POLITECHNIKA WARSZAWSKA

INFORMATYKA TECHNICZNA I TELEKOMUNIKACJA DZIEDZINA NAUK INŻYNIERYJNO-TECHNICZNYCH

Rozprawa doktorska

mgr inż. Mateusz Zaborski

Efektywne zastosowania metamodeli w algorytmach populacyjnych przeznaczonych do rozwiązywania problemów optymalizacji ciągłej

Promotor prof. dr hab. inż. Jacek Mańdziuk Promotor pomocniczy dr inż. Michał Okulewicz

WARSZAWA 2022

Podziękowania

Dziękuję mojemu promotorowi prof. dr. hab. inż. Jackowi Mańdziukowi za poświęcony czas, motywację, oraz ogół wiedzy, którą mi przekazał.

Dziękuję również mojemu promotorowi pomocniczemu dr. inż. Michałowi Okulewiczowi za wsparcie w badaniach naukowych oraz wszelką pomoc, którą zawsze był gotów nieść.

Szczególnie dziękuję moim rodzicom. Za wszystko.

Nikt nie jest kontent ze swej fortuny, każdy – ze swego rozumu Lew Tołstoj, Anna Karenina

Streszczenie

Rozprawa dotyczy efektywnego zastosowania metamodeli w algorytmach populacyjnych przeznaczonych do rozwiązywania problemów optymalizacji ciągłej. Algorytmy populacyjne stanowią wiodące metody rozwiązujące problemy czarnoskrzynkowe, tzn. takie w których jedyną możliwością interakcji z funkcją celu (f.c.) jest dokonanie jej ewaluacji. Algorytmy populacyjne należą do metod niedeterministycznych i na ogół zakładają relatywnie duży budżet optymalizacji (dostępną liczbę ewaluacji f.c.). Wykorzystanie dużych budżetów optymalizacji jest niemożliwe w przypadku optymalizacji kosztownej, która zakłada znaczący czas wymagany na pojedynczą ewaluację f.c.

Metamodele odwzorowują f.c. oraz pozwalają zastąpić jej kosztowną ewaluację za pomocą przybliżonej wartości w zadanym punkcie przestrzeni przeszukiwań. Dzięki temu możliwe jest ograniczenie liczby wykonanych ewaluacji f.c. lub poprawa otrzymanego rozwiązania przy zadanym budżecie optymalizacji. Metamodele znajdują zastosowanie w wielu metodach rozwiązujących problemy optymalizacji, w szczególności w algorytmach populacyjnych.

Rozprawa skupia się na efektywnych zastosowaniach metamodeli w algorytmach populacyjnych, tzn. takich które poprawiają wyniki algorytmu, są wydajne obliczeniowo oraz są skierowane do rozwiązywania szerokiej klasy problemów.

W pierwszej części rozprawy dokonano wprowadzenia do problemów optymalizacji ze szczególnym uwzględnieniem kosztu ewaluacji f.c., mającego wpływ na założony budżet optymalizacji. Wprowadzono pojęcie optymalizacji semikosztownej, która zakłada budżet optymalizacji pomiędzy $10^2 \cdot D$ a $10^4 \cdot D$ ewaluacji f.c., gdzie D jest liczbą wymiarów problemu. Stąd, dla optymalizacji kosztownej przyjęto budżet $\leq 10^2 \cdot D$, a dla taniej $\geq 10^4 \cdot D$ ewaluacji f.c. Przedstawiono metody ewaluacji algorytmów, z uwzględnieniem zbiorów testowych CEC2021 oraz COCO. Opisano wiodące metody rozwiązywania problemów optymalizacji ciągłej. Skupiono się na algorytmach populacyjnych powszechnie uznawanych za skuteczne metody rozwiązujące, takich jak te oparte o adaptacyjną ewolucję różnicową oraz adaptację macierzy kowariancji. Następnie omówiono popularne grupy metamodeli oraz znane ich zastosowania w algorytmach populacyjnych. Rozprawa dyskutuje efektywność obecnie stosowanych metod oraz prezentuje autorskie badanie eksperymentalne czasu estymacji parametrów znanych metamodeli w zależności od wielkości zbioru uczącego oraz liczby wymiarów problemu.

Autor rozprawy proponuje cztery zastosowania metamodeli: (1) inicjalizacja metamodelem, (2) lokalna optymalizacja metamodelem, (3) preselekcja rozwiązań na podstawie wartości metamodelu oraz (4) rekurencyjna estymacja parametrów metamodelu. We wszystkich ww. rozwiązaniach jako metamodel wybrano regresję wielomianową. Zdecydowano się rozszerzać za pomocą metamodeli algorytmy: GAPSO, SHADE, R-SHADE oraz L-SHADE. Wykorzystano lokalną optymalizację metamodelem jako mechanizm poprawiający wyniki algorytmów populacyjnych w optymalizacji taniej. Wykorzystano preselekcję rozwiązań za pomocą metamodelu jako mechanizm poprawiający wyniki algorytmów populacyjnych w optymalizacji semikosztownej. Skutkiem tego, autor rozprawy przedstawia pięć algorytmów populacyjnych wykorzystujących metamodel: M-GAPSO, SHADE-LM, LQ-R-SHADE, psLSHADE oraz rmmLSHADE. M-GAPSO oraz SHADE-LM wykorzystują lokalną optymalizację metamodelem oraz inicjalizację metamodelem. LQ-R-SHADE, psLSHADE oraz rmmLSHADE są oparte o mechanizm preselekcji rozwiązań na podstawie wartości metamodelu. Dodatkowo LQ-R-SHADE korzysta z inicjalizacji metamodelem. Parametry metamodelu w rmmLSHADE są estymowane w sposób rekurencyjny za pomocą rekursywnych najmniejszych kwadratów, dzięki czemu nie jest wymagane konstruowanie zbioru uczącego.

Wszystkie algorytmy populacyjne wspomagane metamodelem zostały poddane eksperymentalnej ewaluacji z wykorzystaniem zbioru testowego CEC2021 lub COCO. W każdym przypadku wykazano zasadność z zastosowania metamodelu oraz omówiono wpływ takiej integracji na czas obliczeń algorytmu.

Rozprawę kończy dyskusja uzyskanych wyników, z uwzględnieniem efektywności lokalnej optymalizacji wspomaganej metamodelem oraz efektywności preselekcji rozwiązań. Przedstawiono możliwe dalsze kierunki badań.

Słowa kluczowe: Optymalizacja, Algorytmy Ewolucyjne, Metamodel, Model zastępczy, Metaheurystyki, Ewolucja Różnicowa

Abstract

Title: Effective applications of meta-models in population-based algorithms designed to solve continuous optimization problems

This thesis presents the effective applications of meta-models in population algorithms designed to solve continuous optimization problems. Population algorithms are leading methods for solving black-box problems, i.e., those in which the only way to interact with the fitness function is to evaluate it. Population algorithms belong to non-deterministic methods and generally assume a relatively large optimization budget, i.e. the available number of fitness function evaluations (FFEs). Using large optimization budgets is not possible in the case of expensive optimization, which assumes a significant time required for a single FFE.

Meta-models mimic fitness functions and allow replacing their expensive evaluation with an approximate value at a given point in the search space. Thus, reducing the number of FFEs performed or improving the final solution with a fixed optimization budget is possible. Meta-models are used in various methods for solving optimization problems, especially in population-based algorithms.

The dissertation focuses on effective applications of meta-models in population algorithms, i.e., those that improve the algorithm's performance, are computationally efficient, and are aimed at solving a wide class of problems.

The first part of the thesis contains an introduction to optimization problems, focusing on the impact of the FFE cost on the assumed optimization budget. The concept of semi-expensive optimization is presented. The semi-expensive optimization assumes an optimization budget between $10^2 \cdot D$ and $10^4 \cdot D$ of FFEs, where D is the problem dimensionality. Hence, for expensive optimization, a budget of $\leq 10^2 \cdot D$ is assumed, and for cheap optimization, $\geq 10^4 \cdot D$ FFEs, respectively. Experimental algorithm evaluation methods are presented, considering the CEC2021 and COCO test sets. Leading methods for solving continuous optimization problems are described. The focus is on population algorithms widely recognized as efficient solution methods, such as those based on adaptive differential evolution and covariance matrix adaptation. Then, the thesis describes popular groups of meta-models and their known applications in population algorithms. The thesis discusses the effectiveness of currently used methods and presents the author's experimental study of parameters estimation time of known meta-models depending on the size of the training set and the number of problem dimensions.

The author of the thesis proposes four applications of meta-models: (1) meta-model initialization, (2) local meta-model optimization, (3) preselection of solutions based on meta-model values and (4) recursive estimation of meta-model parameters. In all of the above cases, polynomial regression was chosen as the meta-model. The following algorithms were extended with meta-models: GAPSO, SHADE, R-SHADE, and L-SHADE. Local meta-model optimization was used as a mechanism to improve the performance of population algorithms in cheap optimization. Preselection of solutions based on metamodel is designed to improve the population algorithms performance in semi-expensive optimization. As a result, the author presents five population algorithms utilizing the meta-model: M-GAPSO, SHADE-LM, LQ-R-SHADE, psLSHADE and rmmLSHADE. M-GAPSO and SHADE-LM use local meta-model optimization and meta-model initialization. LQ-R-SHADE, psLSHADE and rmmLSHADE utilize a preselection of solutions based on meta-model values. In addition, LQ-R-SHADE is enhanced with meta-model initialization. The meta-model parameters in rmmLSHADE are estimated recursively using the Recursive Least Squares filter, so no training set construction is required.

All proposed algorithms were experimentally evaluated using the CEC2021 or COCO test set. In each case, the advantage from meta-model incorporation is demonstrated. Furthermore, the impact of such meta-model integration on the algorithm's computation time is discussed.

The dissertation concludes with a discussion of the obtained results, including the efficiency of local meta-model optimization and the preselection of solutions based on meta-model values. Finally, possible further research directions are presented.

Key words: Optimization, Evolutionary Algorithms, Meta-model, Surrogate model, Metaheuristics, Differential Evolution

Spis treści

1	Wstęp							
	1.1	Motyw	vacja	13				
	1.2	Cel i zakres rozprawy						
	1.3	Hipotezy badawcze						
	1.4	Spis a	utorskich publikacji	19				
	1.5	.5 Układ rozprawy						
2	Wprowadzenie do problemów optymalizacji							
	2.1	1 Problem optymalizacji		25				
	2.2	2 Systematyka problemów optymalizacji						
	2.3	Optymalizacja czarnoskrzynkowa						
	2.4	Ewaluacja metod rozwiązujących						
		2.4.1	Jakość rozwiązania	29				
		2.4.2	Wykorzystywane zasoby	30				
		2.4.3	Zbiór testowy CEC2021	32				
		2.4.4	Zbiór testowy COCO	34				
		2.4.5	Budżety optymalizacji w popularnych zbiorach testowych	37				
	2.5	Koszt optymalizacji		37				
		2.5.1	Optymalizacja kosztowna	38				
		2.5.2	Optymalizacja semikosztowna	39				
3	Wiodące metody rozwiązywania problemów optymalizacji ciągłej							
	3.1	Systematyka metod rozwiązujących						
	3.2	Algorytmy populacyjne						
	3.3	tmy z rodziny DE	48					
		3.3.1	Bazowy DE $(DE/current-to-best/1)$	48				

		3.3.2	Rozszerzenia DE	49
		3.3.3	Mechanizm adaptacji parametrów (SHADE)	51
		3.3.4	Mechanizm liniowej redukcji rozmiaru populacji (L-SHADE)	54
		3.3.5	Mechanizm restartów (R-SHADE)	54
	3.4	Algory	ytmy z rodziny CMA-ES	55
		3.4.1	Bazowy CMA-ES	55
		3.4.2	Rozszerzenia CMA-ES	58
4	Mo	delowa	nie funkcji celu w optymalizacji ciągłej	59
	4.1	Przyb	liżanie funkcji celu	59
	4.2	Metar	nodele w optymalizacji	62
	4.3	Popul	arne grupy metamodeli	63
		4.3.1	Regresja wielomianowa	64
		4.3.2	Kriging	66
		4.3.3	Sztuczne sieci neuronowe	68
		4.3.4	Radialne funkcje bazowe	71
	4.4	Optyn	nalizacja bayesowska	72
	4.5	Algory	ytmy populacyjne wspierane metamodelem (APWM) $\ . \ . \ . \ .$	74
		4.5.1	Metody integracji metamodeli	75
		4.5.2	Metamodele w algorytmach z rodziny DE $\hfill .$	77
		4.5.3	Metamodele w algorytmach z rodziny CMA-ES $\ .$	78
		4.5.4	Metamodele w innych algorytmach populacyjnych	80
5	Zas	tosowa	nia metamodeli w algorytmach populacyjnych	81
	5.1	Obecr	ie stosowane metody	81
		5.1.1	Narzut obliczeniowy metamodelu	83
		5.1.2	Eksperymentalne badanie czasu estymacji parametrów metamodeli	85
		5.1.3	Dyskusja efektywności zastosowań metamodeli	88
	5.2	Propo	nowane zastosowania metamodeli	90
		5.2.1	Inicjalizacja metamodelem	92
		5.2.2	Lokalna optymalizacja metamodelem	93
		5.2.3	Preselekcja rozwiązań na podstawie wartości metamodelu	94
		5.2.4	Rekurencyjna estymacja parametrów metamodelu	95

6	6 Lokalna optymalizacja metamodelem w optymalizacji taniej		99			
	6.1	ytm GAPSO	. 99			
	6.2	M-GA	PSO: GAPSO z lokalną optymalizacją metamodelem	. 101		
		6.2.1	Opis algorytmów w M-GAPSO	. 103		
		6.2.2	Charakterystyka metamodeli w M-GAPSO	. 104		
		6.2.3	Eksperymentalna ewaluacja	. 106		
	6.3	SHAD	DE-LM: R-SHADE z lokalną optymalizacją metamodelem	. 111		
		6.3.1	Charakterystyka metamodelu w SHADE-LM	. 112		
		6.3.2	Eksperymentalna ewaluacja	. 114		
	6.4	Narzu	t obliczeniowy lokalnej optymalizacji metamodelem	. 115		
7	Preselekcja rozwiązań w optymalizacji semikosztownej					
	7.1	LQ-R-	-SHADE: R-SHADE wspierany globalnym metamodelem	. 119		
		7.1.1	Lokalna preselekcja rozwiązań w LQ-R-SHADE	. 120		
		7.1.2	Charakterystyka metamodelu w LQ-R-SHADE	. 122		
		7.1.3	Eksperymentalna ewaluacja	. 122		
	7.2	psLSHADE: LSHADE wspierany globalnym metamodelem				
		7.2.1	Charakterystyka metamodelu w psLSHADE	. 125		
		7.2.2	Eksperymentalna ewaluacja	. 125		
		7.2.3	Badanie skuteczności preselekcji	. 129		
	7.3 rmmLSHADE: LSHADE wspierany rekurencyjnie es		SHADE: LSHADE wspierany rekurencyjnie estymowanym global-			
nym metamodelem		nym n	netamodelem	. 136		
		7.3.1	Globalna preselekcja rozwiązań w rmmLSHADE	. 136		
		7.3.2	Charakterystyka metamodelu w rmmLSHADE	. 138		
		7.3.3	Eksperymentalna ewaluacja	. 139		
		7.3.4	Wpływ czynnika zapominania na zbieżność algorytmu	. 141		
	7.4	Narzu	t obliczeniowy preselekcji rozwiązań	. 143		
8	Poc	Podsumowanie 14				
	8.1	Dyskusja wyników badań		. 146		
		8.1.1	Dyskusja efektywności lokalnej optymalizacji metamodelem	. 147		
		8.1.2	Dyskusja efektywności preselekcji rozwiązań	. 148		
	8.2	Weryfikacja hipotez badawczych				
	8.3	Dalsze kierunki badań				
	8.4	Autorski wkład w dziedzinę				

Lista skrótów

APWM Algorytm populacyjny wspierany metamodelem CMA-ES Strategia ewolucyjna wykorzystująca adaptację macierzy kowariancji (ang. Covariance Matrix Adaptation Evolution Strategy) **DE** Ewolucja różnicowa (ang. *Differential Evolution*) EGO Efektywna globalna optymalizacja (ang. Efficient Global Optimization) JADE Adaptacyjna ewolucja różnicowa z opcjonalnym archiwum zewnętrznym (ang. Adaptive Differential Evolution with Optional External Archive) **ELM** Maszyny ekstremalnego uczenia (ang. *Extreme Learning Machines*) **f.c.** Funkcja celu GAPSO Uogólniona adaptacyjna optymalizacja rojem cząstek (ang. Generalized Adaptive Particle Swarm Optimization) **LPSR** Liniowa redukcja rozmiaru populacji (ang. *Linear Population Size Reduction*) **MLP** Perceptron wielowarstwowy (ang. *Multilayer Perceptron*) **MRFO** Wielowymiarowy czerwony lis (ang. *Multidimensional Red Fox meta-heuristic*) **MNK** Metoda najmniejszych kwadratów (ang. *Multilayer Perceptron*) **OB** Optymalizacja bayesowska (ang. *Bayesian Optimization*) **PSO** Optymalizacja rojem cząstek (ang. *Particle Swarm Optimization*) **RLS** Rekursywne najmniejsze kwadraty (ang. *Recursive Least Squares*) **RW** Regresja wielomianowa (ang. *Polynomial Regression*) SHADE Oparta o sukces adaptacja parametrów dla ewolucji różnicowej (ang. Success-History Based Parameter Adaptation for Differential Evolution) SSN Sztuczna sieć neuronowa (ang. Artificial Neural Network)

Rozdział 1

Wstęp

1.1 Motywacja

Niniejsza rozprawa dotyczy efektywnego zastosowania metamodeli w problemach optymalizacji ciągłej. W ogólności, zadanie optymalizacji polega na znalezieniu optimum zadanej funkcji celu (f.c.). Ze względu na różną charakterystykę zadań optymalizacji istnieje szereg metod rozwiązujących poszczególne problemy [62]. Badania w optymalizacji są prowadzone nie tylko w obrębie projektowania metod rozwiązujących znane problemy, ale także w zakresie definiowania nowych problemów, zaczerpniętych z innych dziedzin. Wskutek tego, badania poświęcone optymalizacji mają charakter interdyscyplinarny.

Część zadań optymalizacji można rozwiązać w sposób analityczny, tzn. znajdując dokładne rozwiązanie za pomocną wyprowadzonego wzoru. Jednak, dla większości zadań rozwiązanie analityczne nie istnieje lub jego wykorzystanie jest zbyt pracochłonne. W takim przypadku zastosowanie znajdują metody numeryczne. Metody numeryczne dzielą się na dwie grupy: deterministyczne i niedeterministyczne. Metody numeryczne deterministyczne wykorzystują niezmienny w obrębie danej metody algorytm i pozwalają na znalezienie dokładnego rozwiązania. Dokładność znalezionego rozwiązania jest uzależniona od precyzji maszynowej i związanego z nią błędu numerycznego. Podobnie jak w przypadku metod analitycznych, części zadań optymalizacji nie da się rozwiązać za pomocą metody numerycznej deterministycznej lub jest to zbyt kosztowne obliczeniowo. W takim przypadku, stosuje się metody numeryczne niedeterministyczne, które opierają się o losowość. Każde uruchomienie algorytmu niedeterministycznego prowadzi w ogólności do znalezienia innego rozwiązania, co nie gwarantuje znalezienia rozwiązania dokładnego.

Rozwój komputerów, a w szczególności dostępnej mocy obliczeniowej sprawił, że meto-

dy numeryczne zyskują na znaczeniu. Coraz większe zadania optymalizacji stają się możliwe do rozwiązania w racjonalnym czasie za pomocą metod deterministycznych. Równocześnie, metody niedeterministyczne za sprawą coraz większych budżetów obliczeniowych pozwalają na uzyskanie satysfakcjonujących rozwiązań.

W rozprawie skupiono się na globalnej optymalizacji ciągłej bez ograniczeń, z funkcją dotyczącą jednego celu. Kolejnym założeniem jest to, że mamy do czynienia z optymalizacją czarnoskrzynkową (ang. *black-box*) [12], tzn. niedostępna jest jakakolwiek informacja o postaci funkcji, w tym o jej gradiencie. Jedyny sposób interakcji z funkcją celu odbywa się poprzez dokonanie jej ewaluacji, a tym samym poznanie jej wartości w danym punkcie. W praktyce większość problemów nie jest w pełni czarnoskrzynkowych, tzn. trudno przy-jąć założenie o braku jakiejkolwiek intuicji o kształcie f.c. bądź jej wartościach. Niemniej, założenie o procesie optymalizacji realizowanym w sposób czarnoskrzynkowy pozwala na konstrukcję uniwersalnych metod rozwiązujących.

Zawężenie problematyki badawczej do konkretnego rodzaju optymalizacji determinuje zbiór metod rozwiązujących, które są szczególnie użyteczne. Kluczową z punktu widzenia tej rozprawy rodziną algorytmów są algorytmy populacyjne - metaheurystyki należące do metod numerycznych niedeterministycznych. W skład algorytmów populacyjnych wchodzą algorytmy ewolucyjne [225, 201] oraz algorytmy rojowe [34]. W algorytmie populacyjnym każdy osobnik reprezentuje punkt w przestrzeni rozwiązań, czyli rozwiązanie zadanej f.c. W każdej iteracji algorytmu populacja zmienia swoje położenie, czemu towarzyszą ewaluacje f.c. Reguły zmiany położenia poszczególnych osobników są określone przez konkretny algorytm i mają charakter niedeterministyczny. Poruszanie się populacji w przestrzeni rozwiązań jest motywowane dwoma celami: przeszukiwaniem tej przestrzeni pod kątem obiecujących rozwiązań (eksploracja) oraz kierowaniem się całej populacji do optimum f.c. (eksploatacja) [42].

Istnieje wiele mechanizmów działania, na których oparte są algorytmy populacyjne [45]. W myśl zasady *nie ma darmowych obiadów* [230] nie jest możliwe jednoznaczne wskazanie uniwersalnie najlepszego algorytmu. Co więcej, wiele relatywnie prostych metaheurystyk [110, 212] jest wciąż z powodzeniem stosowanych ze względu na łatwość implementacji oraz przewidywalność działania. Jednakże, współczesne badania eksperymentalne zostały zdominowane [1, 2, 3, 205] przez dwie rodziny algorytmów: oparte o adaptacyjną ewolucję różnicową (np. JADE [251], SHADE [215]) oraz oparte o adaptację macierzy kowariancji (np. CMA-ES [79], IPOP-CMA-ES [13], BIPOP-CMA-ES [75]).

Użyteczność algorytmów populacyjnych wynika przede wszystkim z ich niskiej złożo-

ności obliczeniowej. Czas trwania procesu optymalizacji zależy od dwóch czynników: czasu wymaganego na pojedynczą ewaluację f.c. oraz towarzyszącego jej narzutu czasowego wynikającego z obliczeń samego algorytmu. Całkowity czas trwania procesu optymalizacji dla większości algorytmów populacyjnych zależy liniowo od budżetu optymalizacji rozumianego jako maksymalna dostępna liczba ewaluacji f.c.

Mimo, że narzut czasowy silnie zależy od specyfiki algorytmu oraz wykorzystywanych w nim operacji algebraicznych, to analizując wartości bezwzględne zazwyczaj jest on niewielki. Przykładowo, dla jednego ze współczesnych i wysoce efektywnych algorytmów opartych o adaptację macierzy kowariancji [23] narzut czasowy przypadający na pojedynczą ewaluację f.c. wynosi < 0.1ms. Skutkiem tego, w przypadku algorytmów populacyjnych proces optymalizacji zawierający 10^6 i więcej ewaluacji f.c., nie stanowi wyzwania obliczeniowego dla samego algorytmu. W badania eksperymentalnych algorytmów populacyjnych również na ogół przyjmuje się, że takie budżety optymalizacji są osiągalne. Dla przykładu, popularne zbiory testowe (COCO BBOB [74], CEC2021 [152] oraz CEC2022 [118]) dla problemów 10-wymiarowych zakładają budżet $2 \cdot 10^5 - 10^7$ ewaluacji f.c. Wykonanie dużej liczby ewaluacji f.c. jest możliwe w sytuacji, kiedy nie tylko sama metoda rozwiązująca jest szybka, ale także czas pojedynczej ewaluacji f.c. jest niewielki. Optymalizacja zakładająca relatywnie mały czas ewaluacji f.c. jest określana optymalizacją tania [71, 126, 198] (ang. *cheap optimization*).

W rzeczywistości istnieje szeroka grupa problemów, w których pojedyncza ewaluacja f.c. jest kosztowna, tzn. czas ewaluacji jest znaczący. Taki rodzaj optymalizacji nazywany jest optymalizacją kosztowną (ang. *expensive optimization*). Przykładem tego są symulacje, w szczególności związane z eksperymentami fizycznymi, gdzie czas pojedynczej ewaluacji f.c. jest liczony w godzinach. Strojenie parametrów w systemach uczenia maszynowego również może być zaliczane do optymalizacji kosztownej [197]. Konsekwencją kosztownej ewaluacji f.c. jest ograniczenie budżetu optymalizacji do setek albo tysięcy ewaluacji f.c. [248]. Za punkt odniesienia może posłużyć zbiór testowy CEC2005 dedykowany dla optymalizacji kosztownej [36], który zakłada budżet optymalizacji 5 · 10² ewaluacji f.c. dla D = 10, gdzie D to liczba wymiarów problemu.

Optymalizacja kosztowna wymaga istotnie różnej strategii rozwiązującej niż ta zapewniona przez algorytmy ewolucyjne. Większość metod rozwiązujących problemy kosztowne wykorzystuje metamodel [207, 22, 181, 63, 226], zwany inaczej modelem zastępczym (ang. *surrogate model*), który przybliża funkcję celu. Jest to motywowane tym, że ewaluacja metamodelu, tj. uzyskanie wartości metamodelu w danym punkcie, jest nieporównywalnie szybsza od ewaluacji rzeczywistej f.c.

Wiodącą metodą rozwiązującą problemy optymalizacji kosztownej jest *optymalizacja bayesowska* [150], która zakłada wykorzystanie metamodelu *Kriginga* [146]. Optymalizacja bayesowska jest metodą sekwencyjną, która w każdej iteracji wyznacza nowe rozwiązanie poddawane ewaluacji f.c. oraz aktualizuje (uczy) metamodel. Kriging jest budowany w oparciu o wszystkie rozwiązania, które były poddane ewaluacji od początku procesu optymalizacji. Wskutek tego, to podejście dokładnie odwzorowuje f.c., lecz bardzo słabo skaluje się ze względu na liczbę jej ewaluacji.

Zestawienie optymalizacji bayesowskiej z algorytmami populacyjnymi uwidacznia znaczącą lukę, nie tylko w kontekście właściwości tych metod rozwiązujących, ale także w kontekście zakładanych budżetów optymalizacji. Dlatego też, w rozprawie wprowadzono pojęcie optymalizacji semikosztownej, która zakłada budżet optymalizacji znajdujący się pomiędzy budżetem optymalizacji kosztownej a budżetem optymalizacji taniej.

W literaturze zaproponowano wiele algorytmów populacyjnych, które wykorzystują metamodel [136, 126, 101]. Metamodele mogą dokonywać preselekcji rozwiązań, celem ewaluacji tylko tych najbardziej obiecujących [56, 222, 77, 242, 241, 243]. Mogą też modyfikować mechanizmy działania algorytmów populacyjnych [182, 10, 4]. Możliwe też jest pełne zastąpienie f.c. metamodelem w obrębie całej iteracji algorytmu [18, 174].

Część algorytmów populacyjnych jest łączonych z optymalizację bayesowską [101, 55], co sprawia, że są one przeznaczone do rozwiązywania problemów optymalizacji kosztownej. Warto zaznaczyć, że Kriging może być stosowany w algorytmach populacyjnych w sposób istotnie inny niż ten określony przez optymalizację bayesowską [131, 18, 174]. Przybliżanie f.c. może być też realizowane przez inne grupy metamodeli, spośród których należy wymienić regresję wielomianową [117, 112, 218], sztuczne sieci neuronowe [98, 86, 148, 195] oraz radialne funkcje bazowe [124, 240, 122, 184].

W większości przypadków zastosowanie metamodeli w algorytmach populacyjnych służy rozwiązywaniu problemów optymalizacji kosztownej. Niemniej, w literaturze można znaleźć algorytmy populacyjne wspierane metamodelem, które wykraczają poza budżet optymalizacji kosztownej. Szczególną uwagę należy zwrócić na algorytm lq-CMA-ES [77], który modeluje f.c. za pomocą regresji wielomianowej. Metamodel jest uczony w oparciu o archiwum próbek o ograniczonym rozmiarze, do którego trafiają kolejno ewaluowane rozwiązania i odpowiadające im wartości f.c. Zastosowanie regresji wielomianowej pozwoliło nie tylko poprawić wyniki bazowego algorytmu CMA-ES, ale także były one lepsze od wyników uzyskanych z pomocą znacznie bardziej złożonych metamodeli. Podsumowując, metamodele znalazły zastosowanie w algorytmach populacyjnych i z powodzeniem służą do rozwiązywania problemów optymalizacji kosztownej. Znane są algorytmy populacyjne, które zakładają wykorzystanie budżetów optymalizacji większych niż zakłada optymalizacja kosztowna, lecz prace nad nimi nie stanowią wiodącego nurtu badawczego. Skuteczne zastosowanie metamodelu regresji wielomianowej w CMA-ES stanowi jeden z argumentów za tym, że jest to obiecujący kierunek badawczy. Jednakże, trudno znaleźć podobne rozwiązanie w drugiej wiodącej grupie algorytmów populacyjnych, tj. algorytmach opartych o adaptacyjną ewolucję różnicową.

