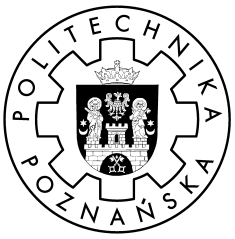
Politechnika Poznańska
Wydział Informatyki
Instytut Informatyki



Streszczenie rozprawy doktorskiej

HEURYSTYCZNE ALGORYTMY ODKRYWANIA FUNKCJI OCENY W PROBLEMACH OPARTYCH NA TESTACH

Paweł Liskowski

Promotor
prof. dr hab. inż. Krzysztof Krawiec

Poznań, 2018 r.

Rozdział 1

Wstęp

1.1 Motywacja

Pragnienie stworzenia inteligentnych maszyn towarzyszy ludzkości co najmniej od antyku. Pierwsze zapiski dotyczące inteligentnych bytów sięgają aż do starożytnej Grecji [28]. W dzisiejszych czasach *sztuczna inteligencja* (SI) to kwitnąca dziedzina mogąca pochwalić się wieloma wspaniałymi sukcesami, niesłabnącą popularnością wśród naukowców, a także licznymi praktycznymi zastosowaniami. Niewątpliwym znakiem tego ostatniego jest fakt, iż wiele procesów w przemyśle i nie tylko podlega obecnie niemal całkowitej automatyzacji. Inteligentne maszyny są odpowiedzialne za rozpoznawanie obrazów, mowy, a nawet wspomagają lekarzy w diagnostyce chorób oraz w zabiegach ratujących życie pacjentom. I choć może się to początkowo wydawać trudne do objęcia, wiele algorytmów i programów komputerowych realizujących te i podobne zadania cieszy się obecnie większą skuteczności aniżeli specjaliści w danej dziedzinie [32, 10, 25, 8, 31].

Niniejsza rozprawa dotyczy problemów, które wyłoniły się w *inteligencji obliczeniowej* (ang. computational intelligence [7]), gałęzi SI poświęconej metodom i algorytmom inspirowanym biologicznie. Chociaż biologiczne korzenie są bez wątpienia bardzo ważne, metody takie jak sztuczne sieci neuronowe czy algorytmy ewolucyjne są w swej istocie wyrafinowanymi modelami obliczeniowymi, które intensywnie wykorzystują element uczenia się z danych, aby nadać swemu zachowaniu inteligencji. Niestety problemy które ludzie rozwiązują intuicyjnie i niemal bez wysiłku, jak przykładowo rozpoznawanie i nazywanie obiektów, czy też rozumienie mowy, stanowią największe wyzwanie dla SI. Zadania te są często trudne o opisie formalnym, niezbędnym dla metod komputerowych. Co więcej, niejednokrotnie jedynym możliwym sposobem zdefiniowania problemu oraz opisania naszych intencji jest wcześniejsze przygotowanie zbioru *przykładów* (testów), które charakteryzują pożądane zachowanie, lub opisują własności idealnego rozwiązania. Innymi słowy, obszar inteligencji obliczeniowej obfituje w tzw. *problemy oparte na testach* (ang. test-based problems). Przykładami testów mogą być obrazy wraz z opisem ich zawartości, strategie gry w szachy, czy w końcu środowiska stosowane w robotyce celem oceny jakości zachowania jego podmiotu. W powyższych przykładach agent (rozwiązanie kandydackie) wchodzi w interakcję z testem oraz *uczy się* z wyniku tej interakcji: algorytm rozpoznawania obrazu uczy się cech niezbędnych do poprawnej klasyfikacji, gracz dopasowuje swoją strategię do przeciwnika, a wirtualny lub fizyczny robot aktualizuje swoją politykę zachowania w środowisku. Wspólnym ogniwem łączącym wszystkie te scenariusze jest to, że liczba testów jest często bardzo duża (lub nawet nieskończona), a interakcje z każdym z nich są w dużej mierze niezależne.

Istnieje wiele algorytmów przeznaczonych dla konkretnych klas problemów opartych na testach, np. dedykowane algorytmy przetwarzania i rozpoznawania obrazów, metody statystycznego ucze-

nia się dla problemów uczenia maszynowego lub algorytmy minimaksowe dla gier. Jednakże takie metody bazują na pewnych cechach charakterystycznych tych konkretnych klas problemów, nieujętych w definicji problemów opartych na testach. W niniejszej rozprawie skupimy się na dwóch ogólnych metodach rozwiązywania problemów opartych na testach – *algorytmach koewolucyjnych* oraz *programowaniu genetycznym*. Obie metody symulują *interakcję* pomiędzy kandydatami i testami, a następnie agregują wyniki wszystkich przeprowadzonych interakcji celem oceny jakości proponowanych rozwiązań problemu. Uzyskiwana w ten sposób skalarna ocena jakości jest głównym elementem odpowiedzialnym za ukierunkowanie przeszukiwania realizowanego w przestrzeni potencjalnych rozwiązań. W niniejszej rozprawie pokażemy, że takiemu schematowi postępowania towarzyszy problem *wąskiego gardła* (ang. evaluation bottleneck). W szczególności zaś wykażemy, że zjawisko to ma negatywny wpływ na zdolność do generalizacji algorytmów uczenia maszynowego, a w szczególności zwiększa ryzyko utknięcia w lokalnym optimum w trakcie optymalizacji. Główną motywacją dla przeprowadzonych badań jest chęć zminimalizowania tego problemu poprzez dostarczenie algorytmom uczącym się bogatszej informacji o zachowaniu rozwiązań kandydackich.

1.1.1 Cel pracy

W kontekście powyżej nakreślonej problematyki, głównym celem niniejszej rozprawy jest opracowanie rodziny algorytmów odpornych na problem wąskiego gardła funkcji oceny oraz zjawisk kompensacji wyników interakcji i utraty gradientu przeszukiwania, które stanowią jego bezpośrednią konsekwencję.

Cele szczegółowe pracy obejmują następujące zagadnienia:

1. Teoretyczna analiza problemu wąskiego gardła oceny oraz możliwych sposobów jego uniknięcia w kontekście algorytmów uczących się rozwiązujących problemy oparte na testach.

2. Opracowanie wielokryterialnej metody profili jakościowych (ang. performance profile) charakteryzującej rozwiązania kandydackie z wykorzystaniem testów o różnym stopniu trudności.

3. Wykazanie praktycznej przydatności powyższej metody do porównywania rozwiązań kandydackich uzyskanych różnymi algorytmami uczenia.

4. Zdefiniowanie pojęcia heurystycznego *kryterium przeszukiwania* (ang. search objective) zorientowanego na wzbogacenie procesu przeszukiwania o bogatszą informację charakteryzującą zachowanie rozwiązań kandydackich. Zademonstrowanie, w jaki sposób takie kryterium może być wykorzystane do ukierunkowania procesu przeszukiwania.

5. Opracowanie algorytmów służących automatycznej ekstrakcji heurystycznych funkcji oceny w trakcie rozwiązywania problemu. Poprzez wieloaspektową charakterystykę własności rozwiązań kandydackich, algorytmy te niwelują w praktyce negatywne konsekwencje wąskiego gardła funkcji oceny, tworząc jednocześnie użyteczny gradient w kierunku lepszych rozwiązań problemu.

6. Eksperymentalna weryfikacja zaproponowanych koncepcji i algorytmów na rzeczywistych problemach opartych na testach.

Rozdział 2

Problemy oparte na testach

2.1 Definicja

W teorii optymalizacji problem optymalizacyjny definiuje się poprzez określenie jego dziedziny oraz obiektywnej funkcji celu zdefiniowanej na jej elementach. Klasyczny problem optymalizacyjny polega na znalezieniu takiego elementu dziedziny, który maksymalizuje lub minimalizuje daną funkcję celu. Takie sformułowanie problemu pozwala skutecznie modelować wiele rzeczywistych problemów. Istnieją jednak problemy, w których wyznaczenie wartości funkcji celu w danym punkcie jest na tyle kosztowne obliczeniowo, że klasyczne metody optymalizacyjne wymagające wielokrotnego wywoływania funkcji celu okazują się być w praktyce zbyt kosztowne obliczeniowo, nawet dla najnowocześniejszych maszyn obliczeniowych. Problemy które charakteryzują się tym, iż jakość potencjalnego rozwiązania zależy od wyników kosztownych interakcji z elementami (zwykle dużego) zbioru testów nazywane są problemami opartymi na testach [5, 11]. Formalnie zdefiniujemy je następująco:

Definicja 2.1. Problem oparty na testach jest obiektem składającym się z elementów $(\mathcal{S}, \mathcal{T}, g)$ takich, że:

- $\mathcal{S}$ jest zbiorem kandydatów, nazywanych również rozwiązaniami kandydackimi (ang. candidate solutions),
- $\mathcal{T}$ jest zbiorem testów z elementami którego kandydaci wchodzą w interakcję,
- $g : \mathcal{S} \times \mathcal{T} \to \mathcal{O}$ jest funkcją interakcji, gdzie $\mathcal{O}$ jest całkowicie uporządkowanym zbiorem,
- pojęcia rozwiązania (ang. solution concept [9]), które określa cel przeszukiwania.

W odróżnieniu od tradycyjnych problemów optymalizacyjnych, problemy oparte na testach nie posiadają obiektywnej funkcji oceny. Jej substytutem jest funkcja interakcji g, a wyniki interakcji pomiędzy elementami $\mathcal{S}$ i $\mathcal{T}$ mogą zostać potraktowane jako wyznacznik jakości kandydatów. W niniejszej rozprawie zakładamy, że $\mathcal{S}$ i $\mathcal{T}$ są skończone, oraz że g jest funkcją deterministyczną. W ogólności gdy $\mathcal{T} = \{t_1\}$, dowolny problem oparty na testach sprowadza się do problemu optymalizacyjnego z funkcją celu $f(s, t_1) = g(s, t_1)$. Z kolei gdy $\mathcal{S} = \mathcal{T}$, mówimy o symetrycznym problemie opartym na testach, gdzie rola kandydatów i testów jest odgrywana przez te same obiekty.

