://en.wikipedia.org/wiki/Microsoft_Windows. [Data uzyskania dostępu: 9 czerwiec 2012].

[3] „Wikipedia - Unix,” Wikimedia Foundation, [Online]. Available: http://en.wikipedia.org/wiki/Unix. [Data uzyskania dostępu: 9 czerwiec 2012].

[4] J. R. Koza, „Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems,” Stanford, 1990.

[5] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, Cambridge: MIT Press, 1992.

[6] R. Poli, W. B. Langdon and N. F. McPhee, A Field Guide to Genetic Programming, http://lulu.com, 2008.

[7] Z. Manna and R. J. Wildinger, "Knowledge and Resoning in Program Synthesis," *Artificial Intelligence,* vol. 6, no. 2, pp. 175-208, 1975.

[8] L. Siklossy i D. Sykas, „Automatic Program Synthesis from Example Problems,” w *IJCAI*, 1975.

[9] M. K. Kamani and R. S. Ramakrishna, "Predicate-Formation for Synthesizing LISP Code," in *Systems, Man and Cybernetics, IEEE Transactions* , 1990.

[10] F. LeGland, S. Antipolis i A. Gondel, „Systematic Numerical Experiments in Nonlinear Filtering with Automatic Fortran Code Generation," Institute for Systems Research Technical Reports, 1986.

[11] K. Krawiec i B. Wieloch, „Analysis of Semantic Modularity of Genetic Programming," *Foundations of Computing and Decision Sciences,* tom 34, nr 4, pp. 265-285, 2009.

[12] N. Q. Uy, N. X. Hoai, M. O'Neill and B. McKay, "The Role of Syntactic and Semantic Locality of Crossover in Genetic Programming," in *PPSN 2010 11th International Conference on Parallel Problem Solving From Nature*, Kraków, 2010.

[13] T. Pawlak, *Automated Design of Semantically Smooth Instruction Spaces for Genetic Programming,* Poznań, 2011.

[14] K. Krawiec and T. Pawlak, "Locally Geometric Semantic Crossover," in *Genetic and Evolutionary Computation Conference*, Philadelphia, 2012.

[15] P. Wilson, "Simplex Creative Problem Solving," *Creativity and Innovation Management,* vol. 6, no. 3, pp. 161-167, 1997.

[16] A. Turing, "On Computable Numbers, with an application to the Entscheidungsproblem," in *Procedings of the London Mathematical Society*, London, 1936.

[17] K. Krawiec and T. Pawlak, "Quantitative Analysis of Locally Geometric Semantic Crossover," in *Proceedings of Parallel Problem Solving from Nature 2012* , Taormina, 2012.

[18] S. Luke, Essentials of Metaheuristics, Lulu, 2009.

[19] D. White and S. Poulding, "Evolving a sort: Lessons in genetic programming," in *Proceedings of the 1993 International Conference on Neural Networks*, 2009.

[20] Granville, Krivanek and Rasson, "Simulated Annealing: A Proof of Convergence," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 16, 1994.

[21] J. Manktelow, MindTools: Essentail skills for an excellent career, Mind Tools Ltd, 2004.