1.2 Cel i zakres rozprawy

Celem badawczym rozprawy jest zbadanie możliwości efektywnego zastosowania metamodeli w algorytmach populacyjnych przeznaczonych do rozwiązywania problemów optymalizacji ciągłej. Poprzez efektywne zastosowanie metamodelu w algorytmie populacyjnym rozumie się takie zastosowanie, które:

- 1. Osiąga zadowalające wyniki.
- 2. Jest uniwersalne.
- 3. Umożliwia osiągnięcie relatywnie dużych budżetów optymalizacji rozumianych jako liczba ewaluacji f.c.
- 4. Cechuje się akceptowalnym narzutem obliczeniowym, wynikającym z zastosowania metamodelu.

Szczegółowa dyskusja efektywności zastosowania metamodeli w algorytmach populacyjnych jest zawarta w rozdziale 5. Główny cel badawczy rozprawy został zdekomponowany na trzy mniejsze cale badawcze, odnoszące się do możliwych efektywnych zastosowań metamodeli w algorytmach populacyjnych:

- C1. Opracowanie sposobu wykorzystania metamodeli w optymalizacji taniej, celem wyznaczania punktów do ewaluacji z wykorzystaniem f.c.
- C2. Opracowanie wydajnych metaheurystyk opartych o adaptacyjną ewolucję różnicową wykorzystujących metamodel przeznaczonych do optymalizacji semikosztownej.
- C3. Opracowanie metody rekurencyjnej estymacji parametrów metamodelu oraz eliminacji archiwum próbek.

Zakres rozprawy jest określony przez badany problem oraz zbiór metod służących do jego rozwiązania. Badanym problemem jest czarnoskrzynkowa globalna optymalizacja ciągła z jednym celem, w której jedynym ograniczeniem jest dostępna przestrzeń przeszukiwań. Szczegółowa definicja problemu jest zwarta w rozdziale 2.

W rozprawie skupiono się na algorytmach populacyjnych, które stanowią wiodącą grupę metod rozwiązujących. W kontekście przedstawionych celów oraz badań eksperymentalnych zbiór badanych algorytmów populacyjnych został ograniczony do wiodących metod, tj. metod opartych o adaptacyjną ewolucję różnicową oraz metod opartych o adaptację macierzy kowariancji. Jest to umotywowane tym, że powyższe grupy metaheurystyk stanowią dominujące podejścia z punktu widzenia wyników osiąganych na współcześnie stosowanych zbiorach testowych [205, 152].

1.3 Hipotezy badawcze

Zaproponowane cele badawcze służą eksperymentalnej weryfikacji poniższych czterech hipotez badawczych:

- H1. Jest możliwe zastosowanie w algorytmie populacyjnym metamodeli służących do bezpośredniego wyznaczania rozwiązań poddawanych ewaluacji z wykorzystaniem funkcji celu, co prowadzi do poprawy wyników algorytmu w optymalizacji taniej.
- H2. Jest możliwe zastosowanie globalnego metamodelu służącego do preselekcji obiecujących rozwiązań w algorytmie populacyjnym opartym o adaptacyjną ewolucję różnicową w sposób umożliwiający osiąganie lepszych wyników w optymalizacji semikosztownej.
- H3. Jest możliwe zaprojektowanie mechanizmu preselekcji obiecujących rozwiązań w sposób niewymagający modyfikacji mechanizmów adaptacji parametrów.
- H4. Jest możliwe zintegrowanie metamodelu z algorytmem populacyjnym, w taki sposób, że będzie on estymowany rekurencyjnie, a tym samym użycie archiwum będzie niepotrzebne.

1.4 Spis autorskich publikacji

Część wyników prac badawczych zaprezentowanych w tej rozprawie zostało uprzednio opublikowanych (bądź jest w trakcie druku) w pracach współautorskich autora rozprawy oraz promotora lub promotora pomocniczego. Poniżej zaprezentowano listę prac wraz z krótkim opisem ich treści.

Prace opublikowane:

M.Uliński, A.Żychowski, M.Okulewicz, M.Zaborski, H.Kordulewski, (2018), Generalized Self-Adapting Particle Swarm Optimization algorithm [221]

Pokazaliśmy, że jest możliwa generalizacja reguł opisujących zachowanie osobników w algorytmach populacyjnych. Zaproponowaliśmy algorytm GAPSO, w którym poszczególne osobniki mają przypisywane w każdej iteracji różne reguły zachowań. Reguły zachowań pochodzą z popularnych metaheurystyk (PSO oraz DE). Wybór reguły zachowania odbywa się w oparciu o dotychczasową wydajność każdej z reguł celem promowania tych, które w przeszłości doprowadzały do relatywnie większej poprawy wyniku.

2. M.Zaborski, M.Okulewicz, J.Mańdziuk (2019) Generalized Self-Adapting Particle Swarm Optimization algorithm with model-based optimization enhancements [244]

Praca rozszerza koncepcję algorytmu GAPSO (wprowadzając algorytm M-GAPSO) o mechanizm pamięci oraz dodatkowe reguły zachowań, nazywane w rozprawie lokalną optymalizacją metamodelem. Pokazaliśmy, że lokalne metamodele (kwadratowe oraz wielomianowe) mogą być hybrydyzowane z algorytmami populacyjnymi oraz służyć bezpośrednio do wyznaczania kolejnych punktów do ewaluacji. Zaproponowano dwie metody tworzenia zbioru uczącego w zależności od rodzaju metamodelu.

 M.Zaborski, M.Okulewicz, J.Mańdziuk, (2020) Analysis of statistical model-based optimization enhancements in Generalized Self-Adapting Particle Swarm Optimization framework [245]

W pracy dogłębniej przeanalizowaliśmy właściwości zachowań opartych o lokalne metamodele będące elementem algorytmu M-GAPSO. W szczególności pokazaliśmy jakie punkty są wyznaczane do ewaluacji oraz jak zmienia się rozproszenie punktów należących do zbioru uczącego podczas całego procesu optymalizacji. 4. M.Okulewicz, M.Zaborski, (2021) Benchmarking SHADE algorithm enhanced with model based optimization on the BBOB noiseless testbed [164]

W pracy pokazaliśmy, że wydajny algorytm SHADE, wykorzystujący adaptację parametrów, może być hybrydyzowany z lokalnymi metamodelami (kwadratowym oraz kwadratowym wraz z interakcjami między zmiennymi). Zaprezentowany został algorytm SHADE-LM, w którym poszczególne osobniki w populacji mogę zmieniać swoje położenie zgodnie z regułami zdefiniowanymi przez SHADE lub kierować się do obiecującego punktu wyznaczonego przez metamodel.

5. Zaborski, M., Mańdziuk, J. (2022): Improving LSHADE by means of a pre-screening mechanism [241]

W pracy szczegółowo przeanalizowaliśmy mechanizm preselekcji osobników wykorzystujący metamodel w połączeniu z algorytmem LSHADE (SHADE z liniową redukcją rozmiaru populacji), wprowadzając algorytm psLSHADE. Metamodel jest modelem kwadratowym z interakcjami, który dodatkowo uwzględnia odwrotne przekształcenia zmiennej ($\frac{1}{x}$ oraz $\frac{1}{x^2}$). Zaprezentowaliśmy trzy dynamiczne miary jakości metamodelu, z czego jedna nie wymaga narzutu w postaci dodatkowych ewaluacji funkcji celu. Eksperymentalnie pokazaliśmy dla jakich budżetów optymalizacji użycie metamodelu poprawia zbieżność algorytmu.

 M.Okulewicz, M.Zaborski, J.Mańdziuk (2022), Generalized Self-Adapting Particle Swarm Optimization algorithm with archive of samples [165]

Praca w sposób szczegółowy opisuje algorytm M-GAPSO oraz jego komponenty. Zbadaliśmy wpływ poszczególnych algorytmów (także metamodeli) na wydajność całego rozwiązania. Opisany został mechanizm restartów oraz mechanizm adaptacji reguł zachowań. Przedstawiliśmy analizę czasu działania w zależności od wykorzystanie poszczególnych reguł zachowań.

 M. Zaborski, M. Woźniak, J. Mańdziuk (2022), Multidimensional Red Fox metaheuristic for complex optimization [246]

W pracy sprawdziliśmy wydajność nowego, inspirowanego zachowaniem lisów, algorytmu RFO wykorzystując popularny zbiór testowy COCO BBOB. Następnie zaproponowaliśmy ulepszoną wersję algorytmu (MRFO) ze zmodyfikowaną fazą lokalnego przeszukiwania oraz reprodukcji. MRFO jest wyraźnie lepszy od bazowego RFO dla problemów o relatywnie dużej liczbie wymiarów. Co więcej, zaproponowany algorytm jest lepszy od popularnego DE (wariant DE-best) dla problemów o liczbie wymiarów 20 oraz 40.

 Zaborski, M., Mańdziuk, J. (2022): Surrogate-assisted LSHADE algorithm utilizing Recursive Least Squares filter [243]

W pracy pokazaliśmy, że współczynniki metamodelu mogą być estymowane w sposób rekurencyjny. Rozwinęliśmy koncepcję rozszerzenia algorytmu LSHADE o mechanizm preselekcji, wprowadzając algorytm rmmLSHADE. Zastosowaliśmy metamodel kwadratowy wraz z interakcjami. Wykorzystaliśmy filtr RLS, który po każdej ewaluacji funkcji celu aktualizuje współczynniki metamodelu. Sprawdziliśmy wpływ współczynnika zapominania na wydajność algorytmu. Zaproponowany algorytm rmmLSHADE nie posiada archiwum próbek. Eksperymentalna ewaluacja pokazała, że dla semikosztownych budżetów rmmLSHADE jest lepszy od LSHA-DE oraz psLSHADE, który estymuje współczynniki metamodelu z wykorzystaniem pełnego zbioru uczącego.

Prace w druku:

Zaborski, M., Mańdziuk, J. (2022): LQ-R-SHADE: R-SHADE with quadratic surrogate model [242]

Pokazaliśmy, że metamodel może służyć do preselekcji osobników i wyboru tych najbardziej obiecujących celem ich późniejszej ewaluacji. Zaprezentowaliśmy algorytm LQ-R-SHADE, będącym rozszerzeniem wydajnego algorytmu SHADE o kaskadę metamodeli (najbardziej złożony to model kwadratowy z interakcjami) oraz mechanizm restartów. Preselekcja osobników odbywa się w sposób transparentny dla mechanizmów adaptacji parametrów. Dodatkowo zastosowaliśmy zmodyfikowany mechanizm inicjalizacji początkowej populacji wykorzystujący metamodel.

1.5 Układ rozprawy

W **rozdziale 2** przedstawiono ogólny opis problemu optymalizacji oraz dokonano klasyfikacji problemów optymalizacji ze względu na ich właściwości. Zdefiniowano problem globalnej optymalizacji ciągłej z ograniczoną przestrzenią przeszukiwań, który jest rozważany w rozprawie. Opisano założenia czarnoskrzynkowości, które są kluczowe dla analizowanych problemów oraz proponowanych metod rozwiązujących. Następnie przedstawiono metody przeprowadzania eksperymentalnej ewaluacji metod rozwiązujących, które są wykorzystywane w części eksperymentalnej rozprawy. Poruszono także kwestię kosztu optymalizacji oraz jego wpływu na dostępny budżet ewaluacji, rozumiany jako dostępna liczba ewaluacji f.c. Wprowadzono pojęcie optymalizacji semikosztownej.

W **rozdziale 3** opisano wiodące metody rozwiązywania problemów ciągłych oraz przedstawiono ich systematykę. Szczegółowo opisano dwie grupy algorytmów. Pierwszą grupę stanowią algorytmy oparte o adaptacyjną ewolucję różnicową. Opisano bazowy algorytm *ewolucji różnicowej* (DE) oraz jego rozszerzenia wprowadzające mechanizmy adaptacji, takie jak SHADE, R-SHADE oraz L-SHADE. Drugą grupą są algorytmy oparte o mechanizm adaptacji macierzy kowariancji. Opisano bazowy algorytm CMA-ES oraz odniesiono się do znanych jego modyfikacji.

W rozdziale 4 zaprezentowano wiodące metody modelowania funkcji celu. Wprowadzono pojęcie metamodelu oraz zbioru uczącego, który jest przez niego wykorzystywany. Opisano popularne grupy metamodeli, takie jak *regresje wielomianowe, Kriging, sztuczne sieci neuronowe* oraz *radialne funkcje bazowe*. Poddano pod dyskusję wydajność procesu uczenia przedstawionych metamodeli, ze szczególnym uwzględnieniem złożoności obliczeniowej \mathcal{O} . Następnie opisano strategię *optymalizacji bayesowskiej*, która jest wiodącą metodą rozwiązującą problemy kosztowne. Przedstawiono metody integracji metamodeli w algorytmach populacyjnych. Opisano wybrane algorytmy populacyjne wspomagane metamodelem. Więcej uwagi poświęcono znanym algorytmom z rodziny DE oraz CMA-ES, które są wspierane metamodelem.

W rozdziale 5 omówiono problem efektywnego zastosowania metamodelu w algorytmie populacyjnym. Przedstawiono szczegółowe kryteria zastosowania metamodelu, które pozwalają dane zastosowanie uznać za efektywne. Zaprezentowano oraz omówiono eksperymentalne badanie czasu estymacji parametrów metamodeli takich jak: Kriging, metamodel oparty o radialną funkcję bazową, kwadratowa regresja wielomianowa oraz kwadratowa regresja wielomianowa z interakcjami. Następnie przeprowadzono dyskusję efektywności znanych metod integracji metamodeli, odnosząc się do wprowadzonych kryteriów efektywności oraz wyników eksperymentalnego badania czasu estymacji parametrów metamodeli. Finalnie, zaproponowano efektywne zastosowania metamodeli takie jak: inicjalizacja metamodelem, lokalna optymalizacja metamodelem, preselekcja rozwiązań na podstawie wartości metamodelu oraz rekurencyjna estymacja parametrów metamodelu. W rozdziale 6 przedstawiono zastosowanie lokalnej optymalizacji metamodelem w wybranych algorytmach populacyjnych. Najpierw opisano środowisko GAPSO, które umożliwia integrację wielu algorytmów populacyjnych w obrębie jednej populacji. Następnie przedstawiono koncepcję integracji lokalnej optymalizacji metamodelem w algorytmie GAPSO, czego wynikiem jest nowy algorytm M-GAPSO. W oparciu o algorytm M-GAPSO stworzono algorytm SHADE-LM, który został opisany w drugiej części tego rozdziału. SHADE-LM jest połączeniem R-SHADE oraz lokalnej optymalizacji metamodelem. Zarówno M-GAPSO, jak i SHADE-LM zostały poddane eksperymentalnej ewaluacji, w której oceniono zysk z integracji lokalnej optymalizacji metamodelem. Na koniec przedstawiono analizę wpływu użycia metamodeli na całkowity czas obliczeń M-GAPSO oraz SHADE-LM.

W rozdziale 7 przedstawiono zastosowanie preselekcji rozwiązań na podstawie wartości metamodelu w adaptacyjnych algorytmach R-SHADE oraz L-SHADE. Skutkiem tego, otrzymano następujące trzy algorytmy: LQ-R-SHADE, psLSHADE oraz rmmLSHADE. Wszystkie trzy zaprezentowane algorytmy za metamodel przyjmują regresję wielomianową lub jej rozszerzenie. LQ-R-SHADE oraz rmmLSHADE zakładają użycie kwadratowej regresji wielomianowej z interakcjami. Algorytm psLSHADE dodatkowo wykorzystuje nielinowe transformacje zmiennej objaśniającej w postaci $\frac{1}{x}$ oraz $\frac{1}{x^2}$. LQ-R-SHADE oprócz preselekcji rozwiązań wykorzystuje mechanizm inicjalizacji metamodelem. Wyróżnikiem rmmLSHADE jest estymacja parametrów metamodelu po każdej ewaluacji f.c. z wykorzystaniem rekursywnego filtru najmniejszych kwadratów. Wszystkie zaprezentowane algorytmy populacyjne wspomagane metamodelem zostały poddane eksperymentalnej ewaluacji. Zaprezentowano oraz przedyskutowano wpływ preselekcji rozwiązań na podstawie wartości metamodelu na czas obliczeń algorytmu.

W **rozdziale 8** podsumowano wyniki prac badawczych omówionych w rozprawie, przeprowadzając stosowną dyskusję efektywności proponowanych rozwiązań. Odniesiono się do postawionych hipotez badawczych oraz wskazano kierunki dalszych prac badawczych. Na koniec przedstawiono wkład autora w dziedzinę optymalizacji z wykorzystaniem metamodeli.

Rozdział 2

Wprowadzenie do problemów optymalizacji

W tym rozdziale przedstawiono ogólny opis problemu optymalizacji oraz dokonano klasyfikacji problemów optymalizacji ze względu na ich właściwości. Zdefiniowano problem globalnej optymalizacji ciągłej z ograniczoną przestrzenią przeszukiwań. Opisano założenia optymalizacji czarnoskrzynkowej, które są kluczowe dla analizowanych problemów oraz proponowanych metod rozwiązujących.

Następnie przedstawiono metody przeprowadzania eksperymentalnej ewaluacji metod rozwiązujących, które są wykorzystywane w części eksperymentalnej rozprawy. Poruszono także kwestię kosztu optymalizacji oraz jego wpływu na dostępny budżet ewaluacji, rozumiany jako dostępna liczba ewaluacji f.c. Wprowadzono pojęcie optymalizacji semikosztownej.

2.1 Problem optymalizacji

Problem optymalizacji jest problemem obliczeniowym polegającym na znalezieniu najlepszego rozwiązania spośród możliwych rozwiązań zgodnie z przyjętą funkcją celu. Funkcja celu (f.c.) jest inaczej nazywana funkcją jakości rozwiązania, funkcją przystosowania lub funkcją kosztu [163]. Ogólny problem optymalizacji [89] może być przestawiony w następujący sposób:

minimalizuj (albo maksymalizuj)
$$f(\boldsymbol{x})$$

ze względu na $\boldsymbol{x} \in S$ (2.1)

gdzie \boldsymbol{x} jest D-elementowym wektorem reprezentującym rozwiązanie, S jest przestrzenią przeszukiwań, f jest funkcją celu, a D jest liczbą wymiarów problemu.

Niezależnie od sposobu realizacji, uzyskanie wartości f.c. f w zadanym punkcie \boldsymbol{x} jest określane ewaluacją f.c. Dostępna (maksymalna) liczba ewaluacji f.c. jest określana budżetem optymalizacji i stanowi wiodące kryterium planowania oraz przerywania procesu optymalizacji. Maksymalizacja $f(\boldsymbol{x})$ jest zadaniem tożsamym minimalizacji $g(\boldsymbol{x}) = -f(\boldsymbol{x})$. Dlatego też, bez utraty ogólności w rozprawie przyjęto, że celem optymalizacji jest minimalizacja f.c. Najlepsze rozwiązanie, zwane inaczej rozwiązaniem optymalnym, jest oznaczane jako \boldsymbol{x}^* . Przyjmując za cel minimalizację $f(\boldsymbol{x})$, problem optymalizacji sprowadza się do znalezienia takiego punktu \boldsymbol{x}^* w przestrzeni przeszukiwań S, że zachodzi:

$$\boldsymbol{x}^* = \operatorname*{arg\,min}_{\boldsymbol{x} \in S} f(\boldsymbol{x}) \tag{2.2}$$

Zaprezentowana powyżej ogólna definicja ma zastosowanie do wszystkich problemów optymalizacji. Rozprawa skupia się na problemie globalnej optymalizacji ciągłej czarnoskrzynkowej z jednym celem, gdzie jedynym ograniczaniem jest organicznie przestrzeni przeszukiwań do *D*-wymiarowego prostopadłościanu. W rozdziale 2.2 zaprezentowano systematykę i opis wybranych problemów optymalizacji, ze szczególnym uwzględnieniem definicji optymalizacji ciągłej, optymalizacji globalnej oraz optymalizacji jednego celu. Wyjaśniono także, czym jest ograniczenie przestrzeni rozwiązań. Rozdział 2.3 definiuje czym jest optymalizacja czarnoskrzynkowa.

2.2 Systematyka problemów optymalizacji

Celem tego rozdziału jest przedstawienie wysokopoziomowej systematyki problemów optymalizacji z jednoczesnym zdefiniowaniem problemu optymalizacji, którego dotyczy rozprawa. Istnieje wiele pozycji literaturowych (np. [119, 89, 105, 173, 162]), które szczegółowo definiują problemy optymalizacji oraz towarzyszące im pojęcia. Z drugiej strony, wiele prac poświęconych konkretnym problemom lub metodom rozwiązującym (np. [130, 81, 96]) prezentuje własne systematyki problemów optymalizacji.

Rysunek 2.1 przedstawia proponowany przez autora rozprawy podział problemów optymalizacji, ze względu na pięć właściwości (A-E). Kolorem niebieskim zaznaczono właściwości stanowiące przedmiot zainteresowania rozprawy.

Właściwość (A) definiuje przestrzeń przeszukiwań S. Optymalizacja ciągła, to taka w której rozwiązanie \boldsymbol{x} składa się z D liczb rzeczywistych, tzn. $S \in \mathbb{R}^D$, gdzie D jest liczbą



Rysunek 2.1: Wysokopoziomowa klasyfikacja wybranych problemów optymalizacji. Kolorem niebieskim zaznaczono właściwości stanowiące przedmiot zainteresowania rozprawy.

wymiarów problemu.

Właściwość (B) ma związek z celem zadania optymalizacji. W przypadku optymalizacji globalnej za cel optymalizacji przyjmuje się znalezienie rozwiązania globalnie optymalnego, tzn. takiego które jest najlepsze w całej przestrzeni przeszukiwań S.

Właściwość (C) określa liczbę celów optymalizacji. W przypadku, gdy istnieje jedna f.c. i jest ona jednowartościowa, to optymalizacja ma jeden cel (ang. *single-objective optimization*).

Właściwość (D) dotyczy możliwości różniczkowania f.c. Brak takiej możliwości sprawia, że optymalizacja określana jest jako bez użycia pochodnych (ang. *derivative free optimization*). W literaturze optymalizację bez użycia pochodnych określa się często jako optymalizację czarnoskrzynkową [41, 116] (ang. *black-box optimization*). Jednakże, w rozprawie przyjęto, że brak możliwości obliczenia i wykorzystania pochodnych jest warunkiem koniecznym do określenia optymalizacji mianem czarnej skrzynki, lecz nie jest to warunek wystarczający.

Właściwość (E) jest związana z występowaniem ograniczeń. Ograniczenia są rozumiane jako warunki, które muszą zostać spełnione przez rozwiązanie \boldsymbol{x} , by było one akceptowalne. Możliwe ograniczenia oraz sposoby uwzględniania ich w metodach rozwiązujących są opisane m.in. w pracach [189, 88, 85]. W rozprawie skupiono się na optymalizacji z ograniczoną przestrzenią rozwiązań (ang. bound constrained optimization) do D-wymiarowego

prostopadłościanu.

Podsumowując, w rozprawie skupiono się na optymalizacji ciągłej, tzn. takiej w której $S \in \mathbb{R}^{D}$. Celem procesu optymalizacji jest znalezienie minimum globalnego \boldsymbol{x}^{*} . Istnieje jedna f.c. $f : \mathbb{R}^{D} \to \mathbb{R}$, która podlega minimalizacji. Postać f.c. jest nieznana, więc nie jest możliwe jej różniczkowanie. Jedynym przyjętym ograniczeniem jest ograniczenie przestrzeni przeszukiwań do *D*-wymiarowego prostopadłościanu.

2.3 Optymalizacja czarnoskrzynkowa

Optymalizacja czarnoskrzynkowa [12] zakłada, że niedostępna jest żadna wiedza o postaci funkcji celu oraz jej właściwościach (np. ciągłość, różniczkowalność, gładkość itp.). Nie jest także dostępna wiedza o oczekiwanych wartościach f.c., a w szczególności o przybliżonej wartości optymalnej. Jedyny sposób interakcji z f.c. odbywa się poprzez dokonanie jej ewaluacji, a tym samym poznanie jej wartości w danym punkcie (rysunek 2.2). Dlatego też, brak możliwości wykorzystania pochodnych jest składową optymalizacji czarnoskrzynkowej, lecz jej nie definiuje.

Należy podkreślić, że ewaluacja f.c. może być realizowana na wiele sposobów. W ogólności, nie musi być realizowana wyłącznie za pomocą obliczeń komputerowych. Przykładowo, czarna skrzynka może być eksperymentem chemicznym lub wiązać się z koniecznością podjęcia szeroko rozumianych ludzkich działań.



Rysunek 2.2: Ilustracja optymalizacji czarnoskrzynkowej.

W praktyce problemy nie są w pełni czarnoskrzynkowe, tzn. trudno przyjąć założenie o braku jakiejkolwiek intuicji dotyczącej kształtu f.c. bądź jej wartości. Niemniej, założenie o procesie optymalizacji realizowanym w sposób czarnoskrzynkowy pozwala na konstrukcję uniwersalnych metod rozwiązujących szeroką klasę problemów. Co więcej, podejście czarnoskrzynkowe nie wyklucza strojenia parametrów metody rozwiązującej bazując na doświadczeniu z wcześniejszych procesów optymalizacji.

2.4 Ewaluacja metod rozwiązujących

Ewaluacja metod rozwiązujących czarnoskrzynkowe problemy optymalizacji ciągłej jest oparta o wyznaczenie relacji pomiędzy jakością znajdowanego rozwiązania a zasobami, jakie były do tego potrzebne. W dalszej części tego rozdziału zaprezentowano wiodące sposoby mierzenia jakości uzyskiwanych rozwiązań oraz zużywanych zasobów.

Ewaluacji metody rozwiązującej można dokonać używając wybranych problemów optymalizacyjnych albo wykorzystując przygotowany do tego celu zbiór testowy wraz z wytycznymi dotyczącymi procedur ewaluacji, który często określany jest mianem *benchmarku*. Ewaluować można pojedynczą metodę rozwiązującą lub wiele metod jednocześnie, celem ich porównania. Analizując prace poświęcone metodom rozwiązującym problemy czarnoskrzynkowej optymalizacji ciągłej dominujące metody ewaluacji zakładają wyko-rzystanie:

- 1. Wybranych problemy inżynieryjnych.
- 2. Wybranych funkcji lub zbiorów funkcji.
- 3. Zbiorów testowych z grupy CEC, publikowanych na potrzeby Special Session and Competition on Single Objective Bound Constrained Numerical Optimization będącej elementem konferencji IEEE Congress on Evolutionary Computation (CEC).
- 4. Zbiorów testowych COCO [74] (ang. COmparing Continuous Optimizers), używanych m.in. na konferencjach Genetic and Evolutionary Computation Conference (GECCO) na warsztatach z benchmarkowania optymalizacji czarnoskrzynkowej (ang. Black-Box Optimization Benchmarking), nazywanych skrótowo BBOB.

2.4.1 Jakość rozwiązania

W przypadku optymalizacji jednego celu wynikiem działania dowolnej metody rozwiązującej jest znalezienie jednego rozwiązania $\boldsymbol{x}^{\varepsilon}$, któremu towarzyszy błąd ε . Błąd ε jest zdefiniowany jako różnica między wartością f.c. dla znalezionego rozwiązania $\boldsymbol{x}^{\varepsilon}$, a wartością f.c. dla rozwiązania optymalnego \boldsymbol{x}^* :

$$\varepsilon = f(\boldsymbol{x}^{\varepsilon}) - f(\boldsymbol{x}^{*}) \tag{2.3}$$

Błąd ε stanowi podstawową miarę jakości uzyskanego rozwiązania. W przypadku gdy ewaluacja metody rozwiązującej zakłada wielokrotne (R powtórzeń) uruchomienie procesu optymalizacji dla jednego lub wielu różnych problemów, to wynikiem działania metody jest zbiór błędów $\mathcal{E} = \varepsilon_1, \ldots, \varepsilon_R$. Zbiór błędów \mathcal{E} może być opisany za pomocą następujących przekształceń stanowiących miary jakości uzyskiwanych rozwiązań:

- 1. Wartość średnia błędów $\varepsilon_i \in \mathcal{E}$.
- 2. Mediania błędów $\varepsilon_i \in \mathcal{E}$.
- 3. Wartość minimalna błędów $\varepsilon_i \in \mathcal{E}$.
- 4. Wartość maksymalna błędów $\varepsilon_i \in \mathcal{E}$.
- 5. Wariancja błędów $\varepsilon_i \in \mathcal{E}$.
- 6. Odchylenie standardowe błędów $\varepsilon_i \in \mathcal{E}$.

W przypadku ewaluacji metody rozwiązującej z wykorzystaniem wielu różnych f.c. stosuje się przekształcenia błędu ε_i służące jego relatywizacji, w szczególności względem wartości $f(\boldsymbol{x}^*)$. Stosowane przekształcenia są właściwe dla poszczególnych zbiorów testowych.