Aby zilustrować powyższą definicję, rozważmy problem znalezienia najlepszej strategii gry dla czarnego gracza w grze w szachy. W takim przypadku $\mathcal{S}$ składać się będzie ze zbioru wszystkich możliwych strategii gry czarnego gracza, z kolei $\mathcal{T}$ zawierać będzie wszystkie strategie gry białego gracza. Funkcja interakcji g odpowiada natomiast rozegraniu pełnej partii w szachy pomiędzy

Algorithm 1 Ogólny schemat jednopopulacyjnego algorytmu koewolucyjnego. Rozwiązania kandydackie (elementy należące do S) pełnią jednocześnie rolę testów T, doskonale nadając się do symetrycznych problemów opartych na testach. Prezentowany schemat można z łatwością uogólnić na przypadek, w którym kandydaci i testy należy do osobnych populacji.

```
1: S ← Inicjalizuj populację
2: T ← Wybierz testy(S)
3: Oceń populację(S, T)
4: while ¬Warunek stopu() do
5:     S ← Wybierz rodziców(S)
6:     S ← Mutuj i krzyżuj kandydatów(S)
7:     T ← Wybierz testy(S)
8:     Oceń populację(S, T)
9: return Najlepszy kandydat(S)
```

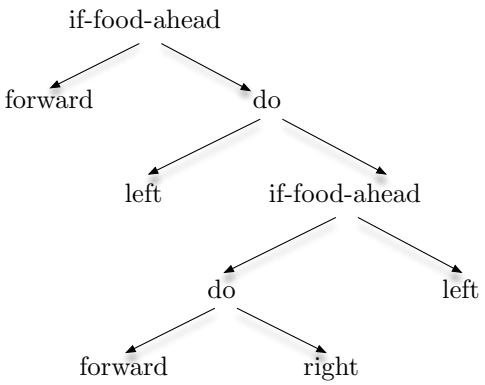
tymi graczami. Aby zidentyfikować najlepszą strategię, możemy poszukiwać kandydata, który maksymalizuje oczekiwany wynik interakcji z losowo wybranym testem. Tak zdefiniowane pojęcie rozwiązania nazywane jest często w literaturze *maksymalizacją oczekiwanej użyteczności* (ang. maximization of expected utility [9]).

Jak sugeruje powyższy przykład, problemy oparte na testach są często spotkane w kontekście gier. W rzeczywistości klasa problemów opartych na testach jest jednak znacznie szersza. Hipotezy uczenia maszynowego oraz programy podlegające ewolucji w programowaniu genetycznym są oceniane w kontekście wielu przykładów uczących. W celu optymalizacji zachowania kontrolera odpowiedzialnego za utrzymanie równowagi odwróconego wahadła lub znalezienia polityki sterowania robota mobilnego, wykonuje się wiele symulacji, które różnią się warunkami początkowymi oraz innymi parametrami. Ponadto, ilekroć ocena kandydatów ma charakter stochastyczny, ich jakość należy oceniać w kontekście wielu scenariuszy różniących się realizacją zmiennej losowej (zmiennych losowych) opisujących to środowisko. Przykładowo, symulatory stosowane do oceny wydajności robota (rozwiązania kandydującego) w środowisku (teście) mogą wykorzystywać losowość do emulowania skutków tarcia. Klasa problemów opranych na testach jest na tyle ogólna, że pozwala z łatwością modelować te oraz inne, znacznie bardziej skomplikowane problemy. Cena, którą należy zapłacić za tę ogólność, to brak wyspecjalizowanych algorytmów gwarantujących pozyskanie wysokiej jakości rozwiązań. W kolejnym podpunkcie przyjrzymy się zatem pokrótce dwóm popularnym metodom rozwiązywania problemów opartych na testach.

2.2 Metody rozwiązywania problemów opartych na testach

2.2.1 Algorytmy koewolucyjne

Kompetytywny algorytm koewolucyjny szczególnie dobrze nadaje się do rozwiązywania problemów opartych na testach, ponieważ w odróżnieniu od klasycznych algorytmów ewolucyjnych nie wymaga dostępu do obiektywnej funkcji celu, a jedynie do wyników interakcji pomiędzy osobnikami. Co więcej, algorytm ten operuje najczęściej na dwóch populacjach osobników, co odpowiada dwóm rolom obecnym w problemach opartych na testach: kandydatom i testom. Idea działania kompetytywnych algorytmów koewolucyjnych polega na rozwijaniu zbioru osobników (kandydatów) poprzez poddanie ich działaniu operatorów ewolucyjnych jak mutacja, krzyżowanie i selekcja. Kluczowym elementem algorytmów koewolucyjnych jest nieustający wyścig zbrojeń pomiędzy konkurującymi ze sobą populacjami osobników [27], co odpowiada koewolucji między gatunkami obserwowanej w przyrodzie.



(a)

```
if foodAhead then
  forward();
else
  left();
  if foodAhead then
    forward();
    right();
  else
    left();
```

(b)

Rysunek 2.1: Struktury drzewiaste są powszechnie stosowane do ewolucji programów komputerowych z wykorzystaniem metodyki programowania genetycznego. Przykładowy program dla problemu Artificial Ant (a), oraz jego implementacje w pseudo-kodzie (b). Procedury `forward()`, `right()` i `left()` mogą podlegać ewolucji lub być elementami zbioru dopuszczalnych instrukcji, z których budowane są rozwiązania kandydackie.

Typowy algorytm koewolucyjny utrzymuje populację kandydatów $S \subset \mathcal{S}$ oraz oddzielną populację testów $T \subset \mathcal{T}$. W każdym pokoleniu, każdy kandydat $s \in S$ wchodzi w interakcję z każdym testem $t \in T$, produkując wynik interakcji $g(s,t)$. Wyniki wszystkich interakcji stanowią podstawę do oceny osobników w S i T. Ogólnie rzecz biorąc, rozwiązania kandydackie są oceniane pod kątem ich skuteczność w rozwiązywaniu testów, a z kolei testy są nagradzane za ich *informatywność* rozumianą jako zdolność do rozróżniania kandydatów. W praktyce, wyniki interakcji są najczęściej agregowane do pojedynczej skalarnej wartości mającej za zadanie odzwierciedlić jakość kandydata/testu. Wskaźnik ten jest następnie wykorzystywany w procesie selekcji najbardziej obiecujących kandydatów i testów, na podstawie których tworzone są nowe, w idealnym przypadku lepsze populacje osobników. Proces uczenia się algorytmu koewolucjnego sprwowadza się zatem do odkrywania obszarów w przestrzeni rozwiązań zawierających coraz lepsze rozwiązania.

Najważniejsza zaleta metod koewolucyjnych wynika z ich ogólności. W przeciwieństwie do wielu wyspecjalizowanych algorytmów, które są dedykowane tylko pewnym podklasom problemów opartych na testach (np. przeszukiwanie drzewa gier metodą Monte Carlo), algorytm koewolucyjny może być zastosowany do dowolnego problemu opartego na testach, pod warunkiem odpowiedniej konfiguracji. Proces konfiguracji wymaga zdefiniowania funkcji interakcji oraz dostosowania operatorów wyszukiwania ewolucyjnego do reprezentacji wykorzystywanej przez osobników w S i T. W praktyce, algorytmy koewolucyjne okazują się być podatne na tzw. *koewolucyjne patologie*, będące często bezpośrednią konsekwencją wąskiego gardła funkcji oceny.

2.2.2 Programowanie genetyczne

Programowanie genetyczne (PG, [13]) jest wariantem obliczeń ewolucyjnych, zaprojektowanym w celu automatycznego rozwiązania całych klas problemów, bez konieczności wcześniejszego dostarczenia przez użytkownika formy lub struktury rozwiązania. Jest to kluczowa różnica koncepcyjna odróżniająca PG od innych metaheurystyk, które przeważnie operują na z góry określonej *reprezentacji* kandydatów. Programowanie genetyczne przeszukuje przestrzeń struktur danych, które można łatwo zinterpretować jako programy komputerowe (programy w skrócie). Programy mogą kodować dowolne procedury, a także kompletne algorytmy dla różnych zadań, w tym klasyfikacji, regresji, inżynierii cech itp. Koncepcja programu zapewnia niezrównany poziom elastyczności w

wyrażaniu rozwiązań praktycznie dla dowolnego problemu, niezależnie od tego, czy obejmuje on uczenie się, czy optymalizację. PG oferuje również skuteczny sposób przeszukiwania przestrzeni dopuszczalnych programów, opierając się na algorytmie ewolucyjnym, który zapożycza swoją mechanikę z ewolucji biologicznej. W dużym stopniu sukces PG przypisuje się efektywnemu połączeniu unikalnej reprezentacji kandydatów (programów) z paradygmatem obliczeń ewolucyjnych.

PG jest rozwijane od ponad 25 lat, może pochwalić się bogatą literaturą (ponad 11000 pozycji [18], w tym w Science [30] i Nature [19, 1]), oraz rozwiązaniem wielu trudnych problemów naukowych i praktycznych, w tym problemów w czystej matematyce, programowania komputerów kwantowych, modelowania złożonych zjawisk i systemów, wykrywania i automatycznego korygowania błędów w programach, automatycznej optymalizacji programów, i wielu innych (zob. przegląd w [29]).

Unikalną cechą algorytmu koewolucyjnego jest fakt, iż zbiór testów T zmienia się w czasie. Z kolei programowanie genetyczne doskonale sprawdza się w modelowaniu problemów opartych na testach w sytuacji gdy zbiór T jest stały i dany jako element definicji problemu, jak np. modelowanie cen nieruchomości w oparciu o dane historyczne.

Algorytm PG rozwiązujący problem oparty na testach utrzymuje populację programów kandydackich $P \subset \mathcal{P}$. W każdej iteracji, programy $p \in P$ wchodzą w interakcję z testami $(x, y) \in T$, podczas której program p jest uruchomiony z danymi wejściowymi reprezentowanymi przez x, zwracając wyjście oznaczane jako $p(x)$. Jeżeli $p(x) = y$, powiemy, że p rozwiązuje test i $g(p(x), y) = 1$. Jeżeli zaś $p(x) \neq y$, wówczas $g(p(x), y) = 0$ i powiemy, że p nie przechodzi testu (x, y). Taka definicja funkcji interakcji jest szczególnie użyteczna w domenach, w których programy zwracają wartości dyskretne. Dla domen ciągłych naturalną funkcją interakcji jest błąd absolutny lub średniokwadratowy.