2.4.2 Wykorzystywane zasoby

Podczas procesu optymalizacji metoda rozwiązująca wykorzystuje pewne zasoby, które są bezpośrednio odnoszone do jakości uzyskanego rozwiązania. Wiodącą miarą zużywanych zasobów jest liczba ewaluacji funkcji celu, która może być wykorzystywana na dwa sposoby:

- 1. Scenariusz ustalonego kosztu (ang. *fixed cost scenario*) mierzona jest jakość rozwiązania uzyskanego po wykonaniu określonej z góry liczby ewaluacji f.c.
- 2. Scenariusz ustalonego celu (ang. *fixed target scenario*) mierzona jest wykonana liczba ewaluacji f.c., do momentu uzyskania zadanego celu jakości rozwiązania (zazwyczaj określonej jako błąd ε).

Procedury ewaluacji metod rozwiązujących zazwyczaj zakładają stały budżet optymalizacji, zdefiniowany jako maksymalna dostępna liczba ewaluacji f.c. Tak skonstruowany eksperyment ewaluujący pozwala na realizację obu przedstawionych wyżej scenariuszy. Praktykowane jest mierzenie jakości otrzymanego rozwiązania w momencie wyczerpania budżetu optymalizacji oraz w dowolnych punktach czasu, rozumianych jako liczba ewaluacji f.c. wykonanych dotychczas, co odpowiada scenariuszowi ustalonego kosztu. Zebrane w trakcie eksperymentu wartości błędów po każdej ewaluacji f.c. pozwalają na realizację scenariusza ustalonego celu, przy założeniu, że nie wszystkie cele mogły zostać zrealizowane.

Spośród innych mierzonych zasobów zużywanych przez metodę rozwiązującą można wymienić:

- 1. Całkowity czas działania metody rozwiązującej uwzględniający czas potrzebny na ewaluacje f.c. W tym przypadku zakłada się pewien budżet optymalizacji i wyznacza całkowity czas procesu optymalizacji lub czas przypadający ja pojedynczą ewaluację f.c.
- Czas działania metody rozwiązującej z wyłączeniem czasu przeznaczonego na ewaluacje f.c.. Podobnie, taki pomiar realizuję się przy ustalonym budżecie ewaluacji, lecz dodatkowo mierzy się czas pojedynczej ewaluacji f.c.
- 3. Wymagania pamięciowe.

Wyżej wymienione sposoby mierzenia zasobów mają charakter eksperymentalny. Jednocześnie, prowadzone są badania nad teoretycznymi właściwościami metod rozwiązujących, spośród których należy wymienić analizę złożoności obliczeniowej \mathcal{O} [53].

W kontekście algorytmów populacyjnych złożoność obliczeniowa jest zależna od wymiarowości problemu D, liczby ewaluacji f.c. wykonanych dotychczas oraz parametryzacji algorytmu. W literaturze można znaleźć prace badające i porównujące złożoność obliczeniową różnych algorytmów populacyjnych [97, 213], lecz w ogólności jest to zadanie trudne [236]. Trudność polega na zmieniającej się parametryzacji w zależności od rozwiązywanego problemu oraz wysokim stopniu zaawansowania współczesnych algorytmów, dotyczącym w szczególności metod hybrydowych. Stąd, dominujące metody mierzenia wykorzystywanych zasobów przez współczesne algorytmy ewolucyjne zakładają podejście eksperymentalne.

Warto podkreślić, że badanie złożoności obliczeniowej \mathcal{O} znajduje zastosowanie w analizie metamodeli, które mogą stanowić rozszerzenie algorytmów populacyjnych (rozdział 4.3). W takim przypadku złożoność obliczeniowa metamodelu jest na ogół jest znacząco większa od złożoności obliczeniowej algorytmu populacyjnego, więc jej określenie pozwala oszacować skalowalność całej metody rozwiązującej.

2.4.3 Zbiór testowy CEC2021

Zbiory testowe z grupy CEC są publikowane na potrzeby specjalnej sesji konkursowej odbywającej się na corocznej konferencji *IEEE Congress on Evolutionary Computation* (CEC). Mimo różnic między ich poszczególnymi wersjami, idea pozostaje niezmienna. Zdefiniowany jest zbiór funkcji testowych oraz procedur walidacyjnych. Dodatkowo, zbiór testowy zawiera opis testowania wydajności ewaluowanego algorytmu. Zbiór funkcji oraz ich wymiarowość podlega modyfikacji wraz z każdą kolejną publikacją reportu opisującego nowy zbiór testowy. Procedura walidacyjna oraz metoda testowania wydajności pozostaje od kilku publikacji zbioru testowego niezmienna. Poniżej przedstawiono zwięzły opis zbioru testowego CEC2021 [152], który został wykorzystany do części badań eksperymentalnych przedstawionych w rozdziale 7.

Zbiór testowy CEC2021, opisany szerzej w technicznym raporcie [152], wykorzystuje 10 funkcji z ograniczoną przestrzenią przeszukiwań. Każda z funkcji należy do jednej z następujących czterech kategorii: funkcje unimodalne (f_1) , funkcje podstawowe $(f_2 - f_4)$, funkcje hybrydowe $(f_5 - f_7)$ oraz funkcje złożone $(f_8 - f_{10})$. Każda funkcja jest określona zarówno dla 10, jak i 20 wymiarów. Przestrzeń przeszukiwań jest stała i ograniczona do $[-100, 100]^D$. Szczegółowe definicje funkcji zostały zawarte w raporcie [152].

Dodatkowo, zaproponowano trzy transformacje funkcji: przesunięcie wartości funkcji (B), przesunięcie rozwiązania optymalnego (S) oraz obrót funkcji (R). Oprócz bazowych wersji funkcji, tj. beż żadnych transformacji, następujące cztery kombinacje transformacji są wykorzystywane: S, B+S, S+R, and B+S+R. Każdy z eksperymentów, rozumianych jako optymalizacja określonej funkcji z określoną wymiarowością oraz kombinacją transformacji, jest powtarzany 30-krotnie. Podsumowując, cały proces ewaluacji zakłada $10 \times 5 \times 30 \times 2 = 3000$ (funkcji × transformacji × powtórzeń × wymiarów) niezależnych przebiegów optymalizacji.

CEC2021 wykorzystuje scenariusz ustalonego kosztu. Przyjęto budżet optymalizacji wynoszący $2 \cdot 10^5$ ewaluacji f.c. dla problemów 10-wymiarowych oraz 10^6 ewaluacji f.c. dla problemów 20-wymiarowych. Ocenie podlega jakość najlepszego rozwiązania $\boldsymbol{x}^{\epsilon}$ znalezionego do momentu wyczerpania budżetu optymalizacji.

Obliczenie finalnej miary jakości *Score*, zdefiniowanej przez zbiór testowy CEC2021, wymaga wykonania kilku kroków. W pierwszej kolejności wyznacza jest miara *SNE*:

$$SNE = 0.5 \sum_{m=1}^{5} \sum_{i=1}^{10} ne_{i,m}^{10D} + 0.5 \sum_{m=1}^{5} \sum_{i=1}^{10} ne_{i,m}^{20D}$$
(2.4)

gdzie $ne_{i,m}^{dim}$ jest znormalizowaną wartością ostatecznego błędu (równanie 2.5) dla funkcji f_i , transformacji m oraz wymiaru dim (dla czytelności niektóre indeksy są pominięte):

$$ne = \frac{f(\boldsymbol{x}_{best}) - f(\boldsymbol{x}^*)}{f(\boldsymbol{x}_{best})_{max} - f(\boldsymbol{x}^*)}$$
(2.5)

gdzie $f(\mathbf{x}_{best})$ jest najlepszym wynikiem jaki osiągnął algorytm spośród 30 powtórzeń, $f(\mathbf{x}^*)$ jest optymalną wartością funkcji f_i , a $f(\mathbf{x}_{best})_{max}$ jest największą (najgorszą) wartością funkcji $f(\mathbf{x}_{best})$ spośród wszystkich algorytmów biorących udział w porównaniu (konkursie).

Następnie miara Score1 jest wyznaczana w następujący sposób:

$$Score1 = \left(1 - \frac{SNE - SNE_{min}}{SNE}\right) \times 50 \tag{2.6}$$

gdzie SNE_{min} jest najmniejszą wartością SNE spośród wartości uzyskanych przez wszystkie algorytmy biorące udział w porównaniu. Następnie miara SR jest obliczana zgodnie z poniższym równaniem:

$$SR = 0.5 \sum_{m=1}^{5} \sum_{i=1}^{10} rank_{i,m}^{10D} + 0.5 \sum_{m=1}^{5} \sum_{i=1}^{10} rank_{i,m}^{20D}$$
(2.7)

gdzie $rank_{i,m}^{dim}$ jest rangą algorytmu liczoną na podstawie wyników wszystkich algorytmów dla funkcji f_i , transformacji m i wymiaru dim. Poprzez wynik rozumie się średni błąd spośród uzyskanych 30 błędów (30 powtórzeń eksperymentu). Dalej, miara *Score2* jest wyznaczana w następujący sposób:

$$Score2 = \left(1 - \frac{SR - SR_{min}}{SR}\right) \times 50 \tag{2.8}$$

gdzie SR_{min} jest najmniejszą wartością SR spośród wartości uzyskanych przez wszystkie algorytmy biorących udział w porównaniu.

Finalna miara jakości *Score* jest sumą miary *Score*1 (równanie 2.6) oraz miary *Score*2 (równanie 2.8):

$$Score = Score1 + Score2 \tag{2.9}$$

Empiryczna złożoność obliczeniowa

CEC2021 dostarcza procedurę badania empirycznej złożoności obliczeniowej algorytmu [152]. Procedura składa się z następujących kroków:

- 1. Uruchomienia testowego programu (pseudokod 1) i zapisania czasu wykonania jako $T_0. \label{eq:transform}$
- 2. Zmierzenia czasu (T_1) 200 000 ewaluacji funkcji f_1 w wymiarze dim.
- 3. Zmierzenia całkowitego czasu (T_2) działania algorytmu, podczas którego dokonane zostało 200 000 ewaluacji funkcji f_1 w wymiarze dim.
- 4. Wykonanie kroku 3. pięć razy, zapisanie pięciu wartości T_2 i wyznaczanie przekształconego T_2 jako ich średniej arytmetycznej.

Kroki 2., 3. oraz 4. są wykonywane niezależnie dla każdego dostępnego wymiaru $dim \in \{10, 20\}$. Finalna empiryczna złożoność obliczeniowa T_3 jest również wyznaczana dla każdego dostępnego wymiaru dim w następujący sposób¹:

$$T_3 = \frac{T_2 - T_1}{T_0} \tag{2.10}$$

Algorytm 1 Program testowy do badania empirycznej złożoności obliczeniowej, zdefiniowany w CEC2021 [152]

```
1: x = 0.55
 2: for i = 1 to 200000 do
 3:
       x = x + x
 4:
       x = x/2
 5:
       x = x * x
       x = sqrt(x)
 6:
 7:
       x = log(x)
 8:
       x = exp(x)
 9:
       x = x/(x+2)
10: end for
```

2.4.4 Zbiór testowy COCO

Zbiór testowy COCO [74] jest wykorzystywany m.in. na warsztatach konferencji GEC-CO i pozostaje w przeważającej części niezmienny od wielu lat. COCO definiuje kilka platform testowych oraz towarzyszących im procedur eksperymentalnej walidacji. Poniżej opisano podstawową platformę przeznaczoną dla czarnoskrzynkowej optymalizacji ciągłej

¹Oryginalny opis CEC2021 nie wprowadza miary T_3 , a jedynie definiuje ją jako $(T_2 - T_1)/T_0$. W rozprawie stosuje się dla czytelności oznaczenie T_3 .

wykorzystującą funkcje bezszumowe (ang. *noiseless functions*), która została wykorzystana w części badań eksperymentalnych przedstawionych w rozdziale 6 oraz rozdziale 7. Opis dotyczy obowiązującej wersji (v. 2.4).

COCO wykorzystuje 24 funkcje, które są szczegółowo opisane w [59]. Każdej z funkcji przyporządkowano jedną z następujących pięciu kategorii: funkcje separowalne $(f_1 - f_5)$, funkcje ze słabym lub średnim uwarunkowaniem $(f_6 - f_9)$, funkcje z dużym uwarunkowaniem i jednomodalne $(f_{10} - f_{14})$, funkcje wielomodalne z wyraźną strukturą globalną $(f_{15} - f_{19})$ oraz funkcje wielomodalne ze słabą strukturą globalną $(f_{20} - f_{24})$. Każda funkcja jest zdefiniowana dla 2, 3, 5, 10, 20 oraz 40 wymiarów. Każda z funkcji występuje w 15 instancjach, różniących się m.in. położeniem rozwiązania optymalnego oraz wartością optymalną. Przestrzeń przeszukiwań jest stała dla wszystkich problemów i wynosi $[-5,5]^D$.

COCO wykorzystuje scenariusz ustalonego celu. Wydajność algorytmu jest mierzona w oparciu o oczekiwany czas działania (ang. *Expected Running Time*), nazywany dalej ERT. Poprzez czas rozumie się liczbę ewaluacji f.c., jaka jest potrzebna, by osiągnąć założony cel. Cel f^{cel} jest określony jako przedział wartości f.c., które są oddalone od wartości optymalnej $f^{opt} = f(\boldsymbol{x}^*)$ o nie więcej niż pewna przyjęta wartość Δf . Dla domyślnego scenariusza wykorzystania zbioru testowego COCO, najbardziej wymagający cel f^{cel} jest określony dla $\Delta f = 10^{-8}$. Całość jest zaprezentowana za pomocą poniższego równania:

$$f^{cel} = f^{opt} \pm \Delta f \tag{2.11}$$

gdzie $\Delta f \in \mathbb{R}$ bezpośrednio określa trudność założonego celu.

COCO dostarcza pełne środowisko do walidacji algorytmów, które generuje tabele i wykresy prezentujące wyniki w różnych agregatach (wymiary, funkcje, klasy funkcji). Wiodąca i powszechnie stosowana w literaturze forma prezentacji wyników COCO wykorzystuje wykres funkcji skumulowanego empirycznego rozkładu (ECDF) wartości ERT. Przykładowy wykres ECDF [74] jest przedstawiony na rysunku 2.3. Wykres zawiera porównanie siedmiu algorytmów optymalizujących funkcję f_{10} w 20 wymiarach. Na osi xznajdują się wartości ERT (oznaczone jako #f-evals) poddane przekształceniu (podzieleniu przez liczbę wymiarów problemu, a następnie zlogarytmowaniu) celem lepszej prezentacji. Na osi y znajdują się wartości dystrybuanty wskazujące jaka cześć problemów została rozwiązana. Prezentowane wartości dystrybuanty są wartościami uśrednionymi, a rodzaj średniej jest uzależniony od wybranego agregatu. Na przytoczonym rysunku 2.3 dystrybuanta jest średnią z 15 przebiegów funkcji pochodzących od 15 instancji funkcji $f_{10} \le 20$ wymiarach.

Dodatkowo, na wykresie zaznaczono linię *best 2009*, która jest pochodną najmniejszych wartości ERT jakie były wymagane by osiągnąć zadany cel podczas podczas konkursu BBOB 2009. Poszczególne wartości ERT pochodzą w ogólności od dowolnego algorytmu ewaluowanego podczas BBOB 2009. Stąd, jest to syntetyczny przebieg stanowiący punkt odniesienia dla nowo ewaluowanych algorytmów.

Podsumowując, główna metoda porównawcza dostarczana przez COCO jest oparta o wykres pokazujący jaka część założonych celów została (średnio) zrealizowana przy określonej liczbie ewaluacji f.c.



Rysunek 2.3: Przykład wykresu porównawczego, wygenerowanego przez COCO, dla funkcji f_{10} w 20 wymiarach w domyślnej konfiguracji. Źródło: [74].

COCO, wykorzystując scenariusz ustalonego celu, nie definiuje wprost budżetu optymalizacji. Możliwa jest ewaluacja algorytmu zakładająca przerwanie przebiegu optymalizacji po dowolnej liczbie ewaluacji f.c. Na wykresie (rysunek 2.3) oznaczane jest to krzyżykiem, dalszy przebieg dystrybuanty ma charakter przybliżony (COCO dokonuje symulacji realizowania następnych celów, o czym można szerzej przeczytać w [74]). Jednakże, domyślne wykresy posiadają zakres osi x od 0 do ponad $10^6 \cdot D$ ERT. Stąd, zazwyczaj przyjmuje się budżety bliskie zakresowi osi x ($10^5 \cdot D - 10^6 \cdot D$), by zachować wysoką jakość porównań (rzeczywistą, a nie podlegającą symulacji).

Zbiór testowy COCO może być używany w scenariuszu kosztownym (opcja –expensive), czego skutkiem jest ograniczenie zakresu osi x do #f-evals = $10^3 \cdot D$. Zmodyfikowany też jest zbiór wykorzystywanych celów. Przykład wykresu porównawczego wygenerowanego w scenariuszu kosztownym zaprezentowano na rysunku 2.4.


Rysunek 2.4: Przykład wykresu porównawczego, wygenerowanego przez COCO, dla funkcji f_{10} w 20 wymiarach w scenariuszu kosztownym. Źródło: [74].

2.4.5 Budżety optymalizacji w popularnych zbiorach testowych

W tabeli 2.1 przedstawiono zestawienie budżetów optymalizacji stosowanych w ww. zbiorach testowych w podziale na wymiarowości problemów (D). Uwzględniono możliwości użycia scenariusza kosztownego w COCO. Dodatkowo, zaprezentowano budżety odpowiadające zbiorom testowym CEC2014 [128], CEC2017 [15] oraz CEC2015 [36], który jest dedykowany optymalizacji kosztownej. Nie wszystkie wykorzystywane wymiarowości problemów zostały zawarte w porównaniu. Podobnie, nie każdy zbiór testowy wykorzystuje wszystkie zaprezentowane wymiarowości. W przypadku gdy benchmark nie wykorzystuje problemów o danej liczbie wymiarów wartość budżetu oznaczono jako Ø. Zbiory testowe zakładające ograniczony budżet optymalizacji w związku z wysokim kosztem optymalizacji oznaczono za pomocą (k).

2.5 Koszt optymalizacji

Koszt ewaluacji funkcji celu jest rozumiany jako czas lub inna przyjęta miara wymagana do jej wykonania. W literaturze istnieje wiele definicji nakładów, które stanowią koszt optymalizacji. Część prac przyjmuje, że w pojedyncza ewaluacja f.c. jest obarczona dużym kosztem (jest kosztowna) i nie jest szerzej dyskutowane jaką miarę kosztu przyjęto [126, 197]. Często zakłada się, że koszt jest rozumiany jako relatywnie duży czas pojedynczej ewaluacji f.c. [126, 101]. Inna grupa prac wskazuje wprost, że chodzi o koszt obliczeniowy [21, 39, 22]. Można także znaleźć odwołania, że koszt może być rozumiany

Tabela 2.1: Budżety optymalizacji (rozumiane jako dostępna liczba ewaluacji f.c.) przyjęte w popularnych zbiorach testowych (Z.T.) w podziale na liczbę wymiarów problemu (D). W przypadku gdy benchmark nie wykorzystuje problemów o danej liczebnie wymiarów wartość budżetu oznaczono jako Ø.

Wymiar Z.T.	D=2	D = 10	D = 20	D = 30	D = 40	D = 100
CEC2014 [128]	Ø	$10^4 \cdot D$	Ø	$10^4 \cdot D$	Ø	$10^4 \cdot D$
CEC2017 [15]	Ø	$10^4 \cdot D$	Ø	$10^4 \cdot D$	Ø	$10^4 \cdot D$
CEC2021 [152]	Ø	$2 \cdot 10^4 \cdot D$	$5\cdot 10^4\cdot D$	Ø	Ø	Ø
COCO [74]	$10^6 \cdot D$	$10^6 \cdot D$	$10^6 \cdot D$	Ø	$10^6 \cdot D$	Ø
CEC2015 (k) [36]	Ø	$50 \cdot D$	Ø	$50 \cdot D$	Ø	Ø
COCO(k) [74]	$10^3 \cdot D$	$10^3 \cdot D$	$10^6 \cdot D$	$10^3 \cdot D$	Ø	Ø

jako środki pieniężne [136] lub nakład pracy związany z pozyskaniem dużej ilości danych na potrzeby właściwej ewaluacji [103].

W kontekście optymalizacji czarnoskrzynkowej założenia dotyczące braku wiedzy o problemie pozostają w mocy bez względu na czas lub środki potrzebne na ewaluację f.c. Wewnętrzna struktura problemu nie może być uwzględniana w procesie projektowania oraz ewaluacji metod rozwiązujących.

2.5.1 Optymalizacja kosztowna

W literaturze wyróżnia się pojęcie optymalizacji kosztownej [198] (ang. expensive optimization), która zakłada, że koszt pojedynczej ewaluacji f.c. jest znaczący, co implikuje organicznie budżetu ewaluacji. Przykładami kosztownych problemów optymalizacji są eksperymenty fizyczne, w których czas pojedynczej ewaluacji f.c. może być liczony w godzinach. W literaturze można znaleźć przykład symulacji samochodowego testu zderzeniowego, którego przygotowanie wymaga 36-160 godzin [203]. Analizy chemiczne ropy naftowej, oprócz nakładów pieniężnych na pozyskanie próbki, zajmują kilkanaście godzin [43]. Co więcej, rozwój uczenia maszynowego sprawił, że finalne systemy (np. systemy rekomendacji, narzędzia analizy medycznej, silniki gier czasu rzeczywistego, czy algorytmy rozpoznawania mowy) posiadają wiele parametrów wymagających dostrojenia, co również jest zaliczane do optymalizacji kosztownej [197]. Patrząc na szerokie spektrum problemów kosztownych, czas między kilkoma minutami a wieloma godzinami wydaje się być racjonalnym określeniem kosztownej ewaluacji [185], jeśli za kryterium przyjmuje się czas.

W większości prac poświęconych metodom rozwiązującym problemy optymalizacji zagadnienie kosztu ewaluacji f.c. jest pomijane, co jest równoznaczne z założeniem, że optymalizowany problem nie jest kosztowny. Jednakże w pracach poświęconych problemom kosztownym używa się ogólnego określenia - tani (ang. *cheap*) [71, 126, 198], sugerującego, że dany sposób ewaluacji f.c. nie jest kosztowny. Stąd, optymalizacja zakładająca tanią ewaluację f.c. jest określana optymalizacją tanią [52].

Budżet optymalizacji kosztownej

W literaturze przyjmuje się różne wielkości budżetów, które uznaje się za właściwe dla optymalizacji kosztownej. W jednej z prac [248] zasugerowano, że budżet jest ograniczony do setek albo tysięcy ewaluacji f.c. Analizując prace prezentujące metody rozwiązujące problemy kosztowne, można znaleźć następujące wartości: poniżej 100 ewaluacji f.c. [104], 250 ewaluacji f.c. [115], czy 100 · D ewaluacji f.c [92], gdzie D jest liczbą wymiarów problemu. Podobne wartości przyjęto w konkursie optymalizacji czarnoskrzynkowej BBComp [137] dedykowanym optymalizacji kosztownej (10 · D - 100 · D ewaluacji f.c.). Zbiór testowy CEC2015 [36] przeznaczony dla optymalizacji kosztownej wykorzystuje budżet 50 · D (tabela 2.1). Scenariusz kosztowny zbioru testowego COCO [74] zakłada aż $10^3 \cdot D$ ewaluacji f.c. (tabela 2.1). Z kolei zbiory testowe przeznaczone do ewaluacji algorytmów w budżetach optymalizacji taniej przyjmują budżety rzędu $10^4 \cdot D - 10^6 \cdot D$ ewaluacji f.c. (tabela 2.1).

2.5.2 Optymalizacja semikosztowna

Zestawienie optymalizacji kosztownej pokazuje, że w literaturze nie istnieje konsensus jednoznacznie wyznaczający granicę między tymi rodzajami optymalizacji. Ma to swoje uzasadnienie w różnorodności problemów optymalizacji. Mimo to, z punktu widzenia optymalizacji czarnoskrzynkowej takie rozróżnienie jest pożądane. Rozróżnienia można dokonać w oparciu o dwa kryteria:

- 1. Koszt ewaluacji f.c.
- 2. Przyjęty budżet optymalizacji.



Rysunek 2.5: Podział optymalizacji ze względu dostępny budżet (liczbę ewaluacji f.c.).

Rozróżnienie w oparciu o koszt ewaluacji f.c. jest właściwe, jeśli za regułę przyjąć naturę problemów optymalizacji oraz towarzyszących im eksperymentów. Z kolei rozróżnienie w oparciu o przyjęty budżet ewaluacji jest wysoce użyteczne z punktu widzenia doboru metody rozwiązującej. Dlatego też, zdecydowano się dokonać podziału optymalizacji ze względu na wynikowy budżet optymalizacji.

Próba przyporządkowania zakresów budżetów do optymalizacji kosztownej oraz taniej uwidacznia znaczącą lukę między tymi dwoma typami optymalizacji. Nie jest jasne, gdzie należy postawić granicę. W rozumieniu optymalizacji kosztownej budżet wynoszący $10^3 \cdot D$ jest wartością nad wyraz dużą, nie znajdującą zastosowania dla metod rozwiązujących dedykowanych problemom kosztownym (rozdział 4.4). Analogicznie, optymalizacja tania zakładająca budżet $10^3 \cdot D$ charakteryzowałaby się relatywnie niewielkim budżetem w porównaniu do budżetów rzędu $10^6 \cdot D$.

Dlatego też, na potrzeby tej rozprawy optymalizacja, której budżet oscyluje pomiędzy optymalizacją kosztowną, a optymalizacją tanią, została nazwana optymalizacją semi-kosztowną. Wartości budżetu optymalizacji będące pomiędzy $10^2 \cdot D$ a $10^4 \cdot D$ wydają się być dla niej rozsądnym założeniem. Takie budżety ewaluacji mają szczególne znaczenie z punktu widzenia zastosowania efektywnych metamodeli, opisanych dalej w rozdziałe 5. Na rysunku 2.5 przedstawiono zaproponowany przez autora podział optymalizacji ze względu na dostępny budżet.

Rozdział 3

Wiodące metody rozwiązywania problemów optymalizacji ciągłej

W tym rozdziale opisano wiodące metody rozwiązywania problemów ciągłych oraz przedstawiono ich systematykę. Szczegółowo opisano dwie grupy algorytmów. Pierwszą grupę stanowią algorytmy o oparte o adaptacyjną ewolucję różnicową. Opisano bazowy algorytm *ewolucji różnicowej* (DE) oraz jego rozszerzenia wprowadzające mechanizmy adaptacji, takie jak SHADE, R-SHADE oraz L-SHADE. Drugą grupą są algorytmy oparte o mechanizm adaptacji macierzy kowariancji. Opisano bazowy algorytm CMA-ES oraz odniesiono się do znanych jego modyfikacji.

Obie przedstawione grupy algorytmów stanowią uniwersalne i wysoce wydajne metody rozwiązujące problemy czarnoskrzynkowej optymalizacji ciągłej. Algorytmy z rodziny DE stanowią podstawę do proponowanych rozwiązań, zaprezentowanych w rozdziale 6 oraz rozdziale 7.

3.1 Systematyka metod rozwiązujących

W literaturze występuje wiele systematyk metod rozwiązujących problemy optymalizacji. Istnieją systematyki patrzące na problem optymalizacji w sposób holistyczny i dzielące znane metody ze względu na wysokopoziomowy sposób działania [234, 200]. Część systematyk jest prezentowana w literaturze poświęconej problemom inżynieryjnym [6, 224, 96], co może skutkować nadreprezentacją metod przeznaczonych do rozwiązywania konkretnych problemów. Kolejną grupą systematyk są te wychodzące od wyodrębnienia pewnej rodziny algorytmów, a następnie dokonują podziału na jej podgrupy. Przykładem te-

ROZDZIAŁ 3. WIODĄCE METODY ROZWIĄZYWANIA PROBLEMÓW OPTYMALIZACJI CIĄGŁEJ

go mogą być systematyki metaheurystyk inspirowanych przyrodą [50], które mogą być przeznaczone zarówno dla optymalizacji ciągłej oraz dyskretnej. Niektóre systematyki są oparte o szczególne cechy algorytmów, takie jak zdolność do obliczeń rozproszonych [72]. Kolejne wyzwanie w opracowywaniu systematyk stanowi obecność metod hybrydowych, które mogą składać się z podejść opartych o istotnie różne mechanizmy. Szczególnym przykładem tego jest przedmiot tej rozprawy, czyli zastosowanie metamodeli, mających źródło optymalizacji bayesowskiej, w algorytmach populacyjnych. Istnieją prace, które dostrzegają problemy znanych systematyk i proponują nowe, bardziej adekwatne do współczesnych nurtów badawczych [211].

Czerpiąc ze znanych źródeł systematyk autor rozprawy proponuje własną systematykę. Przedstawiona na rysunku 3.1 klasyfikacja metod jest ukierunkowana na optymalizację ciągłą z jednym celem. Autor jest świadomy selektywności zaprezentowanych grup oraz rodzin algorytmów, której nie sposób uniknąć przy obecnym rozwoju nauki poświęconej metodom rozwiązującym problemy optymalizacji ciągłej. Niemniej przedstawiona systematyka pozwala określić, które grupy algorytmów są uwzględnione w rozprawie i jaka jest ich relacja względem innych metod. Systematyka nie zawiera metod hybrydowych, które mogą wykorzystywać algorytmy o istotnie różnej mechanice działania, a tym samym pochodzące z różnych grup wykorzystywanych w klasyfikacji.

Wszystkie zaprezentowane metody mają zastosowanie w optymalizacji ciągłej. Kolorem szarym zaznaczono metody znajdujące zastosowanie w optymalizacji spełniającej paradygmat czarnoskrzynkowości. Kolorem niebieskim zaznaczono adaptacyjne algorytmy wywodzące się z *ewolucji różnicowej* (DE) oraz algorytmy oparte o *strategię ewolucyjną wykorzystującą adaptację macierzy kowariancji* (CMA-ES). Obie grupy będą nazywane skrótowo algorytmami z rodziny DE oraz algorytmami z rodziny CMA-ES.

Algorytmy z rodziny DE oraz algorytmy z rodziny CMA-ES należą do metod numerycznych [180, 37]. Metoda numeryczna jest rozumiana jako zbiór procedur wykonywanych na liczbach prowadzących do uzyskania rozwiązania. Bez względu na rodzaj konkretnej metody rozwiązaniu towarzyszy błąd numeryczny wynikający z precyzji maszynowej procesora, na którym były wykonywane obliczenia. Metody numeryczne stosuje się w sytuacji, gdy nie jest możliwe znalezienie rozwiązania metodą analityczną. Algorytmy z rodziny DE oraz algorytmy z rodziny CMA-ES są klasyfikowane jako metody numeryczne niedeterministyczne.

Dalej, algorytmy z rodziny DE oraz algorytmy z rodziny CMA-ES uznaje się za metaheurystyki, czyli wysokopoziomowe heurystyki. Mianem heurystyki określa się specjali-



Rysunek 3.1: Wysokopoziomowa klasyfikacja wybranych metod rozwiązujących problemy optymalizacji ciągłej z jednym celem. Na szaro zaznaczono metody znajdują zastosowanie w optymalizacji spełniającej paradygmat czarnoskrzynkowości. Na niebiesko zaznaczono algorytmy z rodziny DE oraz algorytmy z rodziny CMA-ES.

ROZDZIAŁ 3. WIODĄCE METODY ROZWIĄZYWANIA PROBLEMÓW OPTYMALIZACJI CIĄGŁEJ

zowaną metodę znajdowania rozważań, które nie daje gwarancji znalezienia rozwiązania optymalnego, a jedynie przybliżone. Przedrostek *meta* oznacza, że heurystyka jest ogólną metodą rozwiązującą dowolny problem, co jest zgodne z założeniem czarnoskrzynkowości. Wzrost dostępnej mocy obliczeniowej sprawił, że metaheurystyki zyskują na znaczeniu, ponieważ przy odpowiednio dużej liczbie iteracji oczekuje się, że dokładność otrzymanego rozwiązania będzie akceptowalna. Motywacja do używania metaheurystyk wzrasta w sytuacji gdy optymalizowany problem jest na tyle złożony, że nie jest możliwe sprawdzenie wszystkich rozwiązań, a metody przeszukiwania losowego zbiegają zbyt wolno.

Metaheurystyki dzielą się na dwie grupy: algorytmy populacyjne oraz jednopunktowe. Algorytmy populacyjne zawierają pewną populację znajdującą się w określonych punktach przestrzeni rozumianych jako możliwe rozwiązania. Metody jednopunktowe opierają się o iteracyjne wyznaczanie pojedynczych punktów do ewaluacji.

Podział algorytmów populacyjnych na algorytmy ewolucyjne [225, 201] oraz algorytmy rojowe [34] nie jest jedyną uznawaną taksonomią. Istnieją prace [46] włączające algorytmy rojowe do szerokiej grupy algorytmów ewolucyjnych bez wprowadzania pojęcia algorytmu populacyjnego. Zdaniem autora rozprawy genezy oraz zasady działania obu grup są istotnie różne, więc zasadnym jest klasyfikowanie ich jako równorzędne grupy zawarte w zbiorze algorytmów populacyjnych. Stąd, algorytmy z rodziny DE oraz algorytmy z rodziny CMA-ES zostały zaklasyfikowane do grupy algorytmów ewolucyjnych, które należą do większej grupy algorytmów populacyjnych. Szczegółowy opis algorytmów z rodziny DE oraz algorytmów z rodziny CMA-ES znajduje się odpowiednio w rozdziale 3.3 oraz rozdziale 3.4.

Metody numeryczne deterministyczne stanowią alternatywę dla metod numerycznych niedeterministycznych. Determinizm algorytmu skutkuje tym, że rezultatem wielokrotnego uruchomienia tego samego algorytmu jest uzyskanie zbioru takich samych wyników. Co więcej, znalezione rozwiązanie jest dokładne, z zastrzeżeniem błędu wynikającego z ograniczonej precyzji maszynowej. Większość metod numerycznych deterministycznych nie znajduje zastosowania w optymalizacji czarnoskrzynkowej, ponieważ zakładają znaną postać funkcji, a w przypadku metod gradientowych również jej różniczkowalność. Niemniej, metody takie jak programowanie liniowe [93], programowanie nieliniowe [141], optymalizacja wypukła [28] oraz metody gradientowe [239, 190] mogą stanowić preferowaną technikę rozwiązującą w sytuacji, gdy problem nie jest czarnoskrzynkowy.

Istnieją metody numeryczne deterministyczne mające zastosowanie w optymalizacji czarnoskrzynkowej, czego przykładem jest metoda Neldera-Meada [204], która w iteracyj-

ny sposób wyznacza kolejne punkty do ewaluacji, co skutkuje sukcesywnym zawężaniem przestrzeni rozwiązań i dążeniem do optimum, w ogólności lokalnego.

Należy zaznaczyć, że metody numeryczne deterministyczne gwarantują znalezienie rozwiązania dokładnego i powtarzalnego, lecz może to nie być rozwiązanie optymalne globalnie. Przykładem jest optymalizacja problemów opisanych przez funkcje wielomodalne, tzn. takie w których występuje wiele ekstremów lokalnych. W takim przypadku znajdowane rozwiązanie może być optymalne lokalnie, lecz nie być optymalnym globalnie.

Umieszczenie *optymalizacji bayesowskiej* w metodach numerycznych niedeterministycznych wynika z uogólnienia ich mechanizmów działania, które w ogólności są niedeterministyczne, a jedynie w szczególnych przypadkach konstrukcja optymalizacji bayesowskiej ma charakter w pełni deterministyczny. Szerszy opis optymalizacji bayesowskiej jest zawarty w rozdziale 4.4.

3.2 Algorytmy populacyjne

Algorytmy populacyjne są metaheurystykami, które dokonują losowego przeszukiwania przestrzeni. Przeszukiwanie przestrzeni odbywa się poprzez dokonanie wielokrotnych, w obrębie jednej iteracji, ewaluacji funkcji celu. Wielokrotność ewaluacji wynika z faktu występowania populacji, w której jeden osobnik reprezentuje jedno rozwiązania. Dwa przeciwstawne cele, którymi jest motywowana każda metaheurystyka optymalizacyjna [234] to eksploracja oraz eksploatacja. Celem eksploracji jest kierowanie ruchem populacji w taki sposób, by zachować zdolność do globalnego przeszukiwania przestrzeni. Celem eksploatacji jest skupianie populacji w lokalnym obszarze ze względu na obiecujące rozwiązanie, które się w nim znajduje. Sam algorytm jest w tym kontekście rozumiany jako zbiór reguł, które starają się wypełniać oba cele jednocześnie, co jest problemem trudnym [42].

Istnieje twierdzenie, które mówi że *nie ma darmowych obiadów* [230], tzn. że znalezienie ogólnie najlepszej metaheurystyki nie jest możliwe. Mimo to, badania eksperymentalne dowodzą, że wybór metaheurstyki właściwej dla określonej klasy problemów pozwala na uzyskanie nieprzeciętnych wyników [113].

Za pierwowzór algorytmów ewolucyjnych, należących do metod populacyjnych uznaje się *algorytm genetyczny* (GA) [87] wprowadzony w 1975 roku. GA jest inspirowany teorią ewolucji Darwina [44] oraz procesem naturalnej selekcji znanej z systemów biologicznych. GA jest oparty o trzy fazy: mutację, krzyżowanie oraz selekcje. Mimo, że GA był początkowo zaprojektowany do optymalizacji dyskretnej, to większość współcześnie stosowanych algorytmów ewolucyjnych przeznaczonych do optymalizacji ciągłej jest nim w dużym stopniu inspirowanych.

Druga grupa algorytmów populacyjnych, tj. algorytmy rojowe ma swój początek w roku 1989, kiedy zaprezentowano algorytm *przeszukiwania metodą stochastycznej dyfuzji* (SDS) [25] (ang. *Stochastic Diffusion Search*). Jednakże, algorytmy rojowe zyskały znaczące zainteresowanie dopiero za sprawą zaprezentowanego w 1995 roku algorytmu *optymalizacji rojem cząstek* (PSO) [110] (ang. *Particle Swarm Optimizaion*). PSO jest oparte o rój cząstek, który wymieniając informację pomiędzy poszczególnymi cząstkami kieruje się do obiecujących regionów zachowując przy tym zdolność do eksploracji.

Na przestrzeni lat zaprezentowanych zostało wiele nowych algorytmów ewolucyjnych [225, 201] oraz rojowych [34]. Z punktu widzenia tematyki rozprawy kluczowe znaczenia mają dwie metaheurystyki.

Pierwszą z nich jest algorytm *ewolucji różnicowej* (DE) [212] (ang. *Differential Evolution*) zaprezentowany w 1997 roku, który jest z powodzeniem stosowany do dziś. Jest to relatywnie nieskomplikowana metaheurystyka, inspirowana GA, lecz bez nawiązań do przyrody. Uniwersalność DE sprawiła, że stanowi on podstawę istotnie bardziej zaawansowanych algorytmów [251, 215, 217, 26, 151]. Porównując bazowy DE oraz jego następców można dostrzec, że wzrost efektywności następców względem bazowego algorytmu wynika przede wszystkim z wykorzystania mechanizmu adaptacji parametrów. Rodzina algorytmów DE jest dokładnie opisana w rozdziale 3.3.

Drugim kluczowym algorytmem jest strategia ewolucyjna oparta o adaptację macierzy kowariancji (CMA-ES) [80] (ang. Covariance Matrix Adaptation Evolution Strategy). Jest to iteracyjna metaheurystyka, należąca do grupy Algorytmów Estymujących Rozkład [83] (ang. Estimation of Distribution Algorithms). CMA-ES zakłada, że położenie osobników w populacji jest losowane zgodnie z rozkładem normalnym określonym przez wartość średnią oraz macierz kowariancji, która podlega adaptacji. Algorytm CMA-ES, mimo że jest już w podstawowej wersji wysoce efektywną metaheurystyką, stanowi bazę dla wielu bardziej wyspecjalizowanych podejść [13, 232, 232]. Rodzina algorytmów CMA-ES jest szczegółowo opisana w rozdziale 3.4.

Mechanizmy adaptacji parametrów, obecne w szczególności w następcach DE oraz w CMA-ES, stanowią osobny nurt badawczy. Wiele, innych od DE i CMA-ES, podstawowych metaheurystyk zostało w podobny sposób rozszerzonych o mechanizmy adaptacyjne [249, 145, 231].

Inny, wysoce reprezentowany w literaturze nurt badawczy dotyczy projektowania al-

gorytmów populacyjnych inspirowanych naturą [234]. Poszukiwanie nawiązań do natury było szczególnie widoczne w algorytmach rojowych, które starały się naśladować m.in. zachowaniem pszczół [107], kukułek [235], niedźwiedzi polarnych [176], szczętków [67], bizonów [109] oraz czerwonych lisów [246]. Prezentowane algorytmy inspirowane były nie tylko zachowaniami zwierząt, ale także zachowaniem roślin [8] oraz zjawisk fizycznych [228].

Obecnie obserwuje się zwiększoną krytykę algorytmów inspirowanych naturą, która sugeruje brak innowacyjności i odtwórcze mechanizmy zawarte w niektórych metodach [155]. Istnieją też wątpliwości dotyczące jakości ewaluacji prezentowanych metaheurystyk, będące skutkiem braku powszechnie stosowanych procedur, które pozwoliłyby jednoznacznie ocenić ich wydajność [120].

Kolejnym nurtem badawczym są adaptacje algorytmów do innych rodzajów optymalizacji. Przykładem jest pierwotny GA, który został z powodzeniem zaadaptowany do optymalizacji ciągłej [35]. Z drugiej strony, algorytmy populacyjne przeznaczone do optymalizacji ciągłej mogą być adaptowane do rozwiązywania problemów optymalizacji dyskretnej [163, 186, 171]. Powiązanym nurtem badawczym są aplikacje istniejących metaheurystyk do poszczególnych problemów inżynieryjnych [149, 61, 70, 96].

Oprócz tworzenia nowych algorytmów oraz modyfikacji istniejących w literaturze, istnieje wiele prac poświęconych hybrydyzacji algorytmów populacyjnych [68, 114]. W szczególności należy tutaj wymienić algorytm GAPSO [221], czyli uogólniona adaptacyjna optymalizacja rojem cząstek (ang. Generalized Adaptive Particle Swarm Optimization), opisany w rozdziale 6.1.

Patrząc na historię algorytmów populacyjnych ze szczególnym uwzględnieniem ich uniwersalności oraz efektywności, za przełomowe można uznać następujące trzy wydarzenia badawcze. Pierwszym wydarzeniem było opracowanie podstawowych metaheurystyk, takich jak GA, PSO oraz DE, które za sprawą swojej uniwersalności są do dziś preferowane w wielu praktycznych zastosowaniach Drugim było opracowanie mechanizmów adaptacji, które istotnie zwiększyło efektywność metaheurystyk dedykowanych optymalizacji taniej oraz poskutkowało opracowaniem algorytmów takich jak CMA-ES oraz liczne rozszerzenia DE. Trzecim wydarzeniem było rozpoczęcie nurtu badawczego związanego z poszukiwaniem efektywnych rozwiązań hybrydowych.

3.3 Algorytmy z rodziny DE

Rodziną DE nazywany jest zbiór algorytmów powstałych w oparciu o bazowy algorytm DE [212]. Motywacją autorów do opracowania DE było spełnienie poniższych czterech wymagań, jakie są stawianie przed algorytmem optymalizacyjnym:

- 1. Zdolność do radzenia sobie z nieróżniczkowalnymi, nieliniowymi oraz wielomodalnymi funkcjami celu.
- 2. Możliwość zrównoleglania.
- 3. Łatwość użycia, ze szczególnym naciskiem na wykorzystanie niewielu parametrów, łatwych do określenia.
- 4. Dobre właściwości zbieżności, tzn. konsekwentna zbieżność do optimum globalnego w następujących po sobie niezależnych uruchomieniach procesu optymalizacji.

Bazowy DE występuje w wielu wariantach, które warunkują reguły losowego generowania nowych osobników. Możliwe warianty DE są przedstawione w m.in. w pracach [212, 251]. Wybór wariantu jest zależny od optymalizowanego problemu. Niektóre warianty mają większą zdolność do eksploracji, a niektóre do eksploatacji. Bez względu na wybór wariantu bazowy DE stanowi niezmienną strukturę dla wielu bardziej zaawansowanych metaheurystyk.

Poniżej opisano bazowy DE w wariancie DE/current-to-best/1, a następnie możliwe usprawniania, które skutkowały powstaniem nowych algorytmów należących do rodziny DE. Usprawnione wersje DE stanowią podstawę dla badań eksperymentalnych zaprezentowanych w rozdziale 5.

3.3.1 Bazowy DE (*DE/current-to-best/1*)

Bazowy DE to relatywnie prosty iteracyjny algoryt
m populacyjny, w którym populacja jest przekształcana trzykrotnie podczas jednej iteracji, tj. podczas fazy mutacji, fazy krzy-
żowania oraz fazy selekcji. Populacja P^g bazowego DE składa się ze stałej liczby
 N^g osobników $[\boldsymbol{x}_1^g,\ldots,\boldsymbol{x}_{N^g}^g]$, gdzi
egoznacza numer iteracji. Każdy i-ty osobnik
 $\boldsymbol{x}_i^g = [x_{i,1}^g,\ldots,x_{i,D}^g]$ reprezentuje D-wymiarowy punkt w przestrzeni.

W fazie mutacji dla każdego osobnika \boldsymbol{x}_i^g generowany jest zmutowany wektor \boldsymbol{v}_i^g . Dokładny sposób generacji wektora \boldsymbol{v}_i^g jest zależny od wariantu DE. Poniżej zaprezentowano generowanie wektora \boldsymbol{v}_i^g zgodnie z wariantem DE/current-to-best/1:

$$\boldsymbol{v}_{i}^{g} = \boldsymbol{x}_{i}^{g} + F(\boldsymbol{x}_{pbest_{i}}^{g} - \boldsymbol{x}_{i}^{g}) + F(\boldsymbol{x}_{r1_{i}}^{g} - \boldsymbol{x}_{r2_{i}}^{g})$$
(3.1)

gdzie F jest parametrem - czynnikiem skalującym, $pbest_i$ oznacza indeks losowo wybranego osobnika spośród najlepszych obecnie $p \cdot N^g$ osobników w populacji, gdzie p jest parametrem. Indeksy $r1_i, r2_i \in \{1, \ldots, N^g\}$, gdzie $r1_i \neq i, r2_i \neq i$ oraz $r1_i \neq r2_i$ oznaczają losowo wybranych z populacji dwóch osobników.

Następnie, macierzysty wektor \boldsymbol{x}_{i}^{g} jest krzyżowany ze zmutowanym wektorem \boldsymbol{v}_{i}^{g} , co prowadzi do uzyskania wektora próby $\boldsymbol{u}_{i}^{g} = [u_{i,1}^{g}, \ldots, u_{i,D}^{g}]$, w którym każdy element $u_{i,d}^{g}$, gdzie $d = 1, \ldots, D$ przyjmuje wartość $x_{i,d}^{g}$ (z prawdopodobieństwem krzyżowania CR) albo $v_{i,d}^{g}$, wpp. Dodatkowo, jeden losowo wybrany element $u_{i,d_{rand}}^{g}$ przyjmuje wartość $v_{i,d_{rand}}^{g}$ bez względu na rezultat probabilistyczny. Finalna postać wektora próby jest reprezentowana przez poniższe równanie:

$$u_{i,d}^g = \begin{cases} v_{i,d}^g, & \text{jeśli } rand(0,1) \leqslant CR \text{ or } d = d_i^{rand} \\ x_{i,d}^g, & \text{wpp.} \end{cases}$$
(3.2)

gdzie rand(0,1) oznacza liczbę losowaną z rozkładu jednostajnego z przedziału [0,1), a $d_i^{rand} \in \{1, \ldots, D\}$ jest losowo wybranym indeksem. Wartości rand(0,1) oraz d_i^{rand} są generowane niezależnie w każdej iteracji g dla każdego osobnika i. Wartość rand(0,1) jest dodatkowo generowana niezależnie dla każdego wymiaru $d \in \{1, \ldots, D\}$.

Finalnie, wektor próby \boldsymbol{u}_i^g jest poddawany fazie selekcji. Technicznie, jest on ewaluowany z wykorzystaniem funkcji celu f, a następnie jego wartość $f(\boldsymbol{u}_i^g)$ jest porównywana z wartością $f(\boldsymbol{x}_i^g)$ pochodzącą od macierzystego wektora \boldsymbol{x}_i^g . Wektor lepszy, ze względu na wartość funkcji celu, jest promowany do następnej iteracji g+1. Faza selekcji jest formalnie opisana przez następujące równanie:

$$\boldsymbol{x}_{i}^{g+1} = \begin{cases} \boldsymbol{u}_{i}^{g}, & \text{jeśli} \ f(\boldsymbol{u}_{i}^{g}) < f(\boldsymbol{x}_{i}^{g}) \\ \boldsymbol{x}_{i}^{g}, & \text{wpp.} \end{cases}$$
(3.3)

3.3.2 Rozszerzenia DE

Zmodyfikowane reguły opisujące fazę mutacji (równanie 3.1) oraz krzyżowania (równanie 3.2) to nie jedyne możliwe rozszerzenia bazowego DE. W literaturze zaproponowano wiele innych bardziej zaawansowanych wersji DE lub jego następców, o których można przeczytać m.in. w pracach przeglądowych [48, 47]. Relatywnie nowy przegląd metaheurystyk opartych o DE wraz z ich systematyką można znaleźć w pracy [125]. Spośród wielu zaprezentowanych algorytmów na szczególną uwagę zasługują modyfikacje, które istotnie zwiększyły wydajność bazowego podejścia oraz zostały uznane za uniwersalnie efektywne.

Jednym z pierwszych szeroko znanych algorytmów opartych o DE, który wykorzystują mechanizm adaptacji parametrów jest JADE, czyli adaptacyjna ewolucja różnicowa z opcjonalnym archiwum zewnętrznym [251] (ang. Adaptive Differential Evolution with Optional External Archive). Podejście zaprezentowane w JADE okazało się być wysoce efektywne, mimo, że znane były już inne metody adaptacji parametrów w DE (np. jDE [30], DESAP [219], FADE [133] czy SaDE [179]). Koncepcja JADE rozszerzała DE o dwa komponenty: adaptację parametrów (F i CR) oraz archiwum, które przechowuje macierzyste wektory, które zostały zastąpione w procesie selekcji. Adaptacja parametrów jest oparta o losowe generowanie wartości czynnika skalującego F_i^g oraz prawdopodobieństwa krzyżowani
a ${\cal CR}^g_i$ niezależnie dla każdego osobnika
 iw każdej iteracji g.Wartości
 ${\cal F}^g_i$ oraz CR_i^g są losowane na podstawie średniej z szeregu historycznych wartości (poddanych wygładzaniu wykładniczemu), które skutkowały poprawą rozwiązania w fazie selekcji. Drugi komponent - zbiór zapamiętanych macierzystych wektorów jest wykorzystywany w fazie mutacji, by wybierać osobników pochodzacych nie tylko z bieżacej populacji, ale ze zbioru bedacego połaczeniem bieżacej populacji oraz zbioru zapamiętanych macierzystych wektorów.

Następnie, na bazie JADE powstał SHADE, czyli oparta o sukces adaptacja parametrów dla ewolucji różnicowej [215] (ang. Success-History Based Parameter Adaptation for Differential Evolution). Zmodyfikowany został mechanizm adaptacji parametrów, w taki sposób, że dołączono dodatkowe archiwum przechowujące odnoszące sukcesy wartości parametrów czynnika skalującego F_i^g raz prawdopodobieństwa krzyżowania CR_i^g . Dzięki temu uzyskiwane parametry są bardziej zróżnicowane.

Algorytm SHADE jest szczegółowo opisany w rozdziale 3.3.3. W kontekście dalszych rozszerzeń mających zastosowanie do SHADE należy wyróżnić mechanizm *liniowej redukcji rozmiaru populacji* (LPSR) [217] (ang. *Linear Population Size Reduction*), który stoi za powstaniem algorytmu L-SHADE. Szczegółowy opis mechanizmu jest zawarty w rozdziale 3.3.4.

Algorytmy z rodziny DE zachowują dużą zdolność do globalnego przeszukiwania (eksploracji), co mityguje ryzyko przedwczesnego utknięcia populacji w sąsiedztwie minimum lokalnego. Niemniej, badania eksperymentalne pokazały, że opłacalne może być stosowanie mniejszych rozmiarów populacji i restartowanie procesu optymalizacji w przypadku utknięcia populacji w okolicy optimum lokalnego. Algorytm SHADE wykorzystujący mechanizm restartów został określony jako R-SHADE [216]. W przypadku algorytmu L-SHADE wykorzystującego mechanizm restartów przyjęto nazwę RL-SHADE. Opis mechanizmu restartów zastosowanego w R-SHADE (oraz RL-SHADE znajduje się w rozdziale 3.3.5.

Algorytm SHADE oraz jego rozszerzenia (L-SHADE, R-SHADE, RL-SHADE) stanowią wiodące metaheurystyki optymalizacyjne i służą za punkt odniesienia dla współcześnie opracowywanych algorytmów. W ciągu ostatnich kilku lat, w tym równolegle do prac badawczych wykonywanych przez autora rozprawy, powstało jeszcze kilka wysoce efektywnych algorytmów opartych o DE. Ich sposób działania jest w dużej mierze oparty o idee zawarte w SHADE, czyli adaptowanie sposobu mutacji oraz krzyżowania w oparciu o bieżącą wydajność określonych parametryzacji. Przykładami takich algorytmów są: LSHADE_cnEpSin [17], SALSHADE-cnEpSi [192], j2020 [31], IMODE [193], APGSK-IMODE [151], NL-SHADE-RSP [210] oraz MadDE [26].

Warto podkreślić, że algorytmy APGSK-IMODE, MadDE oraz NL-SHADE-RSP zajęły odpowiednio pierwsze, drugie i trzecie miejsce w konkursie *CEC2021 Special Session* and Competition on Single Objective Bound Constrained Optimization.

3.3.3 Mechanizm adaptacji parametrów (SHADE)

Algorytm SHADE [215] stanowi rozszerzenie DE o 2 komponenty: mechanizm adaptację parametrów z wykorzystaniem pamięci oraz zewnętrzne archiwum. Podobnie jak w przypadku DE, za wariant mutacji przyjęto *DE/current-to-best/1*. Poniżej przedstawiono opis różnic SHADE względem bazowego DE, jakie mają miejsce w fazie mutacji oraz krzyżowania. Następnie opisano zasadę wykorzystania zewnętrznego archiwum oraz mechanizmu adaptacji parametrów. Wysokopoziomowe spojrzenie na SHADE jest zaprezentowane za pomocą pseudokodu 2. W SHADE faza mutacji jest opisana następującym wzorem:

$$\boldsymbol{v}_{i}^{g} = \boldsymbol{x}_{i}^{g} + F_{i}^{g}(\boldsymbol{x}_{pbest_{i}}^{g} - \boldsymbol{x}_{i}^{g}) + F_{i}^{g}(\boldsymbol{x}_{r1_{i}}^{g} - \boldsymbol{x}_{r2_{i}}^{g})$$
(3.4)

gdzie czynnik skalujący F_i^g jest teraz generowany niezależnie w każdej iteracji g dla każdego i-tego osobnika. Ponadto indeks $r2_i \in \{1, \ldots, N^g + |A|\}$ określa osobnika należącego do sumy zbiorów P^g i zewnętrznego archiwum A. Analogicznie do DE, $pbest_i$ oznacza indeks losowo wybranego osobnika spośród najlepszych obecnie $p \cdot N^g$ osobników w populacji, gdzie p jest parametrem. Dalej zachodzi $r1_i \in \{1, \ldots, N^g\}, r1_i \neq i, r2_i \neq i$ oraz $r1_i \neq r2_i$.

Analogicznie w fazie krzyżowania, prawdopodobieństwo krzyżowania CR_i^g jest teraz

określanie w każdej iteracji g niezależnie dla każdego osobnika i. Sprowadza to równanie fazy krzyżowania do następującej postaci:

$$u_{i,d}^g = \begin{cases} v_{i,d}^g, & \text{jeśli } rand(0,1) \leqslant CR_i^g \text{ or } d = d_i^{rand} \\ x_{i,d}^g, & \text{wpp.} \end{cases}$$
(3.5)

gdzie, podobnie jak w bazowym DE, rand(0,1) jest losowane z rozkładu jednostajnego z przedziału [0,1), a $d_i^{rand} \in \{1, ..., D\}$ jest losowo wybranym indeksem. Adekwatnie, wartości rand(0,1) oraz d_i^{rand} są generowane niezależnie w każdej iteracji g dla każdego osobnika i, a wartość rand(0,1) jest dodatkowo generowana niezależnie dla każdego wymiaru $d \in \{1, ..., D\}$.

Finalna faza selekcji (równanie 3.6) jest tożsama z fazą selekcji znaną z bazowego DE z tą różnicą, że wartości \boldsymbol{x}_i^g , dla których spełniony jest warunek $f(\boldsymbol{u}_i^g) < f(\boldsymbol{x}_i^g)$ nie są zapominane i trafiają do zewnętrznego archiwum A.

$$\boldsymbol{x}_{i}^{g+1} = \begin{cases} \boldsymbol{u}_{i}^{g}, & \text{jeśli } f(\boldsymbol{u}_{i}^{g}) < f(\boldsymbol{x}_{i}^{g}) \\ \boldsymbol{x}_{i}^{g}, & \text{wpp.} \end{cases}$$
(3.6)

W SHADE, po fazie selekcji następują kolejno: aktualizacja zewnętrznego archiwum A oraz aktualizacja wybranego wpisu w pamięci, zgodnie z zasadami mechanizmu adaptacji parametrów. Rozmiar populacji jest stały, więc przyjęto $N^{g+1} = N^g$.