Biorąc pod uwagę powyższą dyskusję, konwencjonalna miara jakości programów stosowana w PG może być wyrażona w środowisku problemów opartych na testach jako liczba rozwiązanych testów:

$$f_T(p) = |\{t \in T : g(p, t) = 1\}|. \tag{2.1}$$

Należy zauważyć, że takie sformułowanie dokładnie odpowiada maksymalizacji oczekiwanej użyteczności, tj. pojęciu rozwiązania stosowanemu także w przypadku koewolucji. Podobnie jak w przypadku koewolucji, tak i w PG funkcja oceny f_T 'kompresuje' bogatą informację opisującą zachowanie kandydata na wszystkich testach do skalarnej wartości, która okazuje się w praktyce kiepskim *sygnałem sterującym* procesem przeszukiwania przestrzeni rozwiązań.

Rozdział 3

Konsekwencje skalarnej funkcji oceny kandydatów

Główną motywacją dla tego rozdziału jest obserwacja, że nawyk stosowania konwencjonalnej, skalarnej funkcji oceny kandydatów znacznie ogranicza efektywność algorytmów ewolucyjnych. Poniżej przedstawiamy i pokrótce omawiamy konsekwencje konwencjonalnego podejścia do oceny rozwiązań kandydackich w problemach opartych na testach.

3.1 Wąskie gardło funkcji oceny

Algorytmy zaprojektowane do rozwiązywania problemów opartych na testach poruszają się po przestrzeni dopuszczalnych rozwiązań wykorzystując w tym celu m.in. funkcję oceny, która agreguje wyniki interakcji do skalarnej wartości mającej za zadanie odzwierciedlić jakość rozważanych kandydatów. Funkcję tę możemy zapisać w następujący sposób:

$$f_T(s) = \frac{1}{|T|} \sum_{t \in T} g(s,t). \tag{3.1}$$

Ocena jakości kandydata $s \in S$ sprowadza się zatem do zliczenia liczby rozwiązanych testów w T. Podobnie, konwencjonalna ocena jakości osobników w programowaniu genetycznym polega na zaaplikowaniu programu do wielu testów i agregowaniu obserwowanych różnic między faktycznym a pożądanym wynikiem programu.

Funkcja oceny, która zlicza liczbę rozwiązanych testów stanowi zwykle nieodłączną część problemu i umożliwia jednocześnie zastosowanie wielu klasycznych metod i algorytmów przeszukiwania. Takie sformułowanie jest nie tylko wygodne, ale także zgodne ze standardowym sposobem stawiania problemów w zakresie optymalizacji i uczenia maszynowego.

Z drugiej strony, ocena skalarna jest w istocie niezwykle prymitywna w swojej zbiorczej charakterystyce wyników interakcji. Mimo, iż niektóre rozwiązania w populacji mogą radzić sobie z niektórymi testami znacznie lepiej niż inne, to te indywidualne różnice często nie znajdują odzwierciedlenia w ogólnej ocenie, która informuje tylko o średniej jakości kandydata. Rozwiązania kandydackie najczęściej opisują bardzo złożone podmioty, jak np. programy komputerowe lub skomplikowane strategie zachowania, natomiast $f_T(s)$ zawiera bardzo mało informacji dotyczących ich własności oraz zachowania. Wszystko, co pozostaje z nietrywialnego procesu oceny, to pojedyncza wartość, która wskazuje liczbę rozwiązanych testów lub całkowity błąd na zbiorze testów. Problem wąskiego gardła jest znacznie bardziej powszechny, niż mogłoby się wydawać, ponieważ agregujące funkcje celu są nagminnie stosowane w praktyce.

Główną obserwacją motywującą badania przedstawione w niniejszej rozprawie jest obserwacja, że skalarna funkcja oceny narzuca nieuniknioną utratę informacji w trakcie procesu oceny kandydata, podczas gdy bogatsza charakterystyka zachowania kandydata w kontekście poszczególnych testów może pomóc w zwiększeniu efektywności przeszukiwania przestrzeni rozwiązań, zwłaszcza poprzez zapewnienie naturalnych środków różnicowania populacji kandydatów.

Agregacja wyników testów powoduje zatem *wąskie gardło oceny* w komunikacji między funkcją oceny a algorytmem wyszukiwania. Zaznaczmy jednak, że istnieją domeny, w których przeszukiwanie przy użyciu jedynie oceny skalarnej jest nieuniknione. Przykładem są problemy optymalizacji tzw. czarnej skrzynki (ang. black-box optimization), w których komunikacja ta ogranicza się do wymiany informacji o jakości kandydata [12, 2]. W takich problemach nie ma możliwości poszerzenia wąskiego gardła oceny poprzez dostarczenie algorytmowi bogatszej informacji na temat właśności kandydatów. Należy jednak podkreślić, że problemy czarnej skrzynki stanowią wyjątek aniżeli regułę. W innych domenach, w których szczegóły procesu oceny kandydatów nie są ukryte, a szczególnie w przypadku problemów opartych na testach, naturalne staje się pytanie: czy musimy skompresować wszystkie informacje o wynikach interakcji do jednej wartości skalarnej? Dlaczego nie wykorzystać tej oraz towarzyszących ocenie informacji, aby usprawnić proces przeszukiwania? Powyższa obserwacja jest kamieniem węgielnym opracowanej w dalszych rozdziałach metodyki służącej automatycznemu odkrywaniu heurystycznych funkcji celu, które mają za zadanie poszerzyć wąskie gardło funkcji oceny poprzez dostarczenie algorytmom bogatszej informacji charakteryzującej analizowane rozwiązania kandydackie.

3.2 Kompensacja wyników interakcji

Najpoważniejszą konsekwencją wąskiego gardła funkcji oceny jest *kompensacja* wyników interakcji: jeśli dwoje kandydatów rozwiązuje tę samą liczbę testów, to są one uważane za równoważne, niezależnie od tego, które testy zostały przez nie rozwiązane. W przeciwnym razie jeden z porównywanych kandydatów jest uznany za lepszego, ale zupełnie ignorując zachowania na poszczególnych testach. Zbiór testów jest z natury bardzo zdywersyfikowany, testy różnią się trudnością, mogą być mniej lub bardziej istotne z punktu widzenia rozwiązania danego problemu.

Na skutek zjawiska kompensacji rozwiązania kandydackie mogą uzyskać te samą wartość funkcji f_T, nawet jeśli wyniki ich interakcji z tymi samymi testami są całkowicie różne. To z kolei może sprawić, że będą one nierozróżnialne w fazie selekcji, co prowadzi do utraty różnorodności w populacji kandydatów.

W teorii problem kompensacji może zostać uniknięty jeśli kandydaci s_i i s_j zostaliby porównani z wykorzystaniem relacji dominacji:

$$s_i \succ s_j \iff \forall t \in T : g(s_i, t) \geq g(s_j, t) \wedge \exists t \in T : g(s_i, t) > g(s_j, t). \tag{3.2}$$

Relacja dominacji porównuje zachowanie kandydatów na poszczególnych testach i jest w tym sensie bardziej skrupulatna niż skalarna funkcja celu. Jest to jednak relacja częściowa, która w konsekwencji nie zapewnia użytecznego gradientu wyszukiwania gdy żaden z porównywanych kandydatów nie rozwiązuje nadzbioru testów względem drugiego kandydata [14]. Problem ten jest niestety powszechny: w przypadku dwóch niepowiązanych ze sobą rozwiązań jest znacznie bardziej prawdopodobne, że są one wzajemnie niezdominowane, niż to, że jedno z nich dominuje nad drugim, i to prawdopodobieństwo rośnie wraz z liczbą testów w T.

Ocena skalarna oraz relacja dominacji stanowią dwa ekstrema w analizie wyników interakcji pomiędzy elementami zbiorów S i T. Jedną z głównych motywacji niniejszej rozprawy jest chęć

wypracowania kompromisu pomiędzy tymi dwoma podejściami. W kolejnych rozdziałach pokażemy w jaki sposób użyteczna, wielokryterialna charakterystyka kandydatów może być uzyskana w pełni automatycznie, z wykorzystaniem metod nienadzorowanego uczenia się.

3.3 Utrata gradientu

Agregująca funkcja oceny f_T, która zlicza liczbę rozwiązanych testów jest również ofiarą zjawiska *utraty gradientu*, które ma miejsce w sytuacji gdy kilka kandydatów rozwiązuję tę samą liczbę testów. Ze względu na dyskretną naturę f_T taki scenariusz jest szczególnie prawdopodobny gdy liczba testów jest relatywnie mała. Przeciwdziedzina f_T to zbiór $n+1$ wartości dla n testów. Dla tak ograniczonego zakresu możliwych wartości zwracanych przez f_T jest wysoce prawdopodobne, że ewolucyjne operatory selekcji nie będą zdolne do odkrycia najbardziej obiecujących kandydatów. W konsekwencji proces wyszukiwania staje się niedoinformowany o cechach analizowanych kandydatów, płacąc za to niezadowalającą wydajnością, ograniczoną skalowalnością, a w skrajnych przypadkach czysto losowym przeszukiwaniem.

Tradycyjne funkcje oceny pokroju f_T stosowane w obliczeniach ewolucyjnych cierpią także na niską korelację z odległością do optymalnego rozwiązania (ang. fitness-distance correlation [33]), rozumianą jako liczba kroków algorytmu (zastosowań operatorów przeszukiwania) niezbędną do otrzymania optymalnego rozwiązania. Zilustrujmy to na konkretnym przykładzie problemu opartego na testach, tj. problemie syntezy programu sortującego liczby naturalne w kierunku rosnącym. Załóżmy że algorytm syntezy (np. PG omawiane wcześniej) na skutek swojego dotychczasowego działania zdołał skonstruować rozwiązanie, które sortuje listy wejściowe w porządku odwrotnym (malejąco), np. dla listy wejściowej (5,8,7,3,2) zwraca on listę (8,7,5,3,2), zamiast pożądanej (2,3,5,7,8). Typowa skalarna funkcja oceny (np. odległość Hamminga pomiędzy listami czy współczynnik korelacji Kendalla) nisko oceni tak zachowujący się program, ponieważ jego wyjście znacznie różni się od pożądanego. Jest jednak wysoce prawdopodobne że drobna modyfikacja kodu programu (przypuszczalnie zamiana operatora arytmetycznego $>$ na $<$ lub odwrotnie) da w wyniku program optymalny, zwracający poprawnie uporządkowane listy. Innymi słowy, dwa programy znajdujące się w bardzo małej odległości (dzieli je jedna mutacja w PG) diametralnie różnią się zachowaniem. Konwencjonalne funkcje oceny nie są w stanie dostrzec tak subtelnych różnic w zachowaniu rozwiązań kandydackich.

Powyższy przykład dobitnie ukazuje że funkcja oceny, choć zdolna do odzwierciedlenia obiektywnej jakości kandydata i określa cechy rozwiązania optymalnego, nie koniecznie sprawdza się w roli sygnału sterującego procesem przeszukiwania. Ta obserwacje jest w istocie potwierdzona w praktyce przez tzw. *zwodnicze* problemy (ang. deceptive problems). Pożądane jest zatem poszukiwanie alternatywnych sygnałów sterujących, czy też *funkcji oceny*, które dawałyby większe szanse na skierowanie procesu przeszukiwania we właściwe obszary przestrzeni rozwiązań. Dla powyższego problemu sortowania taki kryterium oceny mogłoby np. nagradzać programy realizujące *jakiekolwiek* sortowanie danych wejściowych.

Rozdział 4

Automatyczne odkrywanie funkcji oceny

Jak pokazano w poprzednim rozdziale, informacja utracona na skutek agregacji wyników interakcji może być kluczowa dla efektywnej oceny i selekcji najbardziej obiecujących kandydatów z S. Co więcej, stosowanie agregującej funkcji celu jest głównym źródłem zjawiska wąskiego gardła oceny i wynikających z niego negatywnych konsekwencji dla procesu przeszukiwania. W tym rozdziale proponujemy metodykę zorientowaną na przeciwdziałanie tym problemom poprzez inteligentną wieloobiektywizację (ang. multiobjectivization) funkcji celu i automatyczne odkrywanie wieloaspektowej charakterystyki kandydatów mającej na celu skierowanie procesu przeszukiwania w najbardziej obiecujące obszary przestrzeni rozwiązań.

4.1 Macierz interakcji

Jak pokazano w sekcji 2.1, konwencjonalna funkcja oceny w problemach opartych na testach polega na przeprowadzeniu licznych interakcji pomiędzy kandydatami i testami, a następnie na ich agregacji. Przebieg interakcji oraz jej wynik zależy w dużej mierze od funkcji interakcji $g : \mathcal{S} \times \mathcal{T} \to \mathcal{O}$, która określa podzbiór testów rozwiązanych przez $s \in S$. Dla przykładu, mając dany zbiór testów T, gdzie każdy test $(x_i, y_i) \in T$ jest parą składającą się z danych wejściowych x oraz oczekiwana wyjścia y, funkcja interakcji g może być zdefiniowana jako:

$$g(s,t) = g(s,(x,y)) = [s(x) = y],$$

gdzie $s(x)$ oznacza efekt działania s na danych x, a $[\cdot]$ jest nawiasem Iversona:

$$[x] = \begin{cases} 1 & \text{if x is true} \\ 0 & \text{otherwise.} \end{cases}$$

W najprostszym przypadku, jeżeli kandydat s *rozwiązuje* test t wedle funkcji interakcji g, wówczas powiemy, że wynikiem tej interakcji jest $g(s,t) = 1$. W przeciwnym razie, powiemy, że s nie przechodzi testu t i $g(s,t) = 0$. Ogólnie rzecz biorąc, wynik interakcji wcale nie musi być binarną wartością, jak powyżej. Popularnym podejściem, które sprawdza się w praktyce jest zastosowanie miary odległości między $s(x)$ a y, np. $g(s,t) = |s(x) - y)|$.

Kluczową obserwacją dla dalszych rozważań jest możliwość zebrania wszystkich wyników interakcji kandydatów w zbiorze S ze wszystkimi testami w T w postaci tzw. *macierzy interakcji*

$$G = [g_{ij} = g(s_i, t_j) : s_i \in P, t_j \in T]. \tag{4.1}$$

Dla zbioru m kandydatów i zbioru n testów, G jest macierzą wymiarach $m \times n$, gdzie g_{ij} oznacza wyniki interakcji między i-tym kandydatem s_i a j-tym testem t_j. Jeśli wyniki interakcji są binarne, to $g_{ij} = [s_i(x_j) = y_j]$. W przeciwnym razie $g_{ij} = |s_i(x_j) - y_j|$.

4.2 Heurystyczne funkcje oceny

W tej sekcji wprowadzimy pojęcie heurystycznej funkcji oceny zaprojektowanej aby pełnić rolę alternatywnego sygnału sterującego procesem przeszukiwania. Proponowana metodyka ma na celu odzwierciedlenie różnych aspektów jakościowych ocenianych kandydatów oraz zapewnienie użytecznego gradientu w stronę bardziej obiecujących obszarów w przestrzeni rozwiązań.

Formalnie możemy zdefiniować heurystyczną funkcję oceny w następujący sposób:

Definicja 4.1. Heurystyczna funkcja oceny dla problemu opartego na testach (S, T, g) to funkcja

$$d : \mathcal{S}^m \to \mathbb{R}^m, \tag{4.2}$$

która odzwierciedla m-elementowy zbiór kandydatów z S na m-elementowy zbiór ich ocen, gdzie m jest rozmiarem populacji S. Przekształcenie implementowane przez d bazuje na przeprowadzanych z wykorzystaniem g interakcjach pomiędzy elementami S a podzbiorem testów $T' \subseteq T$.

Z powyższej definicji wynika bezpośrednio, że w odróżnieniu od funkcji oceny, która odzwierciedla wynik interakcji kandydata z wszystkimi testami, d rozumiana jako sygnał sterujący z założenia niesie ze sobą jedynie *część informacji* charakteryzującej jakość badanych rozwiązań. Naturalnym przykładem takiego sygnału sterującego jest błąd popełniany przez kandydata na wybranym *podzbiorze testów*. W ogólności nie zakładamy też ścisłej monotoniczności d względem liczby testów rozwiązywanych przez kandydata. Poprzez pojęcie heurystycznej funkcji oceny będziemy zatem rozumieć niedoskonały substytut obiektywnej funkcji celu.

Drugim istotnym aspektem wyróżniającym heurystyczne funkcje oceny jest ich *kontekstowość*. Podczas gdy konwencjonalna ocena kandydata s jest niezależna od innych elementów S, sygnatura dana przez (4.2) umożliwia jednoczesną ocenę całej populacji kandydatów. Z tego powodu wygodnie jest myśleć o d jako o funkcji wektorowej

$$d = (h_1, h_2, \ldots, h_m), \tag{4.3}$$

gdzie $h_i : S^m \to \mathbb{R}$ jest funkcją składową zdolną do oceny kandydata s_i nie tylko w oparciu o jego zachowanie na zbiorze testów T' ale także biorąc pod uwagę zachowanie kandydatów $s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_m$. W związku z powyższym, s_i może zostać oceniony różnie w zależności od kontekstu oceny, tj. pozostałych elementów S biorących udział w procesie oceny. Dla wygody będziemy czasem nadużywać notacji pisząc po prostu $h_i^d(s_i)$ aby oznaczyć ocenę przyznaną i-temu kandydatowi s_i przez funkcję oceny d w kontekście członków S.

Pomimo iż d jest zorientowana na kontekstową ocenę kandydatów, warto zauważyć, że powyższa definicja jest wystarczająco ogólna aby objąć również konwencjonalną funkcję oceny $f : \mathcal{S} \to \mathbb{R}$. Takie funkcje są powszechnie stosowane w optymalizacji i mogą zostać łatwo wyrażone jako

$$d(s_1, \ldots, s_m) = (f(s_1), \ldots, f(s_m)), \tag{4.4}$$

gdzie oceny przyznane poszczególnym kandydatom są całkowicie niezależne. Z tego powodu należy postrzegać heurystyczne funkcje oceny jako nadzbiór konwencjonalnych funkcji oceny.

Pojedyncza funkcja oceny d ma za zadanie odzwierciedlić jedynie wybrane cechy kandydatów $s \in S$. Uzasadnione zatem jest stosowanie *wielu* takich funkcji celem uzyskania wieloaspektowej oceny kandydatów. Warto również spojrzeć na tę własność poprzez analogię z klasyfikatorami złożonymi w uczeniu maszynowym, których skuteczność wynika z wykorzystania (czasami bardzo wielu) *prostych* klasyfikatorów składowych (ang. weak classifiers). Innymi słowy, przeszukiwanie przestrzeni rozwiązań przy pomocy wielu takich sygnałów sterujących może dać lepsze efekty niż wykorzystanie oryginalnej funkcji celu. Co najważniejsze, wielokryterialny opis zachowania kandydatów pozwoli również uniknąć niekorzystnych implikacji wąskiego gardła oceny.

4.3 Odkrywanie heurystycznych funkcji oceny

Proponowane heurystyczne funkcje oceny stanowią alternatywne kryteria oceny jakości kandydatów, a kilka takich funkcji stosowanych jednocześnie składa się na wielokryterialny sygnał sterujący procesem przeszukiwania. Nadal jednak nie odpowiedzieliśmy na bardzo ważne pytanie: w jaki sposób możemy skonstruować takie funkcje oceny? Z założenia funkcje te nie powinny być przygotowywane przez człowieka, ponieważ wymagałoby to znacznej wiedzy na temat rozwiązywanego problemu, która najczęściej nie jest dostępna. Kluczowym elementem proponowanej metodyki są więc algorytmy do w pełni *automatycznego* i *nienadzorowanego* odkrywania takich funkcji poprzez analizę macierzy interakcji G.