Algorytm 2 SHADE

- 1: Ustaw parametry $N, M_F, M_{CR}, p, a, H, N_a, N_s$
- 2: Inicjalizuj wszystkie wpisy w pamięci $M^0_{F,k}$ ora
z $M^0_{CR,k}$ za pomocą domyślnych wartości M_F ora
z M_{CR}
- 3: Wylosuj początkową populację $P^0 = [\pmb{x}_1^0, \dots, \pmb{x}_N^0],$ gdzie $N^0 = N$
- 4: g = 1
- 5: while budżet (dostępna liczba ewaluacji f.c.) nie jest wyczerpany ${\bf do}$
- 6: Generuj N^g zmutowanych wektorów \boldsymbol{v}_i^g zgodnie z równaniem (3.4)
- 7: Generuj N^g wektorów próby \boldsymbol{u}_i^g zgodnie z równaniem (3.5)
- 8: Wykonaj N^g ewaluacji punktów \boldsymbol{u}_i^g z wykorzystaniem funkcji celu f
- 9: Dokonaj N^g procesów selekcji (równanie 3.6)
- 10: Zaktualizuj archiwum A wykorzystując zastąpione wektory macierzyste \boldsymbol{x}_{i}^{g}
- 11: Zaktualizuj wybrany wpis w pamięci za pomocą $M_{F,k}^g$ oraz $M_{CR,k}^g$ zgodnie z równaniem (3.7)

12:
$$N^{g+1} = N^g$$

13: g = g + 1

14: end while

Zewnętrzne archiwum

Algorytm SHADE wykorzystuje zewnętrzne archiwum A które rozszerza bieżącą populację P^g o macierzyste wektory $\boldsymbol{x}_i^{g,j}$, które zostały zastąpione w fazie selekcji przez lepsze od nich wektory próby $\boldsymbol{u}_i^{g,j}$. Zewnętrzne archiwum jest rozmiaru $|A| = a \cdot N^g$, gdzie a jest parametrem. Jeśli archiwum jest pełne, to losowo wybrany element jest usuwany, by umożliwić wstawienie nowego elementu $(\boldsymbol{x}_i^{g,j})$ na jego miejsce.

Adaptacja i generowanie parametrów

Zarówno czynnik skalujący F_k^g (równanie 3.4), jak i prawdopodobieństwo krzyżowania CR_k^g (równanie 3.5) są adaptacyjnymi parametrami. Adaptacja parametrów jest oparta o mechanizm pamięci, przechowujący H historycznych wpisów, gdzie H jest parametrem. Poprzez wpis rozumie się parę $(M_{F,m}^g, M_{CR,m}^g)$, gdzie $m = 1, \ldots, H$. W każdej iteracji, po fazie selekcji, wszystkie wartości F_i^g and CR_i^g , które odniosły sukces (w fazie selekcji wektor próby \boldsymbol{u}_i^g był lepszy od odpowiadającego mu wektora macierzystego \boldsymbol{x}_i^g) są przechowywane odpowiednio w zbiorach S_F oraz S_{CR} . Oba zbiory podlegają transformacji z wykorzystaniem ważonej średniej Lehmera, by uzyskać dwie skalarne wartości $mean_{W_L}(S_{CR})$. Przekształcenie jest zilustrowane za pomocą poniższego wzoru:

$$mean_{W_L}(S) = \frac{\sum_{s=1}^{|S|} w_s S_s^2}{\sum_{s=1}^{|S|} w_s S_s}, \qquad w_s = \frac{\Delta f_s}{\sum_{r=1}^{|S|} \Delta f_r}$$
(3.7)

gdzie $\Delta f_p = f(\boldsymbol{x}_p^g) - f(\boldsymbol{u}_p^g), \ p \in \{s, r\}.$

Pary $(M_{F,m}^g, M_{CR,m}^g)$, stanowiące istniejące wpisy w pamięci, są aktualizowane sekwencyjnie za pomocą par $(mean_{W_L}(S_F), mean_{W_L}(S_{CR}))$ począwszy od k = 1 do k = H. Po aktualizacji ostatniego wpisu (k = H), procedura aktualizacji zaczyna od początku, tj. od k = 1.

Dodatkowo, jeżeli wszystkie wartości CR_i^g zawarte w zbiorze S_{CR} są równe 0, to procedura aktualizacji trwale oznacza wpis $M_{CR,k}^g$ za pomocą terminalnej wartości \perp (zamiast wpisać wartość $mean_{W_L}(S_{CR})$).

Wartości F_k^g oraz CR_k^g są generowane losowo z wykorzystaniem, odpowiednio, rozkładu Cauchy'ego oraz rozkładu normalnego, gdzie M_{F,r_i}^g oraz M_{CR,r_i}^g są wartościami średnimi obu rozkładów. Losowy indeks $r_i \in \{1, \ldots, H\}$ oznacza indeks pamięci, ale jest wyznaczany niezależnie dla każdego osobnika *i* w populacji P^g . Podsumowując, wartości F_i^g oraz CR_i^g są generowane w następujący sposób:

$$F_i^g = rand_{Cauchy}(M_{F,r_i}^g, 0.1) \tag{3.8}$$

$$CR_{i}^{g} = \begin{cases} 0 & \text{jeśli } M_{CR,r_{k}}^{g} = \bot \\ rand_{Normal}(M_{CR,r_{i}}^{g}, 0.1) & \text{w p.p} \end{cases}$$
(3.9)

3.3.4 Mechanizm liniowej redukcji rozmiaru populacji (L-SHADE)

Rozszerzeniem algorytmu SHADE jest L-SHADE, czyli SHADE rozszerzony o mechanizm liniowej redukcji rozmiaru populacji (LPSR) [217]. Mechanizm LPSR odpowiada za zmianę rozmiaru populacji w trakcie procesu optymalizacji. Parametrami mechanizmu są N_{init} oraz N_{min} , oznaczające, odpowiednio, początkowy oraz minimalny (końcowy) rozmiar populacji. Pod koniec każdej iteracji g rozmiar populacji N^{g+1} jest określany w następujący sposób:

$$N^{g+1} = round\left(\left(\frac{N_{min} - N_{init}}{MAX_{-}NFE}\right) \cdot NFE + N_{init}\right)$$
(3.10)

gdzie MAX_NFE oznacza budżet optymalizacji, a NFE reprezentuje liczbę ewaluacji f.c. jaka została wykonana dotychczas. Jeśli rozmiar populacji jest redukowany, to najgorsi osobnicy, w sensie wartości funkcji celu, są usuwani.

Algorytm SHADE, leżący u podstaw L-SHADE, wykorzystuje zewnętrzne archiwum A o rozmiarze $|A| = a \cdot N^g$. Stąd, redukcja rozmiaru populacji skutkuje redukcją rozmiaru archiwum. Redukcja rozmiaru archiwum A jest realizowana poprzez usunięcie losowego elementu bądź elementów pochodzących z tego archiwum, tak by osiągnąć pożądany rozmiar.

3.3.5 Mechanizm restartów (R-SHADE)

R-SHADE [216] jest rozwinięciem algorytmu SHADE o mechanizm restartów, który zakłada przerwanie bieżącego procesu optymalizacji oraz rozpoczęcie nowego niezależnego procesu. R-SHADE jest restartowany w przypadku, gdy zachodzi co najmniej jeden z następujących trzech warunków: (1) populacja zbiegła, (2) wartości funkcji w populacji zbiegły oraz (3) wystąpiła stagnacja w poprawianiu najlepszego dotychczas znalezionego rozwiązania. Pierwszy warunek zachodzi, jeśli istnieje wymiar $d' \in \{1, \ldots, D\}$, dla którego następująca nierówność jest spełniona ($\epsilon_x = 10^{-12}$):

$$\max_{i=1,\dots,N} \{x_{i,d'}^g\} - \min_{i=1,\dots,N} \{x_{i,d'}^g\} < \epsilon_x \max_{i=1,\dots,N} \{|x_{i,d'}^g|\}$$
(3.11)

Drugi warunek zachodzi, gdy spełniona jest poniższa nierówność ($\epsilon_f = 10^{-12}$):

$$\max_{i=1,\dots,N} \{ f(\boldsymbol{x}_{i}^{g}) \} - \min_{i=1,\dots,N} \{ f(\boldsymbol{x}_{i}^{g}) \} < \epsilon_{f} \max_{i=1,\dots,N} \{ |f(\boldsymbol{x}_{i}^{g})| \}$$
(3.12)

Trzeci warunek zachodzi, jeśli najlepsze dotych
czas znalezione rozwiązanie nie było poprawione w ciągu ostatnich
 $500 \cdot D$ ewaluacji f.c..

3.4 Algorytmy z rodziny CMA-ES

Rodziną CMA-ES nazywany jest zbiór algorytmów powstałych w oparciu o bazowy algorytm CMA-ES [80, 79, 76]. W odróżnieniu od DE, CMA-ES w swojej bazowej wersji jest wysoce efektywną metaheurystyką zbudowaną w oparciu o mechanizm adaptacji parametrów.

CMA-ES jest algorytmem aktywnie rozwijanym. Mimo, że jego rozwój sprowadza się przede wszystkim do konstruowania mechanizmów rozszerzających lub prób hybrydyzacji, to w literaturze można znaleźć kilka wariantów jego bazowej wersji. Różnice nie są fundamentalne i w większości sprowadzają się do różnego określenia metod ważenia składników we wzorach dot. adaptacji. Poniżej opisano bazowy CMA-ES stosując wariant zaprezentowany w [76], a następnie przedstawiono jego wybrane rozszerzenia.

3.4.1 Bazowy CMA-ES

Bazowy CMA-ES jest algorytmem populacyjnym, należącym do strategii ewolucyjnych, które nie nawiązują bezpośrednio do procesów obserwowanych w naturze. Algorytm w sposób iteracyjny generuje populację (zbiór punktów) zgodnie z określonym rozkładem normalnym. Ideą algorytmu CMA-ES jest sterowanie (poprzez parametry rozkładu) położeniem oraz kształtem generowanej populacji. Zmieniające się położenie, rozumiane jako wartość oczekiwana współrzędnych wygenerowanego punktu, pozwala na sukcesywne przesuwanie populacji w kierunku obiecujących regionów. Kształt generowanej populacji wynika z macierzy kowariancji stanowiącej drugi parametr rozkładu. Macierz kowariancji jest wyznaczana w oparciu o estymowany gradient funkcji celu w otoczeniu środka bieżącej populacji.

Algorytm CMA-ES wykorzystuje wiele parametrów, lecz jedynym parametrem podawanym bezpośrednio przez użytkownika jest rozmiar populacji λ . Pozostałe parametry podlegają adaptacji w każdej iteracji bądź przyjmują stałe wartości właściwe dla danego wariantu CMA-ES. Wysokopoziomowe spojrzenie na CMA-ES jest zaprezentowane za pomocą pseudokodu 3.

Algorytm 3 CMA-ES				
1: Ustaw paramet	Ustaw parametr wielkości populacji λ			
2: Inicializuj pocz	Inicializuj początkowe parametry rozkładu: \pmb{m}^0,σ^0,C^0			
3: Inicializuj pozo	Inicializuj pozostałe parametry: μ , \boldsymbol{w} , c_m , c_c , c_σ , c_μ , c_1 , d_σ			
4: $g = 0$				
5: while budzet (dostępna liczba ewalu acji f.c.) nie jest wyczerpany ${\bf do}$			
6: Generuj λ 1	niezależnych punktów \boldsymbol{x}_i^g zgodnie z równaniem (3.13)			
7: Wykonaj N	g ewaluacji punktów \pmb{x}^g_i z wykorzystaniem funkcji celu f			
8: Aktualizuj	wartość średnią rozkładu \boldsymbol{m}^{g+1} zgodnie z równaniem 3.14			
9: Aktualizuj	ścieżkę ewolucji \pmb{p}_c^{g+1} zgodnie z równaniem 3.16			
10: Aktualizuj	ścieżkę ewolucji $oldsymbol{p}_{\sigma}^{g+1}$ zgodnie z równaniem 3.18			
11: Aktualizuj	macierz kowariancji C^{g+1} zgodnie z równaniem 3.19			
12: Aktualizuj	długość kroku σ^{g+1} zgodnie z równaniem 3.21			
13: $g = g + 1$				
14: end while				

W algorytmie CMA-ES w każdej iteracji g generowana jest populacja P^g składająca się z λ osobników $[\boldsymbol{x}_1^g, \ldots, \boldsymbol{x}_{\lambda}^g]$, gdzie $i \in 1, \ldots, \lambda$. Każdy *i*-ty osobnik $\boldsymbol{x}_i^g = [x_{i,1}^g, \ldots, x_{i,D}^g]$ reprezentuje D-wymiarowy punkt w przestrzeni. Położenie każdego *i*-tego osobnika \boldsymbol{x}_i^g należącego do populacji jest wyznaczane zgodnie z rozkładem normalnym, w następujący sposób:

$$\boldsymbol{x}_{i}^{g} = \boldsymbol{m}^{g} + \sigma^{g} \cdot \boldsymbol{y}_{i}^{g}, \qquad \boldsymbol{y}_{i}^{g} \sim \mathcal{N}(\boldsymbol{0}, C^{g})$$
(3.13)

gdzie $\mathbf{m}^g \in \mathbb{R}^D$ jest parametrem oznaczającym wartość średnią rozkładu, σ jest wartością długości kroku, a C^g jest macierzą kowariancji, która określa kształt elipsy rozkładu. Wszystkie trzy parametry podlegają adaptacji. Bezpośrednio po wygenerowaniu populacji P^g następuje ewaluacja wszystkich punktów \mathbf{x}_i^g , z wykorzystaniem funkcji celu f.

Następnie odbywa się aktualizacja (adaptacja) parametrów rozkładu na potrzeby iteracji g+1. W pierwszym kroku wyznaczana jest nowa wartość średnia rozkładu \boldsymbol{m}^{g+1} . W tym celu populacja $P^g = [\boldsymbol{x}_1^g, \dots, \boldsymbol{x}_{\lambda}^g]$ jest sortowana od najlepszego do najgorszego rozwiązania, ze względu na wartości $f(\boldsymbol{x}_i^g)$, Do wyznaczenia wartości \boldsymbol{m}^{g+1} wykorzystuje się μ początkowych rozwiązań z posortowanego szeregu, gdzie $\mu \approx \lambda/2$ jest parametrem. Sprowadza się to do następującego równania:

$$\boldsymbol{m}^{g+1} = \boldsymbol{m}^g + c_m \sum_{i=1}^{\mu} w_i (\boldsymbol{x}^g_{i:\lambda} - \boldsymbol{m}^g)$$
(3.14)

gdzie $c_m \leq 1$ jest stałą uczenia, zazwyczaj ustawianą na 1. Wektor $\boldsymbol{x}_{i:\lambda}^g$ reprezentuje *i*tego najlepszego osobnika ze względu na wartość $f(\boldsymbol{x}_{i:\lambda}^g)$. Wektor \boldsymbol{w} jest parametrem reprezentującym wagi spełniające następującą zależność:

$$\sum_{i=1}^{\mu} w_i = 1, \qquad w_1 \geqslant w_2 \geqslant \ldots \geqslant w_\mu > 0 \tag{3.15}$$

Następnie wyznaczana jest ścieżka ewolucji p_c^{g+1} zgodnie z poniższym wzorem:

$$\boldsymbol{p}_{c}^{g+1} = (1 - c_{c})\boldsymbol{p}_{c}^{g} + \sqrt{c_{c}(2 - c_{c})\mu_{eff}} \,\frac{\boldsymbol{m}^{g+1} - \boldsymbol{m}^{g}}{\sigma^{g}}$$
(3.16)

gdzie $\mathbf{p}_c^g \in \mathbb{R}^D$ jest ścieżką ewolucji z poprzedniej generacji ($\mathbf{p}_c^0 = \mathbf{0}^D$), $c_c \leq 1$ jest parametrem stanowiącym o szybkości adaptacji, a μ_{eff} jest współczynnikiem wyznaczanym na podstawie wektora wag w następujący sposób:

$$\mu_{eff} = \frac{1}{\sum_{i=1}^{\mu} w_i^2} \tag{3.17}$$

W kolejnym kroku wyznaczana jest ścieżka ewolucji p_{σ}^{g+1} zgodnie z poniższym wzorem:

$$\boldsymbol{p}_{\sigma}^{g+1} = (1 - c_{\sigma})\boldsymbol{p}_{\sigma}^{g} + \sqrt{c_{\sigma}(2 - c_{\sigma})\mu_{eff}} (C^{g})^{-\frac{1}{2}} \frac{\boldsymbol{m}^{g+1} - \boldsymbol{m}^{g}}{\sigma^{g}}$$
(3.18)

gdzie $\boldsymbol{p}_{\sigma}^{g} \in \mathbb{R}^{D}$ jest ścieżką ewolucji z poprzedniej generacji ($\boldsymbol{p}_{\sigma}^{0} = \boldsymbol{0}^{D}$), $c_{\sigma} < 1$ jest parametrem stanowiącym o szybkości adaptacji, a $(C^{g})^{-\frac{1}{2}} \stackrel{\text{def}}{=} (B^{g})(D^{g})^{-1}(B^{g})^{T}$, gdzie $C^{g} = (B^{g})(D^{g})^{2}(B^{g})^{T}$ jest rozkładem macierzy C^{g} szerzej opisanym w [76].

Mając wyznaczone wartości ścieżki ewolucji \boldsymbol{p}_c^{g+1} oraz ścieżki ewolucji $\boldsymbol{p}_{\sigma}^{g+1}$ aktualizacji podlega macierz kowariancji C^{g+1} , zgodnie poniższym wzorem:

$$C^{g+1} = (1 - c_1 - c_\mu \sum_{i}^{\mu} w_i) C^g + c_1 \boldsymbol{p}_c^{g+1} (\boldsymbol{p}_c^{g+1})^T + c_\mu \sum_{i=1}^{\lambda} w_i \boldsymbol{y}_{i:\lambda}^{g+1} (\boldsymbol{y}_{i:\lambda}^{g+1})^T$$
(3.19)

gdzie $c_1 \approx 1/D^2$ oraz $c_{\mu} \approx min(\mu_{eff}/D^2, 1-c_1)$ są parametrami, a wektor $\boldsymbol{y}_{i:\lambda}^{g+1}$ zgodnie z równaniem 3.14 jest zdefiniowany w następujący sposób:

$$\boldsymbol{y}_{i:\lambda}^{g+1} = (\boldsymbol{x}_{i:\lambda}^{g+1} - \boldsymbol{m}^g) / \sigma^g$$
(3.20)

W ostatnim kroku adaptacji podlega długość kroku σ zgodnie z poniższym równaniem:

$$\sigma^{g+1} = \sigma^g \exp\left(\frac{c_\sigma}{d_\sigma} \left(\frac{||\boldsymbol{p}_{\sigma}^{g+1}||}{E||\mathcal{N}(\mathbf{0},\mathbf{I})||} - 1\right)\right)$$
(3.21)

gdzie $d_{\sigma} \approx 1$ jest parametrem tłumienia, a $E||\mathcal{N}(\mathbf{0},\mathbf{I})||$ oznacza wartość oczekiwaną normy Euklidesowsniej wektora otrzymanego zgodnie z rozkładem normalnym o średniej w punkcie $\mathbf{0}^D$ oraz jednostkowej macierzy kowariancji **I**.

3.4.2 Rozszerzenia CMA-ES

Mimo relatywnie wysokiej efektywności podstawowego algorytmu, w literaturze zaproponowano wiele rozszerzeń CMA-ES. Jednakże, sam sposób generowania osobników oraz koncepcja adaptacji macierzy kowariancji pozostaje wspólna dla większości nowszych metaheurystyk.

Algorytmem będącym bezpośrednim następcą CMA-ES jest IPOP-CMA-ES [13], który wprowadził mechanizm zwiększania rozmiaru populacji (IPOP - ang. *increasing population*). Zwiększanie rozmiaru populacji λ odbywa się bezpośrednio po każdym restarcie przed rozpoczęciem nowego niezależnego procesu optymalizacji. Stąd, parametr λ jest rozumiany jako początkowy rozmiar populacji, wykorzystywany w pierwszym przebiegu. Zwiększenie rozmiaru populacji jest realizowane poprzez przemnożenie bieżącego rozmiaru λ przez stałą wartość. W pracy [13] zasugerowano wartość mnożnika między 1.5 a 5.

Rozszerzeniem algorytmu IPOP-CMA-ES jest BIPOP-CMA-ES [75], który zakłada możliwość zwiększania rozmiaru populacji po restarcie lub utrzymania relatywnie małego, lecz zmieniającego się rozmiaru populacji. Zaprezentowano też warianty (PSA-CMA-ES [161], CMAES-APOP [160]), które w dokonują adaptacji rozmiaru populacji λ w trakcie przebiegu optymalizacji. Algorytm KL-BIPOP-CMA-ES [232] dynamicznie ustala inicjalną wartość długości kroku σ po każdym restarcie. NIPOP-aCMA-ES [138] po każdym restarcie jednocześnie modyfikuje rozmiar populacji λ oraz długość kroku σ . Badania nad alternatywnymi mechanizmami adaptacji długości kroku σ można znaleźć w [7, 78, 11]. Operacje algebraiczne związane z adaptacją macierzy kowariancji sprawiają, że bazowy CMA-ES charakteryzuje się relatywnie wysoką złożonością obliczeniową w zadaniach z dużą liczbą wymiarów. Stąd, zaprezentowano kilka wariantów przeznaczonych dla optymalizacji z dużą liczbą wymiarów [223, 99].

Mechanizmy restartów są właściwe dla konkretnych algorytmów należących do rodziny CMA-ES. Przeważnie są one zależne od bieżącej wartości adaptowanych parametrów. Przykładowo, algorytm IPOP-CMA-ES zawiera pięć reguł wyzwalających restart. Trzy z nich dotyczą struktury macierzy kowariancji C^g , jeden dotyczy wartości długości kroku σ oraz wartości $\sigma \cdot \mathbf{p}_c$, i tylko ostatni jest związany z brakiem poprawy najlepszego dotychczas znalezionego rozwiązania. Mechanizm restartów BIPOP-CMA-ES zawiera dziewięć reguł, z czego tylko dwie pochodzą z IPOP-CMA-ES. Z kolei, CMAES-APOP zaadaptował osiem z dziewięciu warunków restartów z BIPOP-CMA-ES. Inne podejście do restartów zaprezentowano w PSA-CMA-ES, gdzie mechanizm adaptacji rozmiaru populacji zastąpił mechanizm restartów, lecz autorzy sugerują, że jego dodanie mogłoby być zasadne.

Rozdział 4

Modelowanie funkcji celu w optymalizacji ciągłej

W tym rozdziale zaprezentowano wiodące metody modelowania funkcji celu. Wprowadzono pojęcie metamodelu oraz zbioru uczącego, który jest przez niego wykorzystywany. Następnie opisano popularne grupy metamodeli. Metamodele oparte o regresję wielomianową znalazły zastosowanie w autorskich metodach opisywanych w rozdziale 6 oraz rozdziale 7. Poddano pod dyskusję wydajność procesu uczenia przedstawionych metamodeli, ze szczególny+m uwzględnieniem złożoności obliczeniowej \mathcal{O} .

Następnie opisano strategię *optymalizacji bayesowskiej*, która jest wiodącą metodą rozwiązującą problemy kosztowne. Przedstawiono metody integracji metamodeli w algo-rytmach populacyjnych. Opisano wybrane algorytmy populacyjne wspomagane metamodelem.

4.1 Przybliżanie funkcji celu

Koszt ewaluacji funkcji celu istotnie wpływa na dostępny budżet optymalizacji. Stąd, poszukuje się metod pozwalających na zastąpienie (pełnej) ewaluacji f.c. pewną formą uproszczonej ewaluacji, która zwraca przybliżoną wartość f.c. Uproszczona ewaluacja jest w założeniu znacząco mniej kosztowna, lecz również mniej dokładna. Im koszt optymalizacji jest większy, tym większa jest motywacja dla stosowania przybliżeń. Zastępowanie ewaluacji f.c. za pomocą uproszczonych ewaluacji pozwala na:

• Zmniejszenie całkowitego kosztu optymalizacji poprzez redukcję liczby ewaluacji f.c.

- Zwiększenie jakości otrzymanego rozwiązania przy zachowaniu zadanej liczby ewaluacji f.c.
- Przeprowadzenie procesu optymalizacji, w sytuacji gdy wykonanie ewaluacji f.c. w ogóle nie jest możliwe.

Istnieje wiele sposobów na wykonanie uproszczonej ewaluacji f.c. Przede wszystkim, dla części rzeczywistych problemów wymagających działań wykraczających poza obliczenia komputerowe istnieje możliwość zastąpienia ewaluacji f.c. za pomocą symulacji obliczeniowych [100]. Koncepcja symulacji obliczeniowych zastępujących ewaluację f.c. jest szczególnie istotna w zagadnieniach fizycznych, czego przykładem są symulacje struktur aerodynamicznych [208] czy konstrukcji budowlanych [159, 58]. W takim przypadku poprzez ewaluację f.c. rozumie się konstrukcję rzeczywistego obiektu. Zastąpienie ewaluacji f.c. za pomocą symulacji komputerowej w sposób znaczący obniża czas i koszt uzyskania (przybliżonej) wartości f.c. Symulacjom podlegają też zjawiska obserwowane w sieciach komputerowych [202], procesach produkcyjnych [5] oraz w konstrukcjach motoryzacyjnych [142].

Jakość symulacji podlega stopniowaniu: od dokładnych do mniej dokładnych. Mniej dokładne symulacje są mniej kosztowne (szybsze), lecz dają wynik obarczony większym błędem. Przykładowo, w problemach aerodynamicznych dokładne trójwymiarowe symulacje przepływów są wciąż wysoce kosztowne, więc mogą zostać zastąpione przez mniej dokładne (dwuwymiarowe). Innym problemem, w którym ewaluacja f.c. może zostać zastąpiona przybliżoną wartością jest optymalizacja hiperparametrów w modelach uczenia maszynowego. W tym przypadku przybliżona wartość nie jest uzyskiwana w sposób symulacyjny, lecz jest możliwe wykorzystanie ograniczonych zbiorów treningowych i walidujących, co sprowadza się do uzyskania wartości f.c. szybciej, lecz będzie ona obarczona większą niepewnością.

Część problemów zakłada wprost że ewaluacja f.c. jest realizowana poprzez wykonanie symulacji. Przykładem tego są m.in.: symulacje zdarzeń w systemie leczniczym [238], łańcuchach dostaw [106], transporcie [169] oraz hydrologii [237]. We wszystkich ww. przykładach nie jest możliwe skonstruowanie rzeczywistego środowiska eksperymentalnego, więc dokładna symulacja jest traktowana jako pełna ewaluacja f.c. Zagadnienia z dziedziny finansowej, które zakładają możliwość wystąpienia nieskończenie wielu scenariuszy, takie jak wybór optymalnego portfolio [156], podobnie mogą być oparte o wynik symulacyjny. Szerszy przegląd podejść symulacyjnych można znaleźć m.in. w pracach [73] oraz [66]. Warto podkreślić, że nie we wszystkich problemach istnieje możliwość zastąpienia ewaluacji f.c. za pomocą symulacji bądź innej uproszczonej formy wyznaczenia f.c. Przykładem tego jest poszukiwanie optymalnego miejsca odwiertu celem pozyskania dostępu do złóż mineralnych [175]. W takim przypadku jedyną możliwością poznania wartości f.c. (ilości i jakości dostępnych złóż) w danym punkcie jest wykonanie fizycznego odwiertu.

Różnorodność problemów oraz sposobów uproszczonych ewaluacji f.c. sprawiła, że zaczęto poszukiwać uniwersalnych metod pozwalających na szybkie otrzymanie przybliżonej wartości f.c. bez wchodzenia w interakcję ze środowiskiem. Innymi słowy, poszukiwano metod pozwalających przybliżać f.c., traktując problem w pełni czarnoskrzynkowo. Stąd, zaczęto konstruować modele f.c., które nazwano metamodelami lub modelami zastępczymi (ang. *surrogate models*).

Metamodel \hat{f} jest funkcją, która ma za zadanie przybliżać f.c., tzn. pozwala na uzyskanie przybliżonej wartości f.c. w dowolnym punkcie przestrzeni rozwiązań [136]. Zastosowanie metamodeli w optymalizacji zostało szerzej opisane w rozdziale 4.2.

Pełny schemat gradacji sposobów pozyskania informacji o wartości f.c. w danym punkcie jest przedstawiony na rysunku 4.1.



Rysunek 4.1: Gradacja sposobów pozyskania informacji o wartości funkcji celu w danym punkcie. Kolorem szarym zaznaczono sposoby właściwe dla algorytmów populacyjnych wykorzystujące metamodele.

Podsumowując, w domenie optymalizacji czarnoskrzynkowej jedyną uniwersalną meto-

dą uproszczenia ewaluacji f.c. jest zastąpienie jej wartością metamodelu, co potwierdzają liczne prace przeglądowe [22, 181, 63, 226]. Stad, w dalszej części rozprawy za możliwe przyjęto tylko dwa sposoby uzyskania informacji o wartości f.c.: (pełna) ewaluacja f.c. oraz zastąpienie ewaluacji f.c. wartością metamodelu.

4.2 Metamodele w optymalizacji

Zadaniem metamodelu $\hat{f}: \mathbb{R}^D \to \mathbb{R}$ jest przybliżanie f.c. $f: \mathbb{R}^D \to \mathbb{R}$ w przestrzeni rozwiązań $S \subset \mathbb{R}^D$. Metamodel \hat{f} jest estymowany w oparciu o pewien zbiór uczący \mathcal{D} , zwany inaczej zbiorem treningowym lub zbiorem obserwacji, składający się z rozwiązań, które były uprzednio poddane ewaluacji z wykorzystaniem f.c. Rozmiar oraz sposób konstrukcji zbioru jest właściwy dla danego rodzaju metamodelu oraz jego sposobu integracji z metodą rozwiązującą. Zbiór uczący jest zdefiniowany w następujący sposób:

$$\mathcal{D} = \{ (\boldsymbol{x}_i, y_i) \} \tag{4.1}$$

gdzie $i = 1, ..., K, \mathbf{x}_i \in \mathbb{R}^D, y_i \in \mathbb{R}$, a K jest licznością zbioru. Wektor $\mathbf{x}_i = [x_{i,1}, ..., x_{i,D}]$ reprezentuje D-wymiarowy punkt w przestrzeni rozwiązań, a y_i odpowiadającą mu wartość f.c. Poszczególne współrzędne punktu \mathbf{x}_i są dalej oznaczane poprzez $x_{i,d}$, gdzie $d \in \{1, ..., D\}$.

Podstawowa klasyfikacja metamodeli zakłada podział na metamodele globalne oraz metamodele lokalne. Metamodel globalny ma za zadanie odwzorować f.c. w całej dostępnej przestrzeni rozwiązań. Metamodel lokalny zakłada odwzorowanie f.c. w ograniczonym obszarze przestrzeni rozwiązań, w szczególności w otoczeniu zadanego punktu.

W ogólności, każdy z metamodeli może odwzorowywać f.c. globalnie albo lokalnie, w zależności od definicji zbioru uczącego \mathcal{D} . Niemniej, pewne klasy metamodeli są predestynowane do modelowania globalnego, a inne do lokalnego, co wynika przede wszystkim z ich stopnia parametryzacji (zazwyczaj im większa liczba parametrów, tym większa zdolność modelowania globalnego).

Metamodele są wykorzystywane na trzy sposoby (rysunek 4.2). Pierwszy sposób zakłada, że odwzorowanie optymalizowanej funkcji jest nadrzędnym zadaniem metamodelu. W problemach czarnoskrzynkowych wykonanie określonej liczby ewaluacji f.c. skutkuje uzyskaniem informacji o jej wartościach wyłącznie w ewaluowanych punktach. Zastosowanie metamodelu pozwala na przekształcenie otrzymanego zbioru punktów oraz towarzyszących im wartości w model f.c. w przestrzeni ciągłej. Dzięki temu możliwe jest na przykład wykonanie wizualizacji f.c. bądź dokonanie eksperckiego wyboru najlepszego (niekoniecznie optymalnego) rozwiązania, które w ogólności nie było poddane ewaluacji (jest jedynie przybliżane).

Drugi sposób jest rozwinięciem sposobu pierwszego, tzn. modelowanie f.c. ma bezpośrednie zastosowanie w dowolnej metodzie rozwiązującej problem optymalizacji. Użycie metamodelu w procesie optymalizacji ma w założeniu pozwolić na znalezienie satysfakcjonującego rozwiązania $\boldsymbol{x}^{\epsilon}$ szybciej bądź otrzymane rozwiązanie $\boldsymbol{x}^{\epsilon}$ będzie obarczone mniejszym błędem ϵ .



Rysunek 4.2: Sposoby wykorzystania metamodeli.

Trzeci sposób zakłada, że metodą rozwiązująca problem optymalizacji ze wsparciem metamodelu będzie algorytm populacyjny. Algorytmy populacyjne wykorzystujące metamodel są określana w literaturze jako *algorytmy ewolucyjne wspierane metamodelami* (SAEA) [101, 103] (ang. *Surrogate-Assisted Evolutionary Algorithms*). Dla zachowania spójnego nazewnictwa algorytmy populacyjne wykorzystujące metamodel będą dalej nazywane *algorytmami populacyjnymi wspieranymi metamodelem* (APWM). Właściwy dla danego APWP jest rodzaj metamodelu oraz sposób jego integracji. Popularne grupy metamodeli opisano niżej w rozdziale 4.3. Możliwe sposoby integracji są omówione w rozdziale 4.5.1.

4.3 Popularne grupy metamodeli

W literaturze za wiodące grupy metamodeli [136, 126, 252] uznaje się: regresję wielomianową [158] (ang. *Polynomial Regression*), Kriging [146] zwany inaczej regresją Procesu Gaussowskiego (ang. *Gaussian Process regression*), sztuczne sieci neuronowe (ang. *Arti-ficial Neural Network* oraz radialną funkcję bazową (ang. *Radial basis function*). Poniżej zaprezentowano zwięzły opis każdej z ww. grup metamodeli. Regresję wielomianową opisano w rozdziale 4.3.1, rozszerzenia regresji wielomianowej w rodziale 4.3.1, Kriging w rozdziale 4.3.2, sztuczne sieci neuronowe w rozdziale 4.3.3, a radialną funkcję bazową w rozdziale 4.3.4.

Przykładami mniej powszechnych, lecz wciąż użytecznych grup metamodeli są: Uogólnione modele addytywne [82] (ang. Generalized Additive Models), wielowymiarowe adaptacyjne splajny regresyjne [65] (ang. Multivariate Adaptive Regression Splines), regresja wektorów nośnych [206] (ang. Support Vector Regression), lasy losowe [49] (ang. Random Forest), jądrowa regresja metodą cząstkowych najmniejszych kwadratów [188] (ang. Kernel Partial Least Squares Regression) oraz regresja k-najbliższych sąsiadów [9] (ang. k-Nearest Neighbors Regression).

4.3.1 Regresja wielomianowa

Regresja wielomianowa [158] (RW) jest rodzajem liniowej regresji, która zakłada nieliniowe przekształcenie zmiennej objaśniającej \tilde{x} za pomocą wektora parametrów β , można przedstawić w następujący sposób:

$$\hat{f} = \tilde{\boldsymbol{x}}^T \boldsymbol{\beta} \tag{4.2}$$

gdzie \hat{f} jest przybliżeniem funkcji celu f.

Podstawowa RW zakłada przekształcenie zmiennej objaśniającej $x_{i,d} \in \boldsymbol{x}_i$ (równanie 4.1), gdzie $d \in \{1, \ldots, D\}$ do postaci $x_{i,d}^{p'}$, gdzie $p' = 1, \ldots, p$, a p jest przyjętym stopniem wielomianu. W przypadku liniowej RW (czyli regresji liniowej) przyjmuje się $\tilde{\boldsymbol{x}} = [x_{i,2}, \ldots, x_{i,D}, 1]$. W przypadku kwadratowej RW przyjmuje się $\tilde{\boldsymbol{x}} = [x_{i,1}^2, \ldots, x_{i,D}^2, 1]$. W ogólności, dla wielomianu stopnia p zachodzi $\tilde{\boldsymbol{x}} = [x_{i,1}^p, \ldots, x_{i,D}^2, 1]$.

Wyznaczenie wektora współczynników $\boldsymbol{\beta}$ może być wykonane za pomocą *metody naj*mniejszych kwadratów [229], zwaną dalej MNK. MNK pozwala na otrzymanie oszacowania wektora współczynników $\hat{\boldsymbol{\beta}}$ za pomocą na stępującego równania:

$$\boldsymbol{\beta} = \left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{y} \tag{4.3}$$

gdzie $\boldsymbol{X} = [\tilde{\boldsymbol{x}_1}, \dots, \tilde{\boldsymbol{x}_K}]^T$ oraz $\boldsymbol{y} = [y_1, \dots, y_K]^T$.

Rozszerzenia regresji wielomianowej

Podstawowe rozszerzenia RW zakładają wprowadzenie dodatkowych przekształceń zmiennej objaśniającej $x_{i,d}$. W szczególności kwadratową Regresję Wielomianową rozszerza się o dodanie interakcji między zmiennymi w postaci $x_{i,d} \cdot x_{i,d'}$, gdzie $d, d' \in \{1, \ldots, D\}$ oraz $d \neq d'$.

Stosowane jest również ważenie obserwacji (\boldsymbol{x}_i, y_i) , pochodzących ze zbioru uczącego \mathcal{D} podczas estymacji wektora współczynników $\boldsymbol{\beta}$ w MNK. Motywacją może być próba nadania większej wagi obserwacjom nowszym lub leżącym w obszarze uznanym za otoczenie rozwiązania optymalnego. Wprowadzenie wektora wag $\boldsymbol{w} = [w_1, \ldots, w_K]$ sprowadza się do utworzenia diagonalnej macierzy \boldsymbol{W} , w której wektor \boldsymbol{w} jest umieszczony na jej głównej przekątnej, następnie przekształcenia równania 4.3 do następującej postaci:

$$\boldsymbol{\beta} = \left(\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X} \right)^{-1} \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{y}$$
(4.4)

Z RW wiąże się pojęcie *Response Surface Methodology* [27, 157] (RSM). Główną ideą RSM jest wykorzystanie sekwencji *zaprojektowanych eksperymentów* (ang. *designed experiments*) w celu uzyskania optymalnej odpowiedzi, rozumianej jako wartość f.c. RSM pierwotnie zakładało modelowanie f.c. za pomocą kwadratowej RW wraz z interakcjami. Modelowanie f.c. za pomocą RW oraz pojęcie RSM bywa stosowane wymiennie.

Obszar modelowania w regresji wielomianowej

RW jest używana do modelowania f.c. w sposób globalny, jak i lokalny, w zależności od definicji wektora zmiennych objaśniających \tilde{x} oraz sposobu konstrukcji zbioru uczącego \mathcal{D} . Stosowanie wielomianów wyższego stopnia niż p = 1 oraz dodanie interakcji predestynuje regresję do modelowania globalnego. Prostsze warianty RW znajdą zastosowanie w modelowaniu lokalnym - są mniej dokładne lecz wymagają mniejszej liczby obserwacji w procesie nauki.

Poprzez sposób konstrukcji zbioru uczącego rozumie się reguły określające które obserwacje (\boldsymbol{x}_i, y) do niego trafiają. Możliwe jest konstruowanie zbioru uczącego z wykorzystaniem obserwacji pochodzących z całej przestrzeni przeszukiwania (modelowanie globalne) lub z otoczenia wybranego punktu (modelowanie lokalne). Możliwy jest także selektywny wybór obserwacji, np. *K*-najlepszych obserwacji (ze względu na y_i) odkrytych dotychczas.

Złożoność obliczeniowa regresji wielomianowej

Złożoność obliczeniowa MNK [111] wynosi $\mathcal{O}(L^2K)$, gdzie $L = |\tilde{\boldsymbol{x}}|$ jest liczbą parametrów RW, a K jest liczbą obserwacji w zbiorze uczącym. Dla liniowej RW L = D + 1, gdzie D jest liczbą wymiarów problemu, więc $\mathcal{O}(D^2K)$. Dla kwadratowej RW L = 2D + 1, co pozostawia złożoność $\mathcal{O}(D^2K)$ bez zmian. Uwzględnienie interakcji istotnie zwiększa złożoność obliczeniową RW, ponieważ liczba parametrów modelu $L = (D^2 + 4D - D)/2$, w wyniku tego otrzymujemy $\mathcal{O}(D^4K)$.

Zakładając, że liczba obserwacji uczących K jest niewiele większa niż liczba parametrów L, to dla liniowej oraz kwadratowej regresji otrzymujemy $\mathcal{O}(D^3)$, a dla kwadratowej regresji z interakcjami $\mathcal{O}(D^6)$.

4.3.2 Kriging

Kriging [146], określany również jako regresja procesu gaussowskiego, jest deterministyczną metodą interpolującą, mającą inspirację w geostatystyce. Kriging jest metamodelem estymowanym na podstawie zbioru uczącego \mathcal{D} , zdefiniowanego w równaniu 4.1. Składa się on z dwóch komponentów: funkcji globalnego trendu, nazywanej również funkcją deterministyczną lub funkcją globalną, oraz Gaussowskiej funkcji zmienności, która modeluje lokalne odchylenia od globalnego trendu. Zdolność do jednoczesnego prognozowania wartości oraz wyznaczania niepewności oszacowania w zadanych punktach jest powodem popularności tego sposobu modelowania nieznanej f.c. [39]. Istnieje wiele wariantów metody Kriging, różniących się szczegółową definicją funkcji globalnego trendu oraz funkcji zmienności [166]. Poniżej opisano wariant zdefiniowany w pracy [38].

Metamodel Kriging zakłada, że funkcja celu jest przybliżana w następujący sposób:

$$\hat{f}(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \epsilon(\boldsymbol{x}) \tag{4.5}$$

gdzie \boldsymbol{x} dowolnym punktem w *D*-wymiarowej przestrzeni przeszukiwań, μ jest funkcją globalnego trendu, rozumianą jako predykcja modelu regresji $F(\boldsymbol{\beta}, \boldsymbol{x})$, a ϵ oznacza zmienność opisaną rozkładem Gaussa o średniej 0 i wariancji σ^2 :

$$\epsilon(\boldsymbol{x}) = \mathcal{N}(0, \sigma^2) \tag{4.6}$$

Model regresji $F(\boldsymbol{\beta}, \boldsymbol{x})$ jest zdefiniowany jako liniowa kombinacja L funkcji, której współczynniki pochodzą z L-elementowego wektora $\boldsymbol{\beta} = [\beta_1, \dots, \beta_L]$:

$$F(\boldsymbol{\beta}, \boldsymbol{x}) = \beta_1 g_1(\boldsymbol{x}) + \ldots + \beta_L g_L(\boldsymbol{x})$$
(4.7)

Niech $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_K]^T$ oznacza macierz utworzoną z rozwiązań pochodzących ze zbioru uczącego \mathcal{D} , a $\boldsymbol{y} = [y_1, \dots, y_K]^T$ oznacza odpowiadający im wektor wartości f.c. Dla dowolnych punktów \boldsymbol{x}_i oraz \boldsymbol{x}_j kowariancja pomiędzy $\boldsymbol{\epsilon}(\boldsymbol{x}_i)$ a $\boldsymbol{\epsilon}(\boldsymbol{x}_j)$ jest zdefiniowana w następujący sposób:

$$cov[\epsilon(\boldsymbol{x}_i), \epsilon(\boldsymbol{x}_j)] = \sigma^2 \boldsymbol{R}([R(\boldsymbol{x}_i, \boldsymbol{x}_j)])$$
(4.8)

gdzie \boldsymbol{R} jest macierzą korelacji o rozmiarze $K \times K$, posiadającą następującą strukturę:

$$\boldsymbol{R} = \begin{bmatrix} R(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & R(\boldsymbol{x}_1, \boldsymbol{x}_K) \\ \vdots & \ddots & \vdots \\ R(\boldsymbol{x}_K, \boldsymbol{x}_1) & \dots & R(\boldsymbol{x}_K, \boldsymbol{x}_K) \end{bmatrix}$$
(4.9)

w której $R(\boldsymbol{x}_i, \boldsymbol{x}_j)$ jest funkcją korelacji pomiędzy $\epsilon(\boldsymbol{x}_i)$ oraz $\epsilon(\boldsymbol{x}_j)$. Często używaną funkcją korelacji jest:

$$R(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp\left(-\sum_{d=1}^D \theta_d |x_{i,d} - x_{j,d}|^2\right)$$
(4.10)

gdzie $\theta_d,$ dla $d=1,\ldots D$ są hiperparametrami.

Aproksymowana wartość $\hat{f}(\boldsymbol{x}_{K+1})$ nowego punktu \boldsymbol{x}_{K+1} (w ogólności nie należącego do zbioru uczącego \mathcal{D}), zgodnie z równaniem 4.5 może być zapisano jako:

$$\hat{f}(\boldsymbol{x}_{K+1}) = \boldsymbol{\beta} + \boldsymbol{r}^{T}(\boldsymbol{x}_{K+1})\boldsymbol{R}^{-1}(\boldsymbol{y} - F\boldsymbol{\beta})$$
(4.11)

gdzie $\mathbf{r}(\mathbf{x}_{K+1})$ jest K-elementowym wektorem korelacji pomiędzy nowym punktem \mathbf{x}_{K+1} a punktami \mathbf{x}_i pochodzącymi ze zbioru uczącego \mathcal{D} , zgodnie z poniższym równaniem:

$$\boldsymbol{r}(\boldsymbol{x}_{K+1})) = [R(\boldsymbol{x}_{K+1}, \boldsymbol{x}_1), \dots, R(\boldsymbol{x}_{K+1}, \boldsymbol{x}_K)]$$
(4.12)

Wektor współczynników β jest wyznaczany zgodnie z *uogólnioną metodą najmniejszych kwadratów* [108] (ang. *Generalized Least Squares*):

$$\boldsymbol{\beta} = \left(F^T \boldsymbol{R}^{-1} F\right)^{-1} F^T \boldsymbol{R}^{-1} \boldsymbol{y}$$
(4.13)

Estymowana wariancja σ^2 jest opisana następującym równaniem:

$$\sigma^{2} = \frac{1}{K} (\boldsymbol{y} - F\boldsymbol{\beta})^{T} \boldsymbol{R}^{-1} (\boldsymbol{y} - F\boldsymbol{\beta})$$
(4.14)

Wartości hiperparametrów θ_d są wyznaczane poprzez maksymalizowanie następującej funkcji wiarygodności (ang. *likelihood function*):

$$\psi(\theta) = -\frac{1}{2} \left(K \ln \sigma^2 + \ln \det(\mathbf{R}) \right)$$
(4.15)

gdzie $det(\mathbf{R})$ jest wyznacznikiem macierzy korelacji \mathbf{R} .

Maksymalizacja funkcji wiarygodności $\psi(\theta)$ stanowi osobny problem optymalizacyjny. Do jego rozwiązania mogą posłużyć w szczególności algorytmy populacyjne.

Obszar modelowania w Krigingu

Kriging jest w swojej naturze metamodelem globalnym, ponieważ opisuje każdy punkt przestrzeni za pomocą wartości oczekiwanej (funkcja globalnego trendu) oraz niepewności (zmienność). Jest także metamodelem relatywnie skomplikowanym (patrz opis złożoności obliczeniowej poniżej), co również przemawia za modelowaniem globalnym.

Złożoność obliczeniowa Kriginga

Uczenie (estymacja parametrów) metamodelu Kriginga posiada złożoność obliczeniową $\mathcal{O}(MK^3 + DK^2)$ [136, 57], gdzie M oznacza liczbę iteracji wykonanych przez algorytm optymalizacyjny podczas estymacji wartości hiperparametrów θ_d (równanie 4.15). Predykcja wartości $\hat{f}(\boldsymbol{x}_{K+1})$ posiada złożoność $\mathcal{O}(K^2)$. Estymacja σ^2 zależy liniowo od wielkości zbioru uczącego, tzn. jej złożoność obliczeniowa wynosi $\mathcal{O}(K)$.

Przedstawione powyżej wielkości pokazują, że estymacja parametrów jest najbardziej złożonym działaniem w całym procesie wykorzystania Kriginga. Ponadto, jest ona silnie zależna (K^3) od wielkości zbioru uczącego. Całość sprawia, że Kriging staje się nieefektywny w sytuacji znaczącego wzrostu rozmiaru zbioru uczącego.

4.3.3 Sztuczne sieci neuronowe

Sztuczne sieci neuronowe [95, 24], zwane dalej SSN, również znajdują zastosowanie w modelowaniu funkcji celu. SSN są zbudowane w oparciu o perceptrony [187], które są rodzajem liniowego klasyfikatora. Wyjście perceptronu jest funkcją ważonej sumy L wejść, co zaprezentowano na poniższym równaniu:

$$\hat{f}(\boldsymbol{x}, \boldsymbol{w}) = f_a \left(w_0 + \sum_{j=1}^L w_j x_j \right)$$
(4.16)

gdzie \boldsymbol{x} jest *L*-elementowym wektorem wejść, \boldsymbol{w} odpowiadającym mu wektorem wag, w_0 jest stałą, $f_a : \mathbb{R} \to \mathbb{R}$ nieliniową funkcją aktywacji. Przykładem funkcji aktywacji f_a może być funkcja logistyczna (sigmoidalna) o następującej postaci:

$$f_a(x) = \frac{1}{1 + e^{-x}} \tag{4.17}$$

Podstawowa SSN przyjmuje formę perceptronu wielowarstwowego (ang. Multilayer Perceptron), określanego skrótowo MLP. MLP jest zbudowany z warstw perceptronów, w których wyjścia poszczególnych perceptronów stanowią wejścia perceptronów w warstwie następnej. Na wejściu podawany jest wektor \boldsymbol{x}_i o rozmiarze D pochodzący ze zbioru uczącego, co stanowi warstwę wejściową. Następnie MLP składa się z pewnej liczby warstw ukrytych, w których każda zawiera H_j neuronów, gdzie j oznacza numer warstwy ukrytej. Na rysunku 4.3 przedstawiono przykład MLP z dwoma wejściami (D = 2), jedną warstwą ukrytą, posiadającą pięć neuronów ($H_1 = 5$) oraz jednym wyjściem.



Rysunek 4.3: Przykład perceptronu wielowarstwowego z jedną warstwą ukrytą.

Uczenie MLP, posiadającego jedno wyjście sprowadza się do znalezienie takich wartości wag w_j (znajdujących się we wszystkich połączeniach między warstwami), które minimalizują przyjętą funkcję kosztu E:

$$E = \frac{1}{K} \sum_{i=1}^{K} (\hat{y}_i - y_i)$$
(4.18)

gdzie y_i jest rzeczywistą wartością f.c. odpowiadającą obserwacji \boldsymbol{x}_i , a \hat{y}_i jest wartością będącą na wyjściu ostatniej warstwy MLP.

Powszechnym algorytmem uczącym SSN jest propagacja wsteczna [191] (ang. *back-propagation*), wykorzystująca iteracyjną metodę spadku gradientu (aktualizację wag sieci w kierunku od wyjścia do wejścia). Podobnie jak w przypadku Kriginga proces uczenia sieci stanowi osobny problemem optymalizacji ciągłej.

Obszar modelowania sztucznych sieci neuronowych

SSN sprawdzają się lepiej w modelowaniu globalnym. Podobnie jak Kriging, są one relatywnie skomplikowane (duża liczba parametrów), co predestynuje je do modelowania globalnego. Co więcej, stopień zaawansowania SSN sprawia, że proces nauki wymaga relatywnie dużych zbiorów uczących, by osiągnąć zadowalające rezultaty.

Maszyny ekstremalnego uczenia

Maszyny ekstremalnego uczenia (ELM) [91] (ang. Extreme Learning Machines) są rodzajem SSN z jedną lub wieloma warstwami ukrytymi. ELM zakładają, że parametry (wagi) warstw ukrytych nie podlegają estymacji, tzn. są stałe (choć mogą być losowo wyznaczone). Stałość parametrów warstw ukrytych sprawia, że ELM jest zbliżona w swojej strukturze do RW. W przypadku ELM przekształcenie zmiennej objaśniającej $x_{i,d} \in \boldsymbol{x}_i$ jest pewną nieliniowa funkcją $h(x_{i,d})$ wynikającą z jej architektury i wartości parametrów. Stałość parametrów warstw ukrytych obniża zdolność SSN do dokładnego modelowanie zależności między zmiennymi, lecz umożliwia szybszy proces uczenia.

Skutkiem tego, oszacowanie wartości wektora wag \boldsymbol{w} może być zrealizowane za pomocą rozszerzonej MNK (równanie 4.3). Szczegółowy opis estymowania parametrów w ELM można znaleźć m.in. w pracy [90].

Złożoność obliczeniowa sztucznej sieci neuronowej

Złożoność obliczeniowa uczenia SSN jest zależna od przyjętej architektury sieci (m.in. liczby i wielkości warstw ukrytych), rozmiaru zbioru uczącego oraz iteracji zawartych w procesie uczenia. Nie jest możliwe aprioryczne określenie złożoności obliczeniowej procesu nauki \mathcal{O} dowolnej SSN. W przypadku MLP posiadającego jedną warstwę ukrytą zawierającą H_1 neuronów, pojedyncze wyznaczenie funkcji kosztu E posiada złożoność $\mathcal{O}(KH_1(D+1))$, a jest ono realizowane wielokrotnie (zgodnie z liczbą iteracji algorytmu propagacji wstecznej). Ponadto, SSN generalnie mają zdolność do przeuczania (z racji relatywnie dużej liczny parametrów i nieliniowych zależności), co implikuje konieczność przeprowadzania cyklicznej walidacji podczas procesu uczenia [147]. Stąd, stopień skomplikowania procesu uczenia SSN należy uznać za wysoki, jeśli za jej rodzaj przyjąć MLP, a za metodę estymacji parametrów propagację wsteczną.

Zastosowanie uproszczonych architektur SSN może obniżyć złożoność obliczeniową procesu nauki. W przypadku ELM złożoność obliczeniowa estymacji parametrów za po-

mocą rozszerzonej MNK wynosi $\mathcal{O}(H_1^3 + H_1^2 K + (D+1)H_1 K)$ [94].

4.3.4 Radialne funkcje bazowe

Radialne funkcje bazowe [168], zwane dalej RBF, są rodzajem funkcji, której wartość w punkcie \boldsymbol{x} zależy wyłącznie od odległości r między punktem \boldsymbol{x} a ustalonym punktem \boldsymbol{x}_0 . Przykładami RBF h są:

- Liniowa: h(r) = r
- Sześcienna: $h(r) = r^3$
- Gaussowska: $h(r)=e^{-\beta r^2},$ gdzie β jest parametrem
- Wielokwadratowa: $h(r) = \sqrt{r^2 + \beta^2}$, gdzie β jest parametrem
- Cienkiej płytki: $h(r) = r^2 \ln r$

gdzie $r = \|\boldsymbol{x} - \boldsymbol{x}_0\|.$

Modelowanie f.c z wykorzystaniem RBF polega na przybliżeniu wartości dowolnego punktu \boldsymbol{x} za pomocą liniowej kombinacji L radialnych funkcji bazowych $h_i(\boldsymbol{x})$. Zwyczajowo przyjmuje się, że L = K, tzn. liczba RBF odpowiada liczebności zbioru uczącego. Założenie to obowiązuje w dalszej części opisu. Metamodel wykorzystujący RBF jest opisana za pomocą poniższego równania:

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{K} w_i h_i(\boldsymbol{x})$$
(4.19)

gdzie $\boldsymbol{w} = [w_1, \dots, w_K]$ jest wektorem wag.

Wartości w_i należące do wektora wag \boldsymbol{w} można uzyskać za pomocą MKN (równanie 4.3), w której jako zmienną objaśniającą przyjmuje się $h_i(\boldsymbol{x})$.

Obszar modelowania radialnych funkcji bazowych

Podobnie jak w przypadku RW, sposób konstrukcji zbioru uczącego ma zasadnicze znaczenie w kontekście obszaru modelowania za pomocą RBF. Możliwe jest modelowanie f.c. zarówno globalnie [250], jak i lokalnie [252].

Złożoność obliczeniowa radialnej funkcji bazowej

Złożoność obliczeniowa estymacji parametrów RBF wynika bezpośrednio ze złożoności MKN, która wynosi $\mathcal{O}(L^2K)$. gdzie L jest liczbą parametrów, a K liczebnością zbioru uczącego. Zgodnie z założeniem, że L = K, złożoność obliczeniowa RBF wynosi $\mathcal{O}(K^3)$. Oznacza to, że złożoność obliczeniowa jest silnie zależna od wielkości zbioru uczącego, co wyklucza stosowania RBF dla dużych zbiorów uczących. Niemniej, złożoność obliczeniowa RBF pozostaje wciąż mniejsza od złożoności obliczeniowej Kriginga.

4.4 Optymalizacja bayesowska

Optymalizacja bayesowska [134], zwana dalej OB, jest sekwencyjną strategią rozwiązywania problemów ciągłych czarnoskrzynkowych. Stanowi ona wiodącą metodę rozwiązywania problemów kosztownych [134]. Popularność OB jest podyktowana jej naturalną zdolnością do balansowania między eksploracją, a eksploatacją przestrzeni przeszukiwań.

Poniżej przedstawiono zwięzły opis OB, pozwalający zrozumieć jej zasadę działania oraz towarzyszący temu nakład obliczeniowy. Więcej o OB oraz jej zastosowaniach można przeczytać w pracach [150, 64, 233].

OB zakłada wykorzystanie metamodelu, który jest estymowany w oparciu o obserwacje zebrane w zbiorze uczącym \mathcal{D} (równanie 4.1). Każda ewaluacja f.c. (w punkcie \boldsymbol{x}_{K+1}) skutkuje dodaniem nowej obserwacji ($\boldsymbol{x}_{K+1}, y_{K+1}$) do zbioru uczącego, w której $y_{K+1} = f(\boldsymbol{x}_{K+1})$. Stąd, rozmiar zbioru uczącego \mathcal{D} zwiększa się wraz z każdą ewaluacją f.c. o jeden.

Wyznaczenie punktu \boldsymbol{x}_{K+1} odbywa się z wykorzystaniem funkcji akwizycji, która W dużym uproszczeniu reprezentuje użyteczność potencjalnej ewaluacji dowolnego punktu \boldsymbol{x} należącego do przestrzeni przeszukiwań. Użyteczność jest wyznaczana w oparciu o kombinację prognozowanej wartości $\hat{f}(\boldsymbol{x})$ oraz pewnej miary niepewności z tym związanej. Konieczność jednoczesnego modelowania wartości f.c. oraz towarzyszącej temu niepewności oszacowania sprawia, że OB rozszerza ideę Kriginga.

Jedna z często wykorzystywanych funkcji akwizycji jest oparta o górną granicę ufności [227, 209] (ang. Upper Confidence Bound). Jej postać przedstawia poniższe równanie:

$$f_{ak}(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \lambda \sigma(\boldsymbol{x}) \tag{4.20}$$

gdzie $\mu(\boldsymbol{x})$ jest wartością prognozowaną (funkcją globalnego trendu z równania 4.5), λ jest
parametrem określającym górną granicę ufności, a $\sigma(\boldsymbol{x})$ oznacza niepewność zależną od położenia \boldsymbol{x} . Szczegółowy sposób wyznaczania $\sigma(\boldsymbol{x})$ został opisany w pracy [209].