Przypomnijmy pokrótce, że dla populacji m rozwiązań i zbioru n testów, G jest macierzą o wymiarach $m \times n$, której elementy odpowiadają wynikom interakcji pomiędzy elementami S i T. Formanie rzecz ujmując, procedura odkrywania heurystycznych funkcji oceny polega na transformacji macierzy G na macierzy G' o wymiarach $m \times k$, czyli:

$$u(G) = G', \tag{4.5}$$

gdzie u jest algorytmem odpowiedzialnym za rzeczoną transformację, a k określa liczbę nowych funkcji oceny. Zauważmy także, że istotnym jest aby k było znacznie mniejsze niż n. W ten sposób możliwe będzie uzyskanie kompromisu pomiędzy skalarną funkcją oceny a relacją dominacji opartą na testach (zob. sekcja 3.2).

Interpretacja wierszy macierzy G' pozostaje niezmieniona, tj. każdy wiersz odpowiada jednemu elementowi w S. Z kolei kolumny G' określają teraz k odkrytych funkcji oceny będących źródłem alternatywnego opisu zdolności kandydatów w S w kontekście ich zachowanie na zbiorze testów T. Wartość j-tej funkcji oceny dla i-tego kandydata s_i to

$$h_i^j(s_i) = g'_{i,j}, \tag{4.6}$$

gdzie $g'_{ij} \in G'$. Od funkcji u wymagać będziemy aby elementy g'_{ij} były nieujemnymi wartościami ciągłymi. Zauważmy też, że w przypadku binarnej funkcji interakcji, G' z definicji dostarcza bardziej szczegółowej (ciągłej) informacji na temat kandydatów.

Aby wyznaczyć funkcje oceny, u wykorzystuje techniki odkrywania wiedzy do analizy macierzy interakcji. Innymi słowy, proces transformacji G w G' polega w znacznej mierze na zastosowaniu mechanizmu *uczenia się z danych* aby znaleźć powiązania pomiędzy kandydatami a testami w oparciu o ich cechy behawioralne. Ze względu na brak nadzorowanego sygnału, który poprowadziłby ten proces, zbiór danych uczących doświadczanych przez u składa się jedynie ze zbioru obserwacji (wyników interakcji) opisujących kandydatów i testy (wiersze i kolumny G). Proces odkrywania funkcji oceny jest zatem zadaniem *nienadzorowanego uczenia* (ang. unsupervised learning), gdzie nadrzędnym celem jest wyjaśnienie kluczowych cech (ang. features) macierzy G i odpowiednie zamodelowanie jej struktury. Z nieco innej perspektywy, proces uczenia się przez u może być widziany jako poszukiwanie nisko wymiarowej *reprezentacji* dla wyników interakcji, która umożliwi wielokryterialną selekcję kandydatów, a jednocześnie zachowa jak najwięcej informacji o G.

W dalszej części rozdziału omawiamy m.in. pożądane własności u oraz demonstrujemy w jaki sposób odkryte funkcje oceny mogą zostać osadzone w metaheurystykach rozwiązujących problemy oparte na testach. W szczególności skupiamy się na możliwych schematach wielokryterialnej selekcji z wykorzystaniem G'. Z kolei dalsze rozdziały skupiają się na konkretnych algorytmach implementujących operację u.

Rozdział 5

Algorytm odkrywania funkcji oceny metodą analizy skupień

W tym rozdziale proponujemy metodę odkrywania alternatywnych funkcji oceny (nazywanych także *kryteriami przeszukiwania*) przestrzeni rozwiązań, nazywaną dalej DOC (ang. Discovery of Search Objectives by Clustering) wykorzystując w tym celu techniki analizy skupień. Idea algorytmu polega na zidentyfikowaniu podobnych testów, a następnie pogrupowaniu kolumn macierzy interakcji G w taki sposób, aby uwydatnić pewne interesujące wzorce zachowania kandydatów na odkrytych grupach testów. W efekcie każda grupa może być utożsamiona z nowym kryterium przeszukiwania. Kluczową własnością algorytmu jest gwarancja, że liczba odkrytych kryteriów będzie znacznie mniejsza niż liczba testów dzięki czemu możliwe staję się wykorzystanie wielokryterialnych technik selekcji kandydatów.

Dla populacji S składającej się z m kandydatów oraz zbioru n testów T algorytm DOC działa w następujący sposób:

1. Oblicz macierz interakcji G o wymiarach $m \times n$ pomiędzy kandydatami a testami używając w tym celu funkcji interakcji g.

2. Pogrupuj testy. Każda kolumna w G jest traktowana jako wektor w m-wymiarowej przestrzeni wyników interakcji wszystkich kandydatów z jednym testem t odpowiadającym tej kolumnie. Zastosuj wybrany algorytm analizy skupień (np. k-means) do n punktów uzyskanych w powyższy sposób. Rezultatem tego kroku jest podział G na partycje $\{T_1, \ldots, T_k\}$ n testów w T na k podzbiorów $T_j \subset T$, $j \in [1, k]$, gdzie $1 \leq k \leq n$ i $\forall_j : T_j \neq \emptyset$.

3. Zdefiniuj nowe kryteria przeszukiwania. Dla każdego podzbioru T_j, uśredniamy po wierszach odpowiednie kolumny w G. W efekcie otrzymujemy macierz $m \times k$ odkrytych kryteriów G', z elementami zdefiniowanymi w następujący sposób:

$$g'_{ij} = \frac{1}{|T_j|} \sum_{t \in T_j} g(s_i, t), \tag{5.1}$$

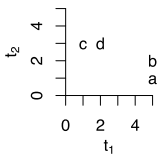
   gdzie s_i jest kandydatem odpowiadającym i-temu wierszowi w G.

Kolumny G' definiują k nowych kryteriów charakteryzujących kandydatów w S. Kryteria te są unikalne dla każdej konkretnej instancji S i T i są niezdefiniowane poza tymi konkretnymi wartościami S i T. Wartość j-tego kryterium dla kandydata s_i to g'_{ij}.

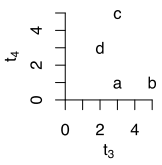
Kryteria przeszukiwania indukowane przez macierz G' mogą być postrzegane jako wielokryterialna charakterystyka kandydatów w kontekście zbioru testów T. Ponieważ $k << n$, to mogą one zostać wykorzystane przez dowolny wielokryterialny algorytm selekcji jak np. NSGA-II [6].

G	t_1	t_2	t_3	t_4
a	5	1	3	1
b	5	2	5	1
c	1	3	3	5
d	2	3	2	3

(a)



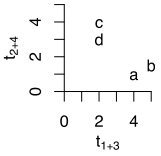
(b)



(c)

G'	t_{1+3}	t_{2+4}
a	4	1
b	5	1.5
c	2	4
d	2	3

(d)



(e)

Rysunek 5.1: Przykład kompresji macierzy interakcji przez algorytm DOC.

Przykład 5.1. Rozważymy macierz interakcji pomiędzy kandydatami $S = \{a, b, c, d\}$, a zbiorem testów $T = \{t_1, t_2, t_3, t_4\}$, jak pokazano na Rys. 5.1a. Czterowymiarowa przestrzeń wyników interakcji została zilustrowana na dwóch dwuwymiarowych wykresach punktowych (Rys. 5.1b oraz 5.1c) rozpiętych na $t_1 \times t_2$ i $t_3 \times t_4$. Jak łatwo zauważyć, zachowanie kandydatów na testach t_1 and t_3 jest skorelowane, podobnie jak w przypadku testów t_2 and t_4. Załóżymy, że algorytm analizy skupień (krok 2 algorytmu DOC) zauważa tę zależność i produkuje $k = 2$ grupy testów, które dzielą T na $\{T_1, T_2\}$ w taki sposób, że $T_1 = \{t_1, t_3\}$ i $T_2 = \{t_2, t_4\}$. Uśredniając wyniki interakcji wewnątrz poszczególnych grup T_j (krok 3 algorytmu DOC) otrzymujemy macierz G' pokazaną na Rys. 5.1d. Rys. 5.1e ilustruje lokalizację kandydatów w przestrzeni nowych kryteriów.

Jeżeli wyniki interakcji są maksymalizowane, to jedyna dominacja zachodząca w oryginalnej przestrzeni testów to $b \succ a$. W przestrzeni odkrytych kryteriów (Rys. 5.1e) b nadal dominuje nad a. Jednakże teraz również c dominuje nad d, choć pierwotnie te dwa rozwiązania były wzajemnie niezdominowane. W efekcie kompresji przeprowadzonej przez DOC część informacji o relacji dominacji została utracona.

W tym konkretnym przypadku, wprowadzenie dominacji na rzecz c może być pożądane, jako że c jest lepszy niż d na dwóch testach (t_3, t_4), podczas gdy tylko jeden test (t_1) podpiera przeciwną relację (t_2 jest neutralny w tym aspekcie). Algorytm DOC wprowadza więc kompromis pomiędzy liczbą kryteriów przeszukiwania względem liczby testów, a niespójnością z pierwotnymi wynikami interakcji.

Algorytm DOC poszerza wąskie gardło funkcji oceny charakteryzując kandydatów w oparciu o k odkrytych kryteriów. Dostatecznie małe k gwarantuje, że relacja dominacji w przestrzeni kryteriów będzie wystarczająco gęsta aby zapewnić skuteczny gradient przeszukiwania (w przeciwieństwie do relacji dominacji rozpiętej na wszystkich testach, jak w 3.2). Co więcej, kandydaci o różnych umiejętnościach (wyrażonych przez podzbiory testów) mogą współistnieć w populacji nawet jeżeli część z nich jest gorsza w sensie f_T. Dla przykładu, kandydat c w powyższym przykładzie nie jest zdominowany w G' przez b, choć $f_T(c) < f_T(b)$.

W dalszej części rozdziału przedstawiono pozostałe własności algorytmu DOC, m.in. formalnie udowodniono, że DOC nie może odwrócić relacji dominacji w G'. Przeprowadzano także liczne eksperymenty porównawcze algorytmów koewolucyjnych i programowania genetycznego dla trzech zróżnicowanych problemów opartych na testach i wykazano statystycznie istotną wyższość DOC nad innymi metodami. W rozdziale przedstawiono także interesującą analizę odkrywanych przez DOC kryteriów, m.in. ich wizualizację w przestrzeni dwuwymiarowej.

Metoda DOC została opublikowana w [21], a następnie rozwinięta w [16, 24].

Rozdział 6

Algorytm odkrywania kryteriów przeszukiwania metodą faktoryzacji macierzy

Rozdział ten przedstawia drugi z zaprojektowanych algorytmów dedykowanych zagadnieniu odkrywania heurystycznych funkcji oceny (kryteriów przeszukiwania przestrzeni rozwiązań) dla problemów opartych na testach. Proponowana metoda, nazywana dalej DOF (ang. Discovery of Search Objectives by Factorization), wykorzystuje popularną w uczeniu maszynowym technikę nieujemnej faktoryzacji macierzy celem odnalezienia powiązań między kandydatami oraz wzorcy w ich zachowaniu na zbiorze testów T. W szczególności metoda ta pozwala na modelowania każdego elementu macierzy G (wyników interakcji g_{ij}) w postaci liniowej kombinacji kilkunastu faktorów. Jak pokażemy w następnym paragrafie, faktory te stanowią główny budulec heurystycznych funkcji oceny odkrywanych przez algorytm DOF.

Z perspektywy metaheurystyk ewolucyjnych, DOF pełni rolę metody oceny oraz selekcji osobników. Metoda DOF zastosowana do populacji S składającej się z m kandydatów oraz zbioru n testów w T oraz dla zadanego współczynnika faktoryzacji r działa w następujący sposób:

1. Oblicz macierz interakcji G o wymiarach $m \times n$ pomiędzy kandydatami a testami używając w tym celu funkcji interakcji g.

2. Przeprowadź nieujemną faktoryzację macierzy G. Efektem tego kroku jest dekompozycja G na iloczyn macierzy W o wymiarach $m \times r$ oraz H o wymiarach $r \times n$.

3. Wyznacz r nowych kryteriów przeszukiwania d_j, $j = 1, \ldots, r$ jako

$$d_j(s_i) = w_{ij}. \tag{6.1}$$

   Innymi słowy, macierz W jest interpretowana jako macierz odkrytych kryteriów G' o wymiarach $m \times r$, której elementy są zdefiniowane jako $g'_{ij} = d_j(s_i)$.

4. Zastosuj wielokryterialną metodę selekcji do kryteriów d_j celem selekcji rodziców dla kolejnej iteracji algorytmu ewolucyjnego.

W wariancie metody nazywanym dalej DOF-W, kolumny macierzy W są natychmiastowo traktowane jako nowe kryteria przeszukiwania, a ich wartości dla kandydatów $s \in S$ są pobrane bezpośrednio z odpowiednich wierszy w W (6.1). Kryteria przeszukiwania dla kandydata s mogą być interpretowane jako koordynaty wektora wyników interakcji w r-wymiarowej przestrzeni indukowanej przez faktoryzację macierzy. Ponieważ wszystkie elementy G, W i H są nieujemne, oraz iloczyn

Algorithm 2 Algorytm odkrywania kryteriów metodą faktoryzacji macierzy.

Require: factorization rank r.

1: **function** DOF(S, T)
2: **for** $s \in S$ **do**
3: **for** $t \in T$ **do**
4: $g(s,t) \leftarrow$ INTERACT(s,t) ▷ wyznacza macierz interakcji G
5: $W, H \leftarrow \mathrm{NMF}(G, r)$
6: **for** $j \in 1, \ldots, r$ **do**
7: **for** $s_i \in S$ **do**
8: **if** DOF-W **then** ▷ określa wariant algorytmu
9: $g'_j(s_i) \leftarrow w_{ij}$
10: **else**
11: $g'_j(s_i) \leftarrow w_{ij}\mathbf{g}_j^T \cdot \mathbf{h}_k$
12: **return** G'

W i H skutkuje addytywną kombinacją wag i odpowiadających im faktorów, to mamy gwarancję, że otrzymane kryteria stanowią rzadką reprezentację (ang. sparse representation) wyników interakcji. To z kolei pozwala nam twierdzić, że są one pozytywnie skorelowane z elementami G, dzięki czemu można je traktować jako kryteria podlegające maksymalizacji w trakcie wielokryterialnej selekcji w kroku 4 algorytmu DOF.

Kluczową obserwacją dla metody DOF jest fakt, iż jedynie iloczyn skalarny odpowiednich wierszy i kolumn macierzy W oraz H w pełni modeluje poszczególne wyniki interakcji. W praktyce oznacza to, że nieujemna faktoryzacja macierzy rzutuje elementy G na łączną przestrzeń indukowaną przez elementy (wagi i faktory) w W i H. Innymi słowy, celowe odrzucenie zawartości H przez metodę DOF-W może wydawać się nieuzasadnione. Aby rozwiązać ten problem, proponujemy drugi wariant metody o nazwie DOF-WH, w którym krok 3 powyższego algorytmu DOF zdefiniowany jest następująco

$$d_j(s_i) = w_{ij}\mathbf{g}_i^T \cdot \mathbf{h}_j = w_{ij} \sum_{k \in I(s)} g_{ik} h_{jk}, \tag{6.2}$$

gdzie $I(s) = \{k : t_k \in T, g(s,t_k) = 1\}$. W tym wariancie metody (zobacz również schemat Algorytmu 2), faktory opisujące kandydatów w W są dodatkowo ważone przez elementy $t_k \in T$, $k \in I(s)$ macierzy H opisującej testy. W rezultacie, j-te kryterium charakteryzujące kandydata s agreguje j-ty faktor opisujący go w W z j-tym faktorem dla wszystkich testów w H, które zostały rozwiązane przez s. Nieujemna faktoryzacja macierzy dekomponuje poszczególne wyniki interakcji w G na sumę r komponentów, z których każdy jest iloczynem wagi w W oraz faktora w H:

$$g_{st} \approx w_{s1}h_{1t} + w_{s2}h_{2t} + \cdots + \underbrace{w_{sj}h_{jt}}_{j\text{-ty komponent}} + \cdots + w_{sr}h_{rt}. \tag{6.3}$$

A zatem j-te kryterium odkryte przez algorytm DOF-WH może być interpretowane jako suma j-tych komponentów modelujących wyniki interakcji s z testami $t_k \in T$, $k \in I(s)$.

W dalszej części rozdziału przedstawiono m.in. dowody interesujących własności algorytmu, oraz przeprowadzono wieloetapowy eksperyment obliczeniowy mający na celu zweryfikowanie algorytmów DOF-W i DOF-WH w kontekście praktycznych problemów opartych na testach. Wczesna wersja opisywanych tu metod została opublikowana w [23].

Rozdział 7

Algorytm SFIMX

Ocena jakości rozwiązań kandydackich w obliczeniach ewolucyjnych jest zwykle najdroższym elementem procesu wyszukiwania pod względem wymaganego czasu obliczeń. Jest to szczególnie widoczne w zastosowaniach, w których funkcja oceny (interakcji) obejmuje pewną formę symulacji, np. robota w środowisku wirtualnym. Proces ewaluacji może stać się jeszcze droższy w problemach opartych na testach, gdzie ocena rozwiązania kandydackiego wymaga symulacji jego zachowania w wielu środowiskach. Zarówno gry dwu- i wieloosobowe stanowią dobry przykład problemu z kosztowną funkcją interakcji. Analogicznie, proces testowania programu komputerowego na zadanym zestawie testów w programowaniu genetycznym może być również postrzegany jako kosztowna symulacja, szczególnie gdy programy stają się duże.

Obniżenie kosztów obliczeniowych związanych z funkcją oceny poprzez redukcję liczby testów najczęściej nie jest możliwe. Niewielka liczba testów implikuje niedokładną estymację jakości kandydatów, a co za tym idzie słabo poinformowany proces wyszukiwania. Co więcej, umyślne odrzucanie testów może spowodować, że zmieni się sama definicja problemu, a tym samym charakterystyka optymalnego rozwiązania. Ponadto, gdy funkcja interakcji jest binarna (test jest rozwiązany lub nie), niewielka liczba testów utrudnia rozróżnianie kandydatów i przyczynia się do problemu wąskiego gardła (zob. sekcja 3.1).

Praktyczną odpowiedzią na te wyzwania są tzw. *modele zastępcze* (ang. surrogate models), w których funkcja celu f jest zastępowana tańszym obliczeniowo, aczkolwiek przybliżonym, *surogatem* $\hat{f}$. W niektórych sytuacjach, $\hat{f}$ może być zaprojektowana ręcznie przez eksperta w danej dziedzinie. Jednakże, od niedawna uczenie się takich modeli z przykładów (danych) stało się popularną i interesującą alternatywą [4].

W tym rozdziale pokażemy, że formalizm faktoryzacji macierzy, który jest kluczowym elementem algorytmu DOF może być wykorzystany także do *predykcji* wyników interakcji pomiędzy kandydatami S a testami T. Poniżej proponujemy metodę inspirowaną tą obserwacją, która heurystycznie estymuje wartość funkcji oceny kandydatów z G wykorzystując do tego celu nieujemną faktoryzację macierzy. Co najważniejsze, proponowany algorytm wymaga poznania jedynie niewielkiej części elementów G, co oznacza, że w praktyce muszą zostać przeprowadzone tylko nieliczne interakcje między kandydatami i testami. Jak wykażemy poniżej, pozwala to na znaczną redukcję wymaganego nakładu obliczeniowego w stosunku do konwencjonalnego podejścia.