Zwiększenie wartości funkcji akwizycji dla punktów o mniejszej wartości $\hat{f}(\boldsymbol{x})$ jest rozumiane jako skłonność do eksploatacji. Z kolei, zwiększenie wartości funkcji akwizycji dla punktów o większej niepewności jest czynnikiem promującym eksplorację. Bez względu na przyjętą postać funkcji akwizycji wyznaczenie punktu \boldsymbol{x}_{K+1} stanowi osobny problem optymalizacji w postaci:

$$\boldsymbol{x}_{K+1} = \underset{\boldsymbol{x} \in S}{\operatorname{arg\,min}} - f_{ak}(\boldsymbol{x}) \tag{4.21}$$

Algorytmy populacyjne również znajdują zastosowanie w rozwiązywaniu wyżej przedstawionego problemu. Możliwość użycia metody niedeterministycznej do wyznaczenia punktu \boldsymbol{x}_{K+1} sprawia, że OB jest w ogólności metodą niedeterministyczną. Mimo to, jasno określone kryteria wyboru kolejnych punktów do ewaluacji sprawiają, że cały proces optymalizacji zyskuje pewne uzasadnienie, co jest kluczowe w sytuacji, gdy ewaluacja f.c. jest kosztowna. Wysokopoziomowe spojrzenie na OB zostało przedstawione za pomocą pseudokodu 4.

Algoryt	m 4 Wysokopoziomowy pseudokod Optymalizacji Bayesowskiej
1: $\mathcal{D} = \{$	}
2: $K = 0$	
3: for <i>i</i> =	$=1,2,\ldots\mathbf{do}$
4: Wy	yznacz nowy punkt \boldsymbol{x}_{K+1} (zgodnie z równaniem 4.21)
5: Do	konaj ewaluacji punktu $oldsymbol{x}_{K+1}$

- 6: Dodaj nową obserwację do zbioru uczącego ($\mathcal{D} = \{\mathcal{D}, (\boldsymbol{x}_{K+1}, y_{K+1})\}, \text{ gdzie } y_{K+1} = f(\boldsymbol{x}_{K+1})$)
- 7: Aktualizuj parametry metamodelu (uczenie metamodelu na podstawie \mathcal{D})
- 8: end for

Efektywna globalna optymalizacja

Powszechnym algorytmem opartym o OB jest efektywna globalna optymalizacja [104] (ang. Efficient Global Optimization), zwana dalej EGO, która za funkcję akwizycji przyjęła oczekiwaną poprawę (ang. Expected Improvement). Algorytm EGO definiuje dodatkowo reguły inicjalnego próbkowania (wyznaczenia inicjalnego zbioru uczącego \mathcal{D} , dbając o odpowiednie rozmieszczenie punktów w przestrzeni). Innymi słowy, pewna liczba początkowych punktów jest ewaluowana z wykorzystaniem f.c. przed uruchomieniem pierwszego wyznaczenia punktu w oparciu o funkcję akwizycji. O szczegółach algorytmu EGO można przeczytać w pracy [104].

Złożoność obliczeniowa optymalizacji Bayesowskiej

OB jest skuteczną strategię optymalizacyjną dla problemów z ograniczonym budżetem optymalizacji. Niemniej, konieczność modelowania funkcji w sposób globalny z wykorzystaniem wszystkich dostępnych informacji (obserwacji w zbiorze uczącym) sprawia, że wraz z przebiegiem optymalizacji jej efektywność znacząco spada. Złożoności obliczeniowa OB jest pochodną złożoności obliczeniowej Kriginga (z dokładnością do jego wariantu i możliwych modyfikacji), powiększoną o złożoność algorytmu rozwiązującego problem maksymalizacji funkcji akwizycji. Stąd, nawet dla małych zbiorów uczących (początek procesu optymalizacji) ta strategia jest relatywnie złożona, ponieważ wyznaczenie każdego kolejnego punktu \boldsymbol{x}_{K+1} stanowi rozwiązanie problemu opisanego równaniem 4.21. W miarę kolejnych ewaluacji f.c. jej efektywność zaczyna drastycznie spadać, ponieważ wymagany nakład obliczeniowy na estymację parametrów metamodelu zależy od K^3 .

4.5 Algorytmy populacyjne wspierane metamodelem (APWM)

Konstrukcja algorytmów populacyjnych sprawia, że nie znalazły one bezpośredniego zastosowania w rozwiązywaniu problemów kosztownych. Przede wszystkim, algorytmy populacyjne należą do metod niedeterministycznych (rozdział 3), co w połączeniu z mocno ograniczoną liczbą dostępnych ewaluacji f.c. sprawia, że otrzymanemu rozwiązaniu może towarzyszyć duży błąd. Ponadto, zasady wyznaczania kolejnych punktów do ewaluacji w algorytmach populacyjnych nie są bezpośrednio motywowane chęcią równoważenia eksploracji oraz eksploatacji, co jeszcze bardziej zmniejsza zaufanie do uzyskanego wyniku.

Mimo to, wysoka wydajność algorytmów populacyjnych w rozwiązywaniu problemów optymalizacji taniej sprawiła, że zaczęto poszukiwać sposobów na ich adaptację do rozwiązywania problemów kosztownych. Wspieranie algorytmów populacyjnych metamodelami pozwala na zmniejszenie negatywnych skutków ich niedeterminizmu, ponieważ każde rozwiązanie zyskuje pewną interpretację, w postaci przybliżonej wartości f.c.

Skutkiem tego powstało wiele APWM z powodzeniem konkurujących z klasyczną OB. Mimo, że główną motywacją dla projektowania takich rozwiązań było rozwiązywanie problemów kosztownych, to niektóre metody znalazły zastosowanie w optymalizacji taniej lub semikosztownej.

Zastosowanie metamodeli w algorytmach populacyjnych nie jest nowym pomysłem. Pierwsze APWM były prezentowane już pod koniec lat 80 [60]. W latach 90. nastąpił wzrost zainteresowania APWM, czego skutkiem były udane próby integracji metamodeli takich jak Kriging [183] czy SSN [195, 32]. Z czasem wzrost zainteresowania APWM tylko rósł, czego dowodem są liczne prace przeglądowe z lat: 2005 [100], 2010 [199], 2011 [101], 2013 [136], 2018 [103] oraz 2022 [126].

Poniżej opisano metody integracji metamodeli w algorytmach populacyjnych, a następnie opisano wybrane metody wykorzystujące metamodel należące do rodziny DE oraz CMA-ES. Dodatkowo opisano przykładowe rozwiązania oparte o inne algorytmy populacyjne. Całość ma za zadanie przedstawić wysokopoziomowe spojrzenie na użycie metamodeli w algorytmach populacyjnych, po to by w rozdziale 5 podjąć dyskusję o sposobach ich efektywnego zastosowania.

4.5.1 Metody integracji metamodeli

Przegląd literatury poświęconej APWM pozwala zauważyć, że metamodele mogą znaleźć zastosowanie w modyfikacji większości mechanizmów wykorzystywanych przez algorytmy populacyjne [100]. Stąd, trudno o wskazanie jednej uniwersalnej klasyfikacji metod integracji metamodeli. Poniżej przedstawiono trzy znane klasyfikacje, które pozwalają spojrzeć z różnych stron na sposoby aplikacji metamodeli w algorytmach populacyjnych.

Podstawowa klasyfikacja zakłada podział na metamodele globalne oraz lokalne (rozdział 4.2). W szczególności, oznacza to różne sposoby konstruowania zbiorów uczących. Metamodele globalne na ogół wykorzystują jeden zbiór, w którym obserwacje pochodzą z całej przestrzeni rozwiązań. Inaczej jest w przypadku metamodeli lokalnych, w których zbiór uczący jest ograniczony do pewnego małego obszaru przestrzeni rozwiązań. Co więcej, na ogół metamodel lokalny służy do przybliżania różnych obszarów przestrzeni rozwiązań, a każdy z tych obszarów ma zdefiniowany swój własny zbiór uczący. Warto dodać, że istnieją algorytmy wykorzystujące metamodele złożone [69, 121] (ang. *ensemble metamodels*), czyli zespół różnych metamodeli globalnych lub lokalnych.

Inna klasyfikacja zakłada podział metod integracji metamodeli ze względu na zasięg wykorzystywania metamodelu w algorytmie populacyjnym [101]. Możliwe jest wykorzystanie metamodeli w obrębie:

- 1. Osobnika [29, 123, 170, 178].
- 2. Iteracji [32, 129].
- 3. Populacji [54, 196].

Wykorzystanie metamodelu w obrębie osobnika służy ocenie czy warto ewaluować danego osobnika, ponieważ reprezentuje on obiecujące rozwiązania czy lepiej oszczędzić budżet optymalizacji na potrzeby bardziej obiecujących rozwiązań. Wykorzystanie metamodelu w obrębie iteracji zakłada, że metamodel może w pełni zastępować f.c. w wybranych iteracjach algorytmu populacyjnego. Ideą tego sposobu jest wykonanie pewnej liczby iteracji w oparciu o metamodel, po to by umożliwić zbieganie populacji w kierunku obiecujących rozwiązań, bez wykonywania czasochłonnych ewaluacji f.c. Ostatni sposób, czyli wykorzystanie metamodeli w obrębie całej populacji ma zastosowanie w przypadku, gdy algorytm zakłada wykorzystanie wielu populacji i każda z tych populacji wykorzystuje inny metamodel lub nie wykorzystuje go wcale.

Kolejna klasyfikacja metod integracji metamodeli [136] jest zbudowana w oparciu o techniczny sposób implementacji metamodelu w algorytmie populacyjnym. Metamodel może:

- 1. Dokonywać preselekcji obiecujących rozwiązań [56, 222, 77, 242, 241, 243].
- 2. Modyfikować mechanizmy działania algorytmu [182, 10, 4].
- 3. Zastępować funkcję celu [102, 18, 174].

W sposobie pierwszym, metamodel służy wyznaczaniu zbioru rozwiązań poddawanych późniejszej ewaluacji z wykorzystaniem f.c. Istnieją dwa wiodące warianty realizujące to zadanie. Pierwszy z nich zakłada generowanie większej liczby kandydatów (potencjalnych rozwiązań) niż wskazuje bieżący rozmiar populacji, a następnie ewaluację z wykorzystaniem f.c. wyłącznie tych najbardziej obiecujących z punktu widzenia wartości metamodelu. Drugi wariant zakłada utrzymanie bieżącego rozmiaru populacji, lecz dopuszcza odstąpienie od ewaluacji z wykorzystaniem f.c. niektórych rozwiązań i zastąpienie ich wartościami metamodelu.

Istotą drugiego sposobu jest modyfikowanie działania algorytmu w taki sposób, by mechanizm generowania rozwiązań uwzględniał estymacje metamodelu. W założeniu, ma

to spowodować generowanie obiecujących rozwiązań, które będą poddane ewaluacji z wykorzystaniem f.c. Modyfikacjom mogą podlegać m.in. operatory mutacji lub krzyżowania [182, 4, 10].

Sposób trzeci opiera się na założeniu, że proces optymalizacji może być przeprowadzany do pewnego stopnia z wykorzystaniem metamodelu, a nie f.c. Ten sposób najmniej ingeruje w logikę algorytmu, ponieważ sprowadza się do zastąpienia ewaluacji f.c. obliczeniem wartości metamodelu. Niemniej, stopień zastępowania f.c. za pomocą metamodelu jest właściwy dla konkretnej metody i reguły za tym stojące mogą stanowić osobny relatywnie złożony algorytm.

4.5.2 Metamodele w algorytmach z rodziny DE

Przeważającą grupą metamodeli stosowanych w algorytmach z rodziny DE są metamodele Kriginga. Przykładem tego jest GPEME [131], który zakłada, że Kriging służy do preselekcji rozwiązań generowanych przez DE, celem ewaluacji tylko tych najbardziej obiecujących. Podobnie, algorytm Sa-DE-DPS [55] jest hybrydą DE oraz Kriginga. W Sa-DE-DPS rola DE sprowadza się do wyznaczenia kolejnych punktów do ewaluacji z wykorzystaniem f.c., a Kriging ma za zadanie modelować f.c. i dostarczyć postać funkcji akwizycji. Dlatego też, Sa-DE-DPS jest realizacją strategii OB.

Inne metamodele, takie jak RW oraz regresja wektorów nośnych również mogą zwiększać skuteczność DE w domenie optymalizacji kosztownej [51].

Przykładem wykorzystania metamodeli złożonych w DE jest DE-ELS [117], czyli DE rozszerzone o preselekcję rozwiązań z wykorzystaniem metamodelu Kriginga, RW oraz *k*-najbliższych sąsiadów. JADE również może korzystać z metamodeli złożonych. Przykładem tego jest S-JADE [33] przeznaczony do problemów kosztownych z dużą liczbą wymiarów. S-JADE został zintegrowany z globalnym metamodelem opartym o RBF oraz wieloma lokalnymi metamodelami, również opartymi o RBF. Metamodel globalny oraz metamodele lokalne służą modyfikacji fazy mutacji, celem kierowania nowych osobników do obiecujących obszarów.

Warto zwrócić szczególną uwagę na trzy algorytmy należące do rodziny DE wykorzystujące Kriginga: SMA-EPSDE [143] oparty na DE, ESMDE [144] oparty na DE oraz iDEaSm [16] oparty na L-SHADE. We wszystkich ww. algorytmach metamodel Kriginga jest estymowany raz na iterację i wyłącznie w oparciu o zbiór rozwiązań tworzony przez bieżącą populację. Ograniczony rozmiar populacji sprawia, że taka forma integracji Kriginga pozwala na użycie algorytmu w budżetach optymalizacji taniej. W algorytmie SMA-EPSDE metamodel ocenia czy wygenerowany wektor próby będzie lepszy od wektora macierzystego, i jeśli tak, to jest on poddawany ewaluacji z wykorzystaniem f.c. Generowanych jest wiele wektorów próby w oparciu o wiele wariantów mutacji oraz krzyżowania. ESMDE oraz iDEaSm w zbliżony sposób wykorzystują metamodel do preselekcji generowanych wektorów próby. Algorytm iDEaSm można uznać za rozszerzenie SMA-EPSDE oraz ESMDE, ponieważ za algorytm populacyjny przyjęto L-SHADE, będący rozwinięciem DE.

4.5.3 Metamodele w algorytmach z rodziny CMA-ES

Algorytmy z rodziny CMA-ES również z powodzeniem mogą być rozszerzane o wykorzystanie metamodeli. Wykorzystanie ważonej kwadratowej RW z interakcjami w modelowaniu wartości f.c. w otoczeniu danego punktu znalazło zastosowanie w algorytmie lmm-CMA-ES [112]. Mimo, że zadaniem algorytmu jest rozwiązywanie problemów kosztownych, to autorzy pracy zwrócili uwagę na problem złożoności obliczeniowej proponowanej metody. Szczególnie problematyczne okazało się uwzględnienie interakcji w modelu regresji.

Wykorzystanie kwadratowej RW może również znaleźć zastosowanie w modyfikacji mechanizmów CMA-ES, takich jak aktualizacja macierzy kowariancji [14].

Przykładem wykorzystania metamodelu Kriginga w CMA-ES jest S-CMA-ES [18], który w obrębie całej iteracji może zastępować ewaluacje f.c. za pomocą przybliżeń metamodelu. Archiwum punktów, które zostały poddane ewaluacji nie jest ograniczone rozmiarem. Jednakże sam zbiór uczący wykorzystuje obserwacje będące w pewnym otoczeniu wartości oczekiwanej rozkładu populacji, a jego rozmiar jest ograniczony do pewnej stałej wartości. Dlatego też, S-CMA-ES, mimo że jest przeznaczony do optymalizacji problemów kosztownych, może z powodzeniem osiągać budżety semikosztowne, co potwierdzają jego wyniki eksperymentalne. Algorytm DTS-CMA-ES [174] rozszerza S-CMA-ES o dodatkowe uczenie modeli w każdej iteracji po wykonaniu ewaluacji kilku rozwiązań.

Możliwe jest także połączenie EGO oraz CMA-ES w taki sposób, że optymalizacja inicjowana jest za pomocą EGO i dopiero po pewnej liczbie iteracji następuje przełączenie na optymalizację z wykorzystaniem CMA-ES [154]. Proponowana hybryda osiągnęła lepsze rezultaty w rozwiązywaniu problemów kosztownych niż EGO uruchomione bez przełączenia na CMA-ES.

Szczegółowy przegląd oraz porównanie algorytmów z rodziny CMA-ES wykorzystujących metamodel Kriginga można znaleźć w pracy [19]. CMA-ES został również rozszerzony o metamodel wykorzystujący rankingową maszynę wektorów nośnych, skutkując powstaniem algorytmu saACM-ES [139, 140]. W zaproponowanym podejściu CMA-ES naprzemiennie wykonuje jedną iteracją z wykorzystaniem f.c. oraz pewną liczbę iteracji z wykorzystaniem metamodelu. Metamodel uczony jest po każdej iteracji wykonanej z udziałem ewaluacji f.c. Zbiór uczący budowany jest w oparciu o archiwum rozwiązań poddanych wcześniejszej ewaluacji, lecz jest ograniczony do pewnej liczby najnowszych obserwacji. Sugerowany rozmiar zbioru uczącego został wyznaczony empirycznie. Sposób integracji metamodelu oraz ograniczony rozmiar zbioru uczącego sprawiły, że saACM-ES był ewaluowany w budżetach optymalizacji taniej.

Algorytm lq-CMA-ES

Szczególną uwagę należy poświęcić algorytmowi lq-CMA-ES [77], który jest rozszerzeniem CMA-ES o globalny metamodel dokonujący preselekcji rozwiązań. Za metamodel przyjęto ważoną kwadratową RW z uwzględnieniem interakcji między zmiennymi.

Metamodel jest estymowany w oparciu o zbiór uczący składający się z ostatnich $max(N^g, 2df_{max})$ rozwiązań poddanych ewaluacji z wykorzystaniem f.c., gdzie N^g jest wielkością populacji, a $df_{max} = (D^2 + 3D)/2 + 1$ jest liczbą stopni swobody metamodelu.

Idea wykorzystania metamodelu sprowadza się do tego, by w każdej iteracji dokonać ewaluacji przy użyciu f.c. f pewnej liczby osobników (rozwiązań) i zobaczyć czy estymacje metamodelu \hat{f} są obarczone małym błędem. Jeżeli błąd jest dostatecznie mały, to dla pozostałych osobników z populacji ewaluacja z wykorzystaniem f.c. jest zastępowana wartością metamodelu \hat{f} . Jeżeli błąd nie jest akceptowanie mały, to kolejne osobniki są poddawane ewaluacji z wykorzystaniem f.c., a metamodel jest na nowo uczony i sprawdzany jest jego błąd. Jako miarę błędu przyjęto korelację τ -Kendalla, a za jej próg 0.85.

Jednym z dodatkowych mechanizmów lq-CMA-ES jest kaskada metamodeli. W przypadku, gdy liczba obserwacji zgromadzonych w zbiorze uczącym jest niewystarczająca do estymacji parametrów kwadratowej RW z uwzględnieniem interakcji, wykorzystuje się kwadratową RW bez interakcji. W przypadku, gdy liczba obserwacji wciąż pozostaje niewystarczająca, używana jest liniowa RW.

Estymacja parametrów metamodelu w lq-CMA-ES jest oparta o dodatkowe założenia. Przede wszystkim tylko 75% obserwacji ze zbioru uczącego \mathcal{D} trafia do metamodeli. Następnie, wagi obserwacji uwzględnionych w MNK są wyznaczone arbitralnie i zmniejszają się liniowo od 20 do 1, gdzie 20 jest dla obserwacji najlepszej (rozwiązania \boldsymbol{x}_i o najmniejszej wartości $f(\boldsymbol{x}_i)$), a 1 dla obserwacji najgorszej. Co więcej, wartość korelacji τ -Kendalla jest liczona w oparciu o $max(15, min(1.2 \cdot n_{eval}, 0.75 \cdot N^g))$, gdzie n_{eval} jest liczbą rozwiązań poddanych ewaluacji w bieżącej iteracji. Szczegółowy opis metody oraz wszystkich założeń można znaleźć w pracy źródłowej [77].

4.5.4 Metamodele w innych algorytmach populacyjnych

PSO, jako przedstawiciel algorytmów rojowych, również zyskał wiele rozszerzeń wykorzystujących metamodele. W algorytmie FSAPSO [124] zastosowano globalne modelowanie f.c. z wykorzystaniem RBF. Metamodel jest estymowany w oparciu o całą dostępną historię ewaluacji f.c., a jego zadaniem jest preselekcja obiecujących osobników w populacji. Dodatkowo, część rozwiązań jest poddawana ewaluacji w oparciu o niepewność ich estymowanej wartości. Niepewność jest rozumiana jako najmniejsza odległość od dowolnego rozwiązania znajdującego się w zbiorze uczącym.

RBF może być również użyta w PSO jako wsparcie dla procesu restartowania populacji poprzez wpływ na rozmieszczenie nowo generowanych osobników w przestrzeni przeszukiwań [240].

Preselekcja rozwiązań za pomocą RBF znajduje też zastosowania w przypadku, gdy PSO wykorzystuje więcej niż jedną populację [122]. Inne przykłady rozwinięcia PSO o preselekcję rozwiązań z wykorzystaniem RBF można znaleźć w pracach [184, 178]

Możliwe jest łączenie PSO z metamodelem Kriginga. W algorytmie sPSO [172] rozwiązania otrzymane wskutek minimalizacji funkcji akwizycji mogą stanowić punkty przyciągania dla wszystkich osobników należących do roju. Wartym zauważenia mechanizmem jest parametryzowalna częstość wykorzystywania metamodelu Kriginga wynosząca raz na zadaną liczbę iteracji. Motywacją jest częstsze wykorzystanie Kriginga w przypadku funkcji wysoce kosztownych i rzadsze w przypadku, gdy koszt ewaluacji f.c. nie jest uznawany za bardzo duży. Tworzenie nowych punktów przyciągania w PSO może być tez realizowane z wykorzystaniem RW [218].

Algorytmy populacyjne inne niż PSO również są rozszerzane o wykorzystani metamodeli, czego przykładem jest GA [167] oraz liczne algorytmy inspirowane naturą [247, 127, 132, 220]. Więcej przykładów APWM można znaleźć w pracach przeglądowych [100, 199, 101, 136, 103, 126].

Rozdział 5

Zastosowania metamodeli w algorytmach populacyjnych

W tym rozdziale omówiono problem efektywnego zastosowania metamodelu w algorytmie populacyjnym. Przedstawiono szczegółowe kryteria zastosowania metamodelu, które pozwalają dane zastosowanie uznać za efektywne. Zaprezentowano oraz omówiono eksperymentalne badanie czasu estymacji parametrów metamodeli takich jak: Kriging, metamodel oparty o radialną funkcję bazową, kwadratowa regresja wielomianowa oraz kwadratowa regresja wielomianowa z interakcjami. Następnie przeprowadzono dyskusję efektywności znanych metod integracji metamodeli, odnosząc się do wprowadzonych kryteriów efektywności oraz wyników eksperymentalnego badania czasu estymacji parametrów metamodeli.

Finalnie, zaproponowano efektywne zastosowania metamodeli takie jak: inicjalizacja metamodelem, lokalna optymalizacja metamodelem, preselekcja rozwiązań na podstawie wartości metamodelu oraz rekurencyjna estymacja parametrów metamodelu. Implementacje zaprezentowanych efektywnych zastosowań metamodeli w algorytmach populacyjnych zostały opisane w kolejnych dwóch rozdziałach.

5.1 Obecnie stosowane metody

Funkcja celu może być przybliżana za pomocą wielu metamodeli (rozdział 4). Każdy z metamodeli posiada swoją charakterystykę, która wpływa na zdolność do skutecznego odwzorowywania f.c., ale także ma swój udział w złożoności obliczeniowej. Teoretyczne właściwości metamodeli są istotne, lecz jak pokazano w rozdziale 4.5 sposoby ich integracji odgrywają nie mniejszą rolę w kontekście oceny szeroko rozumianej efektywności.

ROZDZIAŁ 5. ZASTOSOWANIA METAMODELI W ALGORYTMACH POPULACYJNYCH

Efektywne zastosowania metamodeli, będące tematem tej rozprawy, mogą być rozumiane na wiele sposobów. W rozprawie założono że, by dane zastosowanie metamodelu w algorytmie populacyjnym uznać za efektywne, muszą być spełnione następujące kryteria:

- K1. Rodzaj metamodelu oraz sposób jego integracji pozwala na wykorzystanie proponowanego APWM w optymalizacji semikosztownej lub taniej.
- K2. Narzut obliczeniowy wynikający z użycia metamodelu jest akceptowalny.
- K3. Proponowany APWM osiąga zadowalające wyniki.
- K4. Proponowany AWPM jest metodą możliwie uniwersalną.

Kryterium K1 wymaga możliwości zastosowania proponowanego APWM w rozwiązywaniu problemów optymalizacji semikosztownej lub taniej. Metody oparte o OB znalazły zastosowanie w rozwiązywaniu problemów kosztownych, lecz mają problemy ze skalowalnością do większych budżetów optymalizacji. Ponadto, większość APWM (rozdział 4.5) również jest przeznaczona do rozwiązywania problemów kosztownych. Stąd, by zastosowanie metamodelu uznać za efektywne, musi ono być na tyle wydajne, by pozwolić na osiągnięcie budżetów optymalizacji semikosztownej lub taniej.

Kryterium K2 jest rozszerzeniem kryterium K1 o miary ilościowe. Zastosowanie metamodeli w APWM będzie skutkowało zwiększeniem czasu działania algorytmu w odniesieniu do bazowego algorytmu populacyjnego. Wpływ integracji metamodelu na czas działania algorytmu zależy od przyjętego budżetu optymalizacji. W ogólności, nie jest to zależność liniowa. Stąd, oczekuje się by narzut obliczeniowy, skutkujący dłuższym wykonaniem obliczeń całego algorytmu pozostawał na akceptowalnym poziomie. Szerszą analizę narzutu obliczeniowego metamodelu oraz towarzyszące temu badanie eksperymentalne przedstawiono poniżej, w rozdziale 5.1.1.

Kryterium K3 jest spełnione, gdy APWM osiąga zadowalające wyniki. Nie istnieje obiektywna definicja zadowalających wyników, lecz w rozprawie przyjęto, że ewaluacja algorytmu powinna być wykonana przy użyciu jednego z popularnych zbiorów testowych (rozdział 2.4). Wyniki należy uznać za zadowalające w sytuacji, gdy ewaluacja potwierdzi przewagę proponowanego APWM względem innego APWM lub względem bazowego algorytmu populacyjnego bez wsparcia metamodelu. Ponadto, zakłada się, że bazowy algorytm populacyjny bez wsparcia metamodelu będzie powszechnie uznawany za skuteczną metodę rozwiązującą. Dlatego też, za bazę efektywnego APWM uznaje się adaptacyjne algorytmy z rodziny DE lub algorytmy z rodziny CMA-ES. Kryterium K4 jest spełnione, gdy proponowany APWM jest możliwie uniwersalną metodą rozwiązującą problemy optymalizacji. Uniwersalność jest rozumiana jako zdolność do uzyskiwania zadowalających wyników w rozwiązywaniu szerokiej klasy problemów czarnoskrzynkowych. Wykorzystanie podczas ewaluacji zbiorów testowych zawierających zróżnicowane funkcje częściowo odpowiada na potrzebę potwierdzenia uniwersalności proponowanego rozwiązania. Mimo to, ocenie podlega również stopień parametryzacji proponowanego algorytmu. Większa liczba parametrów, w tym także tych związanych z metamodelem i jego integracją, pozwala na strojenie algorytmu celem poprawy wyników ewaluacji. Z drugiej strony, jeśli celem jest opracowanie metody uniwersalnej, to metody z mniejszą liczbą parametrów są preferowane.

5.1.1 Narzut obliczeniowy metamodelu

Rozwiązywanie problemów optymalizacyjnych wymaga określenia budżetu optymalizacji, czyli dostępnej liczby ewaluacji f.c. Strategie wyznaczenia dostępnej liczby ewaluacji f.c. są różne w zależności od typu optymalizacji (kosztowna, semikosztowna, tania).

Głównym wyzwaniem w projektowaniu metod rozwiązujących problemy czarnoskrzynkowe jest to, że akceptowalny poziom kosztu (czasu) całego procesu optymalizacji wynika z subiektywnych założeń. W przypadku optymalizacji semikosztownej wykonanie 10⁴ ewaluacji f.c. w kilka godzin może być zupełnie akceptowalne, natomiast w przypadku optymalizacji taniej wykonanie 10⁶ ewaluacji f.c. w ciągu minuty może uchodzić za zbyt wolne. Niemniej, określenie budżetu optymalizacji jest wypadkową kosztu pojedynczej ewaluacji f.c. oraz czasu działania metody rozwiązującej.

W optymalizacji kosztownej dominujące kryterium stanowi koszt pojedynczej ewaluacji f.c. Koszt obliczeń metody rozwiązującej jest pomijalne mały w porównaniu z kosztownością ewaluacji f.c. Stąd, OB jest uznawana za wiodącą strategię rozwiązywania tych problemów, ponieważ jej relatywnie duża złożoność obliczeniowa nie stanowi ograniczenia dla małych budżetów optymalizacji. W miarę jak koszt pojedynczej ewaluacji f.c. maleje, większą wagę przypisuje się do kosztu obliczeniowego metody rozwiązującej.

Zastosowanie metamodelu w algorytmie populacyjnym skutkuje powstaniem pewnego narzutu obliczeniowego. Poniższe równanie przedstawia ogólne spojrzenie na całkowity czas T_c , jaki zajmuje proces optymalizacji w zależności od przyjętego budżetu optymalizacji B:

$$T_c(B) = B \cdot T_e + \sum_{b=1}^{B} T_{ap}(b) + \sum_{b=1}^{B} T_{mm}(b)$$
(5.1)

gdzie b oznacza numer porządkowy ewaluacji f.c., T_e oznacza czas pojedynczej ewaluacji f.c., $T_{ap}(b)$ jest czasem obliczeń bazowego algorytmu populacyjnego (bez metamodelu) przypadający na b-tą ewaluację f.c., a $T_{mm}(b)$ dodatkowym czasem (narzutem czasowym) wynikającym z zastosowania metamodelu, jaki przypada na b-tą ewaluację f.c.

Czas T_{ap} zależy od b, ponieważ w ogólności struktura algorytmu populacyjnego może zmieniać się w czasie. Przykładem tego jest algorytm L-SHADE, który zakłada liniową redukcję rozmiaru populacji. Wskutek tego średni czas obliczeń przypadający na pojedynczą ewaluację f.c. rośnie, ponieważ wykonaniu każdej iteracji towarzyszy pewien stały koszt obliczeniowy przypadając na zmniejszającą się liczbę ewaluacji f.c.

Czas T_{mm} podobnie zależy od *b*, czego przykładem jest OB, w której zbiór uczący zwiększa się w każdej iteracji. W przeciwieństwie do T_{ap} , dla T_{mm} może zachodzić $T_{mm}(1) \ll T_{mm}(B)$. Zakładając stały budżet optymalizacji *B* powyższe równanie 5.1 można uprościć do:

$$T_c = T_e + T_{ap} + T_{mm} \tag{5.2}$$

gdzie T_c oznacza całkowity czas procesu optymalizacji, T_e jest łącznym czasem ewaluacji f.c., T_{ap} jest łącznym czasem obliczeń algorytmu populacyjnego (zakładając brak wsparcia metamodelu), a T_{mm} jest dodatkowym czasem wynikającym z integracji metamodelu w algorytmie populacyjnym.

Narzut obliczeniowy metamodelu, rozumiany jako T_{mm} w równaniu 5.2, wynika przede wszystkim z czasu wymaganego na uczenie metamodelu, czyli estymację jego parametrów. W drugiej kolejności znaczenie ma częstość wykonywania procesu uczenia, która może być dokonywana w szczególności po każdej ewaluacji f.c. lub po każdej iteracji. Istotną rolę odgrywa także liczba estymacji $\hat{f}(\boldsymbol{x})$ jaka jest wykonywana przez metamodel podczas działania APWM. Oprócz tego, wpływ na czas obliczeń ma zmiana logiki algorytmu populacyjnego wskutek integracji metamodelu, np. poprzez zwiększenie liczby wykonywanych iteracji lub generowanie większej liczby osobników niż określa to wielkość populacji.

Niemniej, kluczowym czynnikiem określającym narzut obliczeniowy metamodelu jest czas estymacji jego parametrów, który jest właściwy dla danego rodzaju metamodelu.

5.1.2 Eksperymentalne badanie czasu estymacji parametrów metamodeli

W pracach omawiających APWM nie ma spójnej metodologii badania narzutu obliczeniowego metamodeli. Najczęściej spotykane analizy wydajności metamodeli sprowadzają się do opisania złożoności obliczeniowej metamodeli \mathcal{O} , badania całkowitego czasu T_c ewaluowanej metody lub dyskusji na temat złożoności metamodelu i wynikających z tego problemów wydajnościowych.

Warto zauważyć, że zbiór testowy CEC zawiera procedurę badania empirycznej złożoności obliczeniowej algorytmu (rozdział 2.4.3), która pozwala wyznaczyć sumę czasów $T_{ap} + T_{mm}$. Jednakże, wykorzystanie zbioru testowego CEC nie stanowi dominującej metody ewaluowania APWM. Ponadto, procedura badania empirycznej złożoności obliczeniowej w CEC nie pozwala na stwierdzenie jaki jest udział obliczeń związanych z metamodelem w sumie wszystkich obliczeń APWM.

Skutkiem powyższych obserwacji autor rozprawy wykonał eksperymentalne badanie czasu estymacji parametrów metamodelu. Uwzględniono Kriginga (za model regresji opisany równaniem 4.7 przyjęto liniową RW), RBF $(h(r) = r^3)$, kwadratową RW oraz kwadratową RW z interakcjami. Czas estymacji parametrów mierzono w relacji do wielkości zbioru uczącego oraz wymiarowości problemu. Badanie wykonano w środowisku *MATLAB R2022a* działającym na systemie Windows 10 z wykorzystaniem sprzętu o następujących parametrach: CPU: Intel Core i7-4700MQ (2.40Ghz), RAM: 16GB.

Zbiór uczący $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$, gdzie $i = 1, \ldots, K$ został wygenerowany (pseudo)losowo. Rozwiązania $\boldsymbol{x}_i \in [0, 1]^D$ uzyskano za pomocą generatora *Latin Hypercube Sampling* [84], a odpowiadające im wartości f.c. y_i wylosowano zgodnie z rozkładem normalnym $\mathcal{N}(0, 1)$. Zmierzono średni czas estymacji parametrów metamodelu na podstawie zbioru uczącego \mathcal{D} . W przypadku RW, tworzenie wektora zmiennych objaśniających $\tilde{\boldsymbol{x}}$ (rozdział 4.3.1) zostało włączone do pomiaru czasu estymacji parametrów. Podobnie uczyniono z mierzeniem wydajności metamodelu opartego o RBF, tzn. mierzono całkowity czas estymacji parametrów, włączając w to wyznaczanie odległości między punktami oraz kalkulację h(r). Parametry metamodelu Kriginga zostały wyznaczone za pomocą pakietu *DACE* [135] dedykowanego dla *MATLAB*. Konsekwentnie, mierzono całkowity czas estymacji parametrów, tj. od podania zbioru uczącego \mathcal{D} do otrzymania parametrów metamodelu.

W tabeli 5.1 przedstawiono porównanie czasów estymacji parametrów metamodeli (Mm.) ze względu na rozmiar $K \in \{50, 100, 500, 1000, 2000, 4000\}$ zbioru uczącego \mathcal{D} . Przyjęto stałą wymiarowość problemu D = 20. Brak wyniku dla kwadratowej RW z interakcjami dla $K \in \{50, 100\}$ wynika z tego, że w tych przypadkach liczba obserwacji w zbiorze uczącym jest mniejsza niż liczba parametrów podlegających estymacji.

Porównanie pokazuje przede wszystkim bardzo słabą skalowalność metamodelu Kriginga. Estymacja parametrów dla 4000 obserwacji w zbiorze uczącym trwa w przybliżeniu 590*s*, a dla 8000 obserwacji 5363*s* (około 1.5*h*). Zauważmy, że dla K = 500 uzyskany czas to blisko 6*s*. Stąd nawet bardzo ograniczone budżety optymalizacji będą obarczone długim czasem obliczeń metamodeli Kriginga.

Wykorzystanie metamodelu Kriginga dla małych zbiorów uczących, takich jak K = 50 czy K = 100, może być akceptowalne pod względem wydajności, ponieważ w obu przypadkach estymacja parametrów trwa poniżej 0.5s. Niemniej, jeżeli założyć, że taki proces estymacji parametrów jest ponawiany wielokrotnie, to całkowity czas staje się znaczący, zwłaszcza w zestawieniu z innymi metamodelami.

Zarówno metamodel oparty o RBF, jak i oba warianty RW, charakteryzują się o rzędy wielkości mniejszymi czasami estymacji parametrów w porównaniu do metamodelu Kriginga. Zgodnie z oczekiwaniem skalowanie obu wariantów RW jest zależne w przybliżeniu liniowo od wielkości K zbioru uczącego. Inaczej jest w przypadku metamodelu opartego o RBF, dla którego zaobserwowano zależność oscylującą między kwadratem a sześcianem wielkości K. Dlatego też, o ile dla małych wielości zbiorów uczących $K \leq 100$ czas estymacji parametrów RBF jeszcze można uznać za porównywalny do czasu estymacji parametrów RW, to dla K > 100 wykorzystanie RBF w porównaniu do RW jest znacząco mniej wydajne.

W tabeli 5.2 przedstawiono porównanie czasów estymacji parametrów metamodeli (Mm.) ze względu na liczbę wymiarów problemu $D \in \{2, 5, 10, 20, 30, 40, 60\}$ przy stałym rozmiarze zbioru uczącego K = 2000. Dla metamodelu Kriginga obserwuje się zależność w przybliżeniu liniową czasu estymacji parametrów od liczby wymiarów D. W metamodelu opartym o RBF spadek wydajności wraz ze wzrostem liczby wymiarów D należy uznać za nieistotny. Skalowanie kwadratowej RW jest w przybliżeniu zgodne z kwadratem liczby wymiarów D. Mimo to, dla przedstawionych wielkości D zmierzony czas jest istotnie mniejszy od czasów uzyskanych przez metamodel wykorzystujący RBF. Inną charakterystykę posiada kwadratowa RW z interakcjami, w której liczba parametrów silnie rośnie wraz ze wzrostem liczby wymiarów D. Skutkiem tego, czas estymacji parametrów silnie rośnie wraz ze wzrostem liczby wymiarów D, a dla D = 60 jest on już zbliżony do czasu uzyskanego przez metamodel oparty o RBF.

Tabela 5.1: Porównanie czasu [s] estymacji parametrów metamodeli (Mm.) ze względu na rozmiar $K \in \{50, 100, 500, 1000, 2000, 4000\}$ zbioru uczącego \mathcal{D} . Liczba wymiarów problemu jest stała i wynosi D = 20. W porównaniu uwzględniono metamodel Kriginga, metamodel wykorzystujący RBF, kwadratową RW (ozn. jako k. RW) oraz kwadratową RW z interakcjami (ozn. jako k. RW + in.).

Mm. K	Kriging	RBF	k. RW	k. RW + in.
50	0.1317	0.0006	0.0001	-
100	0.2934	0.0017	0.0002	-
500	5.9226	0.0427	0.0005	0.0043
1000	32.7787	0.1618	0.0007	0.0076
2000	129.3367	0.7067	0.0016	0.0132
4000	589.6646	3.1278	0.0038	0.0250
8000	5363.28570	15.4883	0.0097	0.0571

Podsumowując, eksperymentalne badanie czasu estymacji metamodeli potwierdziło przypuszczenia dotyczące kosztowności procesu uczenia, wynikające z oszacowań złożoności obliczeniowej, zaprezentowanych w rozdziale 4.3. Badanie uwzględniło dodatkowe operacje, takie jak tworzenie wektorów zmiennych objaśnianych, celem możliwie rzeczywistego oszacowania czasu potrzebnego na naukę metamodeli. Kriginga jest metamodelem o najdłuższym czasie nauki oraz z bardzo słabą skalowalnością ze względu na rozmiar zbioru uczącego K. Ograniczenie wielkości zbioru uczącego może pomóc, do pewnego stopnia, zredukować czas estymacji parametrów, lecz wciąż są to wartości relatywnie bardzo duże. Najszybszy oraz najlepiej skalowalny proces nauki charakteryzuje kwadratową RW. Kwadratowa RW z interakcjami również jest bardzo wydajnym metamodelem, lecz dla dużej liczby wymiarów jej wydajność drastycznie spada. Metamodel oparty o RBF osiąga wyniki będące pomiędzy wynikami dla metamodelu Kriginga oraz obu wariantów RW. Sytuacją, w której metamodel oparty o RBF może być estymowany szybciej niż RW jest uwzględnienie interakcji oraz duża liczba wymiarów problemu. Tabela 5.2: Porównanie czasu [s] estymacji parametrów metamodeli (Mm.) ze względu na liczbę wymiarów problemu $D \in \{2, 5, 10, 20, 30, 40, 60\}$. Rozmiar zbioru uczącego \mathcal{D} jest stały i wynosi K = 2000. W porównaniu uwzględniono metamodel Kriginga, metamodel wykorzystujący RBF, kwadratową RW (ozn. jako k. RW) oraz kwadratową RW z interakcjami (ozn. jako k. RW + in.).

D Mm.	Kriging	RBF	k. RW	k. RW + in.
2	11.7991	0.6379	0.0003	0.0003
5	20.2665	0.6468	0.0005	0.0008
10	39.5955	0.6621	0.0007	0.0033
20	129.4163	0.7053	0.0016	0.0143
30	262.6120	0.7585	0.0024	0.0418
40	437.0700	0.7886	0.0047	0.1107
60	747.9587	0.8310	0.0068	0.4339

5.1.3 Dyskusja efektywności zastosowań metamodeli

W rozdziale 4.5.1 dokonano przeglądu znanych metod integracji metamodeli oraz opisano najważniejsze, z punktu widzenia tematyki rozprawy, przykłady APWM. Zasadnym jest ocena efektywności zastosowania metamodeli w APWM zgodnie z kryteriami zawartymi w rozdziale 5.1. Eksperymentalne badanie czasu estymacji parametrów metamodeli, zaprezentowane w rozdziale 5.1.2 stanowi drugi punkt odniesienia w dyskusji.

Znaczna część APWM wykorzystuje metamodel Kriginga lub jego rozszerzenie w sposób zgodny z ideą OB, czyli w oparciu o wszystkie dostępne obserwacje (rozwiązania poddane ewaluacji z wykorzystaniem f.c.). Takie zastosowanie metamodelu nie jest efektywne, ponieważ nie jest spełnione kryterium K1 (duże budżety) ze względu na bardzo słabą skalowalność procesu estymacji parametrów w metamodelu Kriginga w odniesieniu do rozmiaru zbioru uczącego.

Wykorzystanie metamodeli w relatywnie prostych algorytmach populacyjnych, który nie mogą być obecnie uznane za skuteczne metody rozwiązujące, jest sprzeczne ze spełnieniem kryterium K3 (zadowalające wyniki). Przykładem tego są APWM oparte na PSO oraz DE bez mechanizmów adaptacji.

Efektowność niektórych APWM jest trudna do oceny, czego przykładem jest

S-JADE [33], który wykorzystuje metamodel globalny oraz metamodele lokalne, wszystkie oparte o RBF. Bazowy algorytm JADE jest adaptacyjnym rozwinięciem DE, a modelowanie f.c. z wykorzystaniem RBF jest znacząco bardziej wydajne od modelowania f.c. z wykorzystaniem metamodelu Kriginga. Niemniej, S-JADE był ewaluowany na problemach kosztownych, a przedstawiona analiza jego złożoności nie pozwala na ocenę jego wydajności w większych budżetach optymalizacji.

APWM takie jak SMA-EPSDE [143], ESMDE [144], iDEaSm [16], S-CMA-ES [18] oraz DTS-CMA-ES [174] są oparte o Kriginga który jest uczony na podstawie ograniczonego zbioru uczącego. Dzięki temu możliwe jest spełnienie kryteriów K1 (duże budżety), K3 (zadowalające wyniki) oraz K4 (uniwersalność), lecz dyskusyjne pozostaje spełnienie kryterium K2, czyli akceptowalności narzutu obliczeniowego wynikającego z użycia metamodelu. Badania eksperymentalne pokazały, że estymacja parametrów w metamodelu Kriginga dla małych zbiorów uczących jest relatywnie wydajna, lecz wciąż jej czas jest rzędy wielkości większy niż w przypadki estymacji parametrów kwadratowej RW lub RBF. Stąd, narzut obliczeniowy wynikający z użycia metamodelu należy uznać za nieakceptowalny.

Algorytm saACM-ES [139, 140], będący rozwinięciem CMA-ES o rankingową maszynę wektorów nośnych, był ewaluowany w budżecie optymalizacji taniej, w którym pokazał przewagę nad bazowym CMA-ES. Stąd, należy uznać, że saACM-ES spełnia kryterium K1 (duże budżety) oraz K3 (zadowalające wyniki). Kryterium K4 (uniwersalność) jest w dużej mierze spełnione, ponieważ CMA-ES jest algorytmem uniwersalnego przeznaczenia. Wątpliwości budzi zastosowanie strojenia parametru wielkości zbioru uczącego, celem uzyskania jak najlepszych wyników na zbiorze testowym COCO. Problemem saACM-ES jest duży narzut obliczeniowy metamodelu. Czas uczenia metamodelu, zgodnie z wyliczeniami autorów, zależy od sześcianu liczby wymiarów. Skutkiem tego, dla badanych funkcji o D = 40 czas obliczeń całego APWM ($T_{ap} + T_{mm}$) przypadający na jedną ewaluację f.c. oscyluje między $10^{-2}s$ a 0.5s. Dlatego też, kryterium K2 (akceptowalny narzut) nie jest spełnione, jeśli zestawić te wartości z czasami estymacji RW.

Algorytmem, który wprost odnosi się do problemu efektywnego zastosowania metamodelu jest lq-CMA-ES [77]. Wedle jego autorów zaproponowane rozwiązanie jest najprostszym możliwym sposobem wykorzystania metamodelu. Podobnie określono sposób estymacji parametrów metamodelu. Zaprezentowana eksperymentalna ewaluacja lq-CMA-ES z wykorzystaniem zbioru testowego COCO pokazała, że zaprezentowany algorytm z powodzeniem osiągnął zadowalające wyniki w budżetach optymalizacji taniej. Empiryczne badanie czasu działania pokazało, że w przypadku funkcji o D = 20 lq-CMA-ES był 53 razy wolniejszy od bazowego CMA-ES, co należy uznać za wartości akceptowalne biorąc pod uwagę budżet optymalizacji. Co więcej, uzyskane wyniki były porównywalne, a nawet lepsze od alternatywnych rozwiązań opartych o znacznie bardziej złożone metamodele (lmm-CMA-ES, DTS-CMA-ES oraz sa-ACM-ES).

Algorytm lq-CMA-ES pokazał, że jest możliwe stosowanie metamodeli wykorzystujących RW w budżetach optymalizacji większych niż kosztowne i że uzyskane wyniki są satysfakcjonujące. Niemniej, ewaluacja lq-CMA-ES oraz jego konkurentów pokazała, że wszystkie sprawdzane algorytmy wykorzystujące metamodel czerpały z niego zysk w budżecie mniejszym niż $10^3 \cdot D$ ewaluacji f.c. Powyżej tej wartości CMA-ES bez wsparcia metamodelu był w stanie osiągać założone cele (błędy) równie szybko (przy podobnej liczbie ewaluacji f.c.). Ponadto, należy zauważyć, że użycie metamodelu w lq-CMA-ES wymaga określenia wielu parametrów (m.in. wagi w MNK, reguły wyznaczania τ -Kendalla), co istotnie wpływa na ocenę spełniania kryterium K4, mówiącym o uniwersalności metody.

Podsumowując, w literaturze znajduje się wiele APWM różniących się zarówno rodzajem wykorzystywanego metamodelu, ale także sposobem jego integracji, ze szczególnym uwzględnieniem konstrukcji zbioru uczącego. Część APWM znalazła bądź mogłaby znaleźć zastosowanie w rozwiązywaniu problemów semikosztownych lub tanich. Biorąc pod uwagę analizowane cztery kryteria efektywnego zastosowania metamodelu wyróżnić należy lq-CMA-ES, który na tle innych APWM osiąga dobre wyniki na zbiorze testowym COCO, a sama konstrukcja metamodelu RW skutkuje akceptowalnym narzutem obliczeniowym.

5.2 Proponowane zastosowania metamodeli

W ramach prac badawczych autor rozprawy poszukiwał sposobów na efektywne zastosowanie metamodeli w algorytmach populacyjnych w kontekście celów badawczych, zaprezentowanych w rozdziale 1.2. Skutkiem tego, opracowano nowe APWM oraz zbadano ich wybrane właściwości, skupiając się na analizie działania metamodeli. Następnie, zaproponowane APWM oraz zawarte w nich sposoby integracji metamodeli zostały poddane eksperymentalnej ewaluacji celem potwierdzenia lub odrzucenia prawdziwości hipotez badawczych zaprezentowanych w rozdziale 1.3.

Autor rozprawy zaznacza, że proponowane efektywne zastosowania metamodeli oraz towarzyszące im APWM nie stanowią jedynych możliwych sposobów na spełnienie postawionych celów badawczych. Niemniej jednak, stanowią podejście spójne i potwierdzone wynikami eksperymentalnymi. W trakcie prac badawczych podejmowano liczne decyzje dotyczące opracowywanych rozwiązań, motywowane analizą bieżącego stanu wiedzy oraz otrzymywanymi wynikami eksperymentalnymi. Wskutek tego poczyniono następujące główne założenia projektowe:

- 1. We wszystkich rozwiązaniach jako metamodel wybrano regresję wielomianową.
- 2. Zdecydowano się rozszerzać za pomocą metamodeli algorytmy GAPSO, SHADE, R-SHADE oraz L-SHADE.
- 3. Wykorzystano lokalną optymalizację metamodelem jako mechanizm poprawiający wyniki algorytmów populacyjnych w optymalizacji taniej.
- 4. Wykorzystano preselekcję rozwiązań za pomocą metamodelu jako mechanizm poprawiający wyniki algorytmów populacyjnych w optymalizacji semikosztownej.

Motywacją wyboru RW jako metamodelu jest jej wydajność, potwierdzona wynikami badań eksperymentalnych (rozdział 5.1.2) oraz wynikami literaturowymi, w szczególności dot. lq-CMA-ES [77].

Wybrane algorytmy są skutecznymi metodami rozwiązującymi, czego dowodzą wyniki ich eksperymentalnej ewaluacji. Ponadto, relatywnie duża liczba znanych i skutecznych zastosowań metamodeli w algorytmach z rodziny CMA-ES sprawiła, że zbadanie możliwości wykorzystania metamodeli w algorytmach z rodziny DE wydawało się wartościowe oraz obiecujące.

Poniżej opisano zwięźle idee stojące za proponowanymi zastosowaniami metamodeli w algorytmach populacyjnych. W rozdziale 5.2.1 opisano mechanizm inicjalizacji metamodelem, w rozdziale 5.2.2 mechanizm lokalnej optymalizacji metamodelem, w rozdziale 5.2.3 mechanizm preselekcji rozwiązań za pomocą metamodelu, a w rozdziale 5.2.4 rekurencyjną metodę estymacji parametrów metamodelu.

Zastosowanie ww. mechanizmów w konkretnych algorytmach populacyjnych oraz odpowiadające im wyniki eksperymentalne opisano w rozdziale 6 oraz rozdziale 7. Rozdział 6 omawia algorytmy wykorzystujące mechanizm lokalnej optymalizacji metamodelem, które znalazły zastosowanie w optymalizacji taniej, a rozdział 7 algorytmy dedykowane do optymalizacji semikosztownej oparte o preselekcję rozwiązań za pomocą metamodelu.

5.2.1 Inicjalizacja metamodelem

Celem mechanizmu inicjalizacji metamodelem jest skierowanie małym kosztem inicjalnej populacji w obiecujący obszar przestrzeni rozwiązań. W tym celu generowana jest losowo inicjalna populacja $P^0 = [\boldsymbol{x}_1^0, \dots, \boldsymbol{x}_N^0]$. Następnie, wszystkie osobniki \boldsymbol{x}_i^0 , gdzie $i \in \{1, \dots, N\} \setminus \{df_{lin} + 1, df_{kw} + 1\}$ są kolejno ewaluowane, tzn. wyznaczana jest wartość $f(\boldsymbol{x}_i^0)$. Każde obliczenie wartości $f(\boldsymbol{x}_i^0)$ skutkuje dodaniem pary $(\boldsymbol{x}_i^0, f(\boldsymbol{x}_i^0))$ do zbioru uczącego \mathcal{D} .

Współrzędne osobników $\boldsymbol{x}_{df_{lin}+1}^{0}$ oraz $\boldsymbol{x}_{df_{kw}+1}^{0}$ są modyfikowane bezpośrednio przed wykonaniem na nich ewaluacji z wykorzystaniem f.c. W tym celu estymowany jest metamodel RW, odpowiednio liniowej dla $\boldsymbol{x}_{df_{lin}+1}^{0}$ oraz kwadratowej dla $\boldsymbol{x}_{df_{kw}+1}^{0}$. Stąd, wartość df_{lin} jest liczbą stopni swobody liniowej RW, a df_{kw} liczbą stopni swobody kwadratowej RW. Współrzędne osobnika $\boldsymbol{x}_{df_{lin}+1}^{0}$ są zmieniane na współrzędne rozwiązania optymalnego liniowej RW. Analogicznie, dla osobnika $\boldsymbol{x}_{df_{kw}+1}^{0}$ współrzędne są zmieniane na współrzędne rozwiązania optymalnego kwadratowej RW. Postać obu modeli jest przedstawiona w tabeli 5.3.

Tabela 5.3: Opis metamodeli stosowanych w mechanizmie inicjalizacji metamodelem. Estymowane współczynniki przy każdej ze zmiennych zostały pominięte dla zwiększenia czytelności.

Nazwa	Postać	Stopnie swobody
Liniowa RW	$\tilde{\boldsymbol{x}}_{lin} = [1, x_1, \dots, x_D]$	$df_{lin} = D + 1$
Kwadratowa RW	$\tilde{\boldsymbol{x}}_{kw} = [X_{lin}, x_1^2, \dots, x_D^2]$	$df_{kw} = 2D + 1$

W przypadku liniowej RW każda ze współrzędnych rozwiązania optymalnego przyjmuje wartość dolnego $(x_{min,d})$ albo górnego $(x_{max,d})$ ograniczenia przestrzeni rozwiązań w wymiarze d. W przypadku kwadratowej RW każda ze współrzędnych rozwiązania optymalnego może dodatkowo przyjąć wartość wierzchołka paraboli $x_{peak,d}$ w wymiarze d. Całościowe spojrzenie na mechanizm inicjalizacji metamodelem jest zwarte w pseudokodzie 5.

Inicjalizacja metamodelem jest mechanizmem, który może być zastosowany w każdym algorytmie populacyjnym. Spośród algorytmów omawianych w rozprawie z proponowanego mechanizmu korzysta M-GAPSO (rozdział 6.2) oraz LQ-R-SHADE (rozdział 7.1). Algorytm 5 Inicjalizacja metamodelem

1: g = 02: $P^0 = [\boldsymbol{x}_1^0, \dots, \boldsymbol{x}_N^0], \boldsymbol{x}_i^0 \sim \mathcal{U}(x_{min}, x_{max})$ 3: for i = 1 do N do if $i == df_{lin} + 1$ then 4: Estymuj metamodel \hat{f}_{lin} liniowej RW za pomocą zbioru uczącego \mathcal{D} 5: $\pmb{x}_{i}^{0} = \pmb{x}_{lin}^{*}$ (rozwiązanie opt. liniowej RW, gdzi
e $x_{lin,d}^{*} \in \{x_{min,d}, x_{max,d}\}$) 6: 7:else if $i == df_{quad} + 1$ then 8: Estymuj metamodel \hat{f}_{lin} kwadratowej RW za pomocą zbioru uczącego \mathcal{D} 9: $\boldsymbol{x}_{i}^{0} = \boldsymbol{x}_{kw}^{*}$ (rozwiązanie opt. kwadratowej RW, gdzie $x_{quad,d}^{*} \in \{x_{min,d}, x_{max,d}, x_{peak,d}\}$) 10: end if Ewaluuj \boldsymbol{x}_i^0 za pomocą f.c. 11: Dodaj $(\boldsymbol{x}_i^0, f(\boldsymbol{x}_i^0))$ do zbioru uczącego \mathcal{D} 12:13: **end for**

5.2.2 Lokalna optymalizacja metamodelem

Lokalna optymalizacja metamodelem zakłada modyfikację reguł zachowania pojedynczego osobnika w populacji, tzn. reguł wyznaczających nowe rozwiązanie \boldsymbol{x}_i^g , poddawane ewaluacji z wykorzystaniem f.c.

Optymalizacja wykorzystuje metamodel, który jest estymowany w oparciu o zbiór uczący \mathcal{D} , składający się z rozwiązań dotychczas poddanych ewaluacji oraz odpowiadających im wartości f.c. Zakłada się, że zbiór uczący jest ograniczony do obszaru stanowiącego otoczenie rozwiązania \boldsymbol{x}_{i}^{g} , wyznaczone w poprzedniej iteracji.

Nowe położenie *i*-tego osobnika w iteracji g jest zdefiniowane jako rozwiązanie optymalne \hat{x}^* metamodelu \hat{f} . Szczegółowa definicja metamodelu oraz sposobu konstruowania zbioru uczącego \mathcal{D} jest zależna od konkretnej implementacji lokalnej optymalizacji metamodelem. Całościowe spojrzenie na logikę lokalnej optymalizacji metamodelem, która ma zastosowanie dla *i*-tego osobnika w iteracji g jest przedstawione za pomocą pseudokodu 6.

$\operatorname{Algorytm}$	6	Lokalna	optyma	lizacja	metamodelem
---------------------------	---	---------	--------	---------	-------------

```
1: Skonstruuj zbiór uczący \mathcal{D},stanowiący pewne otoczenie punktu\pmb{x}_i^g
```

```
2: Estymuj parametry metamodel<br/>u\hat{f}na podstawie zbioru uczącego \mathcal D
```

3: $\pmb{x}_i^g = \hat{\pmb{x}}^*,$ gdzie $\hat{\pmb{x}}^*$ jest rozwiązaniem optymalnym metamodelu \hat{f}

4: Ewaluuj \boldsymbol{x}_i^g za pomocą f.c.

5: Dodaj