Zaznaczmy też pokrótce, że w przeciwieństwie do metod DOC i DOF, proponowana w tym rozdziale metoda nie jest zorientowana na zwalczanie negatywnych konsekwencji wąskiego gardła funkcji oceny. Celem algorytmu jest zaś zademonstrowanie, że koncepcja alternatywnych funkcji oceny odkrywanych dynamicznie w trakcie rozwiązywania problemu może być przydatna także w zgoła innych sytuacjach.

Algorithm 3 Algorytm SFIMX dedykowany problemowi predykcji (estymacji) brakujących elementów macierzy interakcji.

Require: ranga faktoryzacji r.

function SFIMX(S, T, α)
  for $s \in S$ **do**
    $T' \leftarrow \textsc{Sample}(T, \alpha)$
    for $t \in T'$ **do**
      $g(s,t) \leftarrow \textsc{Interact}(s,t)$
  $W, H \leftarrow \mathrm{NMF}(G, r)$
  $\hat{G} \leftarrow WH$
  $\hat{G} \leftarrow \textsc{ReplaceKnown}(G, \hat{G})$
  for $s_i \in S$ **do**
    $f(s_i) \leftarrow \sum_{j=1}^{n} \hat{g}_{ij}$
  return F

7.1 SFIMX

Jak zademonstrowano we wstępie do tego rozdziału, obliczenie pełnej macierzy G może być obliczeniowo bardzo wymagające, szczególnie gdy przeprowadzenie pojedynczych interakcji jest kosztowne, gdy liczba testów w T jest duża lub gdy zachodzą oba warunki jednocześnie. Aby rozwiązać ten problem, proponujemy algorytm SFIMX (ang. Surrogate Fitness via Factorization of Interaction Matrix), który oblicza jedynie ułamek wszystkich interakcji (a więc G jest macierzą rzadką), a następnie wykorzystuje faktoryzację macierzy w procesie predykcji brakujących wartości na podstawie znanych elementów w G. Algorytm SFIMX oczekuje na wejściu dwóch parametrów: rangi faktoryzacji r oraz pożądanej gęstości $\alpha \in (0, 1]$ częściowej macierzy interakcji. Algorytm SFIMX modyfikuje konwencjonalną ocenę jakości rozwiązań kandydackich w następujący sposób:

1. Wyznacz rzadką macierz interakcji G pomiędzy kandydatami z bieżącej populacji a testami ze zbioru T w następujący sposób:

   a) Dla każdego kandydata s, wylosuj niepusty podzbiór testów $T' \subset T$ zawierający αn elementów, gdzie $\alpha \in (0, 1]$ jest parametrem kontrolującym ułamek interakcji, które będą obliczone.

   b) Przeprowadź interakcje między kandydatami $s \in S$ a testami w T', umieszczając wyniki tych interakcji w odpowiednich komórkach macierzy G.

2. Przeprowadź faktoryzację G na nieujemny macierze W i H, używając jedynie znanych elementów G oraz ignorując wszystkie nieznane (brakujące) wyniki interakcji.

3. Użyj otrzymane macierze do predykcji wyników interakcji poprzez obliczenie $\hat{G} = WH$.

4. Dla wszystkich znanych elementów G zastąp ich predykcje wartościami z G.

5. Wyznacz na podstawie $\hat{G}$ estymatę wartości funkcji oceny kandydata $s \in S$ jako:

$$f_{SFIMX}(s) = \sum_{j=1}^{n} \hat{g}_{sj}, \tag{7.1}$$

   tj. w ten sam sposób jak w przypadku konwencjonalnego podejścia, z tą różnicą, że podstawą do obliczeń jest macierz $\hat{G}$.

Algorytm 3 podsumowuje powyższe kroki w postaci pseudokodu.

Przykład 7.1. Rozważmy populację kandydatów $S = \{s_1, s_2, s_3, s_4\}$ oraz zbiór testów $T = \{t_1, t_2, t_3, t_4, t_5\}$. Przyjmijmy, że algorytm SFIMX został uruchomiony z $\alpha = \frac{3}{5}$ oraz, że pierwszy krok algorytmu zwrócił następującą rzadką macierz interakcji G:

$$G = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} 2 & & 1 & 2 & \\ & 2 & 1 & & 1 \\ 1 & & & 2 & 2 \\ 2 & 1 & & & 1 \end{pmatrix} \end{array}$$

Niech $r = 3$. Przyjmijmy, że faktoryzacja macierzy G (linia 6 Algorytmu 3) wyznaczyła następującą dekompozycję na macierze W i H:

$$W = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{ccc} f_1 & f_2 & f_3 \end{array} \\ \begin{pmatrix} 0.46 & 1.96 & 0.6 \\ 1.27 & 0.1 & 0.95 \\ 1.37 & 0.02 & 2.83 \\ 0.4 & 1.86 & 1.60 \end{pmatrix} \end{array}, \quad H = \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} 0.48 & 1.50 & 0.01 & 0.41 & 0.41 \\ 0.87 & 0.14 & 0.19 & 0.77 & 0.01 \\ 0.11 & 0.09 & 1.02 & 0.50 & 0.51 \end{pmatrix} \end{array}.$$

Wymnożenie macierzy W i H (linia 7 Algorytmu 3) zaowocowało następującą rekonstrukcją G:

$$\hat{G} = WH = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} 2 & 1.02 & 1 & 2 & 0.52 \\ 0.8 & 2 & 1 & 1.07 & 1 \\ 1 & 2.31 & 2.1 & 2 & 2 \\ 2 & 1 & 2.01 & 2.4 & 1 \end{pmatrix} \end{array}$$

W kolejnym kroku, procedura `ReplaceKnown` kopiuje znane wyniki interakcji z G do $\hat{G}$. Ostatecznie, w linii 10, obliczamy wartość funkcji oceny poszczególnych kandydatów poprzez sumowanie odpowiednich wierszy zrekonstruowanej macierzy interakcji, co skutkuje następującymi wartościami funkcji f_{SFIMX}: $f_{SFIMX}(s_1) = 6.54$, $f_{SFIMX}(s_2) = 5.87$, $f_{SFIMX}(s_3) = 9.41$, i $f_{SFIMX}(s_4) = 8.41$. Podsumowując, algorytm SFIMX umożliwił wyznaczenie tych wartości używając jedynie $\alpha nm = 12$ wyników interakcji w porównaniu do $nm = 20$ interakcji wymaganych przez konwencjonalne podejście wykorzystujące f_T.

W powyższym przykładzie zrekonstruowana macierz $\hat{G}$ doskonale odtwarza znane wyniki interakcji, tak więc błąd aproksymacji minimalizowany przez faktoryzację macierzy osiąga zero. Ogólnie rzecz biorąc, błąd aproksymacji jest zwykle większy dla mniejszych r i większych α. Należy jednak zauważyć, że niezależnie od skuteczności procesu faktoryzacji, nieznane wyniki interakcji są tylko ekstrapolowane, a więc f_{SFIMX} będzie w ogólności odbiegać od f_T.

W pozostałej części rozdziału przeprowadzono szczegółową analizę eksperymentalną algorytmu SFIMX dla wybranych problemów opartych na testach w kontekście algorytmów koewolucyjnych oraz programowania genetycznego. Zaproponowano także dalsze rozszerzenia metody, w tym sposób na automatyczny dobór parametru α, oraz optymalizację procesu wyboru testów do przeprowadzenia interakcji stanowiących podstawę do predykcji brakujących elementów G.

Algorytmy opisane w tym rozdziale zostały pierwotnie zaproponowane w [22], a później rozwinięte i dopracowane w [26, 20].

Rozdział 8

Podsumowanie

8.1 Konkluzje

Wyniki badań przedstawione w niniejszej rozprawie wyraźnie sugerują, że macierze interakcji będące naturalnym elementem każdego problemu opartego na testach są niezwykle bogatym źródłem informacji opisującym zachowanie rozwiązań kandydackich w kontekście zbioru testów. Zaprojektowane algorytmy automatycznej analizy tych danych okazały się skutecznym sposobem na pozyskanie wieloaspektowej charakterystyki mogącej posłużyć m.in. do ograniczenia zjawiska wąskiego gardła funkcji oceny.

Poniżej podsumowujemy najważniejsze cechy zaproponowanej metodyki odkrywania heurystycznej funkcji oceny:

1. Pojęcie heurystycznej funkcji oceny odzwierciedla tylko wybrane cechy rozwiązań kandydackich. Ich główną rolą jest sterowanie procesem wyszukiwaniem w taki sposób, aby stworzyć użyteczny gradient w stronę lepszych rozwiązań.

2. Heurystyczne funkcje oceny, rozumiane jako kryteria wyszukiwania, zapobiegają problemowi nadmiernej eksploatacji pojedynczych testów oraz dywersyfikują populację poprzez ciągłą zmianę presji selekcyjnej pomiędzy kryteriami. Nacisk kładziony jest więc na różne aspekty jakości rozwiązania, promując w ten sposób behawioralną różnorodność wśród kandydatów, minimalizując jednocześnie ryzyko zjawiska przedwczesnej zbieżności (ang. premature convergence).

3. Nowe funkcje oceny są odkrywane w każdym pokoleniu procesu ewolucyjnego zupełnie niezależnie. W ten sposób zmieniają się dynamicznie, aby zauważyć nowe i interesujące zachowania pojawiające się w macierzach interakcji podczas ewolucji. Proponowana metodyka może więc być postrzegana jako adaptatywna metoda analizy kandydatów i testów.

4. Proces odkrywania kryteriów wyszukiwania jest procesem heurystycznym. Relacja dominacji indukowana przez te kryteria nie musi być zgodna z pierwotną relacją zdefiniowaną na testach. Jest jednak w stanie zapewnić gradient wyszukiwania, który jest wystarczająco silny, aby skutecznie rozwiązać szereg problemów praktycznych. Podejście heurystyczne pozwala nam również minimalizować koszty obliczeniowe i umożliwia odkrywanie kryteriów w czasie rzeczywistym podczas ewolucji.

5. Proces odkrywania kryteriów wyszukiwania przekształca problem jednokryterialny do postaci wielokryterialnej. To z kolei pozwala uniknąć niebezpieczeństw związanych ze skalarną funkcją oceny, a w szczególności zapobiega problemowi wąskiego gardła funkcji oceny. Ponadto

podejście wielokryterialne zwiększa tzw. ewolucyjność (ang. evolvability), tj. możliwość stopniowego znajdowania coraz lepszych rozwiązań. Podejście oparte na wielu kryteriach ułatwia także analizę przetargu między potencjalnie sprzecznymi kryteriami wyszukiwania.

6. Przedstawione metody i algorytmy są całkowicie niezależne od konkretnego problemu opartego na testach, a w szczególności od reprezentacji rozwiązań kandydackich i testów. Jedynym wymaganiem jest dostęp do macierzy interakcji.

W szerszej perspektywie, wyniki przedstawione w niniejszej rozprawie stanowią istotny argument na rzecz poszukiwania alternatywnych sposobów przeszukiwania przestrzeni rozwiązań przez algorytmy heurystyczne [15, 17]. W porównaniu do badań prowadzonych nad hiperheurystykami [3], gdzie kluczowe pytanie brzmi: *jak przeprowadzić* wyszukiwanie, w proponowanym tu podejściu skupiamy się na pytaniu, *jakiej informacji* użyć do ukierunkowania procesu przeszukiwania. W wielu dziedzinach nie istnieją konceptualne ani techniczne przeszkody dla ekstrakcji szczegółowych informacji o zachowaniu rozwiązań kandydackich. W niniejszej rozprawie wykorzystaliśmy w tym celu macierze interakcji, ale istnieje też wiele inne możliwości. Biorąc pod uwagę potencjalne korzyści wynikające z takiego podejścia, które omówione zostały w niniejszej rozprawie, zdecydowanie zachęcamy czytelników do rozważenia analogicznej ścieżki postępowania, w której skalarna funkcja jakości jest zastępowana wielokryterialną metodą oceny.

8.2 Najważniejsze wyniki pracy

Najważniejsze osiągnięcia przedstawione w niniejszej rozprawie są następujące:

- Identyfikacja pułapek towarzyszących skalarnej ocenie kandydatów w problemach opartych na testach, a szczególnie zjawiska wąskiego gardła oceny, które powstaje w wyniku agregacji wyników interakcji między kandydatami a testami. [Rozdział 3]
- Koncepcja profilu jakościowego kandydata, generycznego narzędzia do wielokryterialnej oceny kandydatów opracowanych przez algorytmy ewolucyjne rozwiązujące problemy oparte na testach.
- Wprowadzenie i sformalizowanie ujednoliconego systemu automatycznego odkrywania heurystycznych funkcji oceny w problemach opartych na testach, mającego na celu poszerzenie wąskiego gardła oceny poprzez dostarczenie algorytmom wyszukiwania bogatszej informacji na temat charakterystyki rozwiązań. [Rozdział 4]
- Algorytm odkrywania funkcji oceny metodą analizy skupień (DOC) [Rozdział 5].
- Algorytm odkrywania funkcji oceny metodą nieujemnej faktoryzacji macierzy interakcji (DOF) [Rozdział 6].
- Algorytm SFIMX dedykowany redukcji kosztu oceny kandydatów w problemach opartych na testach [Rozdział 7].
- Eksperymentalna analiza proponowanych algorytmach na rzeczywistych problemach opartych na testach [Rozdziały 5 – 7].

Literatura

[1] Wolfgang Banzhaf, Guillaume Beslon, Steffen Christensen, James A Foster, François Képès, Virginie Lefort, Julian F Miller, Miroslav Radman, and Jeremy J Ramsden. Guidelines: From artificial evolution to computational evolution: a research agenda. *Nature Reviews Genetics*, 7(9):729, 2006.

[2] Sébastien Bubeck et al. Convex Optimization: Algorithms and Complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[3] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An Emerging Direction in Modern Search Technology. In *Handbook of metaheuristics*, pages 457–474. Springer, 2003.

[4] Alison Cozad, Nikolaos V Sahinidis, and David C Miller. Learning Surrogate Models for Simulation-based Optimization. *AIChE Journal*, 60(6):2211–2227, 2014.

[5] E.D. De Jong and J.B. Pollack. Ideal Evaluation from Coevolution. *Evolutionary Computation*, 12(2):159–192, 2004.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[7] A.P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley, 2007.

[8] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level Classification of Skin Cancer with Deep Neural Networks. *Nature*, 542(7639):115, 2017.

[9] Sevan Gregory Ficici. *Solution Concepts in Coevolutionary Algorithms*. PhD thesis, Computer Science Department, Brandeis University, USA, May 2004. URL http://www.demo.cs.brandeis.edu/papers/long.html#ficici_thesis_04.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Wojciech Jaśkowski and Krzysztof Krawiec. Formal Analysis, Hardness, and Algorithms for Extracting Internal Structure of Test-based Problems. *Evolutionary computation*, 19(4):639–671, 2011.

[12] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient Global Optimization of Expensive Black-box Functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[13] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-11170-5. URL http://mitpress.mit.edu/books/genetic-programming.

[14] Krzysztof Krawiec. Genetic Programming-based Construction of Features for Machine Learning and Knowledge Discovery Tasks. *Genetic Programming and Evolvable Machines*, 3(4): 329–343, December 2002. ISSN 1389-2576. doi: 10.1023/A:1020984725014.

[15] Krzysztof Krawiec. *Behavioral Program Synthesis with Genetic Programming*, volume 618 of *Studies in Computational Intelligence*. Springer International Publishing, 2016. ISBN 978-3-319-27563-5. doi: 10.1007/978-3-319-27565-9. URL http://www.springer.com/gp/book/9783319275635.

[16] Krzysztof Krawiec and Paweł Liskowski. Automatic Derivation of Search Objectives for Test-based Genetic Programming. In Penousal Machado, Malcolm I. Heywood, James McDermott, Mauro Castelli, Pablo Garcia-Sanchez, Paolo Burelli, Sebastian Risi, and Kevin Sim, editors, *18th European Conference on Genetic Programming*, volume 9025 of *LNCS*, pages 53–65, Copenhagen, 8-10 April 2015. Springer. doi: 10.1007/978-3-319-16501-1_5.

[17] Krzysztof Krawiec, Jerry Swan, and Una-May O'Reilly. Behavioral Program Synthesis: Insights and Prospects. In *Genetic Programming Theory and Practice XIII*, pages 169–183. Springer, 2016.

[18] W. B. Langdon and S. M. Gustafson. Genetic Programming and Evolvable Machines: Ten Years of Reviews. *Genetic Programming and Evolvable Machines*, 11(3/4):321–338, September 2010. ISSN 1389-2576. doi: 10.1007/s10710-010-9111-4. URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/gppubs10.pdf. Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.

[19] Hod Lipson and Jordan B Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974, 2000.

[20] Paweł Liskowski and Wojciech Jaśkowski. Accelerating Coevolution with Adaptive Matrix Factorization, (nominated to Best-paper Award). In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 457–464, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4920-8. doi: 10.1145/3071178.3071320.

[21] Paweł Liskowski and Krzysztof Krawiec. Discovery of Implicit Objectives by Compression of Interaction Matrix in Test-based Problems. In *Parallel Problem Solving from Nature–PPSN XIII*, pages 611–620. Springer International Publishing, 2014.

[22] Paweł Liskowski and Krzysztof Krawiec. Surrogate Fitness Via Factorization of Interaction Matrix (Best-paper Award winner). In Malcolm I. Heywood, James McDermott, Mauro Castelli, and Ernesto Costa, editors, *EuroGP 2016: Proceedings of the 19th European Conference on Genetic Programming*, volume 9594 of *LNCS*, pages 65–79, Porto, Portugal, 30 March–1 April 2016. Springer Verlag.

[23] Paweł Liskowski and Krzysztof Krawiec. Non-negative Matrix Factorization for Unsupervised Derivation of Search Objectives in Genetic Programming. In Tobias Friedrich, editor, *GECCO '16: Proceedings of the 2016 Annual Conference on Genetic and Evolutionary Computation*, pages 749–756, Denver, USA, 20-24 July 2016. ACM. doi: 10.1145/2908812.2908888.

[24] Paweł Liskowski and Krzysztof Krawiec. Online Discovery of Search Objectives for Test-based Problems. *Evolutionary Computation*, 25(3):375–406, 2016.

[25] Paweł Liskowski and Krzysztof Krawiec. Segmenting Retinal Blood Vessels with Deep Neural Networks. *IEEE transactions on medical imaging*, 35(11):2369–2380, 2016.

[26] Paweł Liskowski and Krzysztof Krawiec. Adaptive Test Selection for Factorization-based Surrogate Fitness in Genetic Programming. *Foundations of Computing and Decision Sciences*, 42(4):339–358, 2017.

[27] S. Nolfi and D. Floreano. Coevolving Predator and Prey Robots: Do ąrms RacesĄrise in Artificial Evolution? *Artificial Life*, 4(4):311–335, 1998.

[28] Ovid and Charles Martin. *Metamorphoses*. W.W. Norton, 2004.

[29] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. URL http://www.gp-field-guide.org.uk. (With contributions by J. R. Koza).

[30] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

[31] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the Game of Go without Human Knowledge. *Nature*, 550(7676):354, 2017.

[32] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the Gap to Human-level Performance in Face Verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[33] Marco Tomassini, Leonardo Vanneschi, Philippe Collard, and Manuel Clergue. A Study of Fitness Distance Correlation As a Difficulty Measure in Genetic Programming. *Evolutionary Computation*, 13(2):213–239, Summer 2005. ISSN 1063-6560. doi: 10.1162/1063656054088549.

Instytut Informatyki, Wydział Informatyki
Politechnika Poznańska

Skład przy użyciu systemu LaTeX.

BibTeX:

```
@PHDTHESIS{LiskowskiPhd2018,
    author  = {Paweł Liskowski},
    title   = {Heurystyczne Algorytmy Odkrywania Funkcji Oceny w Problemach Opartych na Testach},
    school  = {Poznan University of Technology},
    address = {Poznan, Poland},
    year    = {2018}
}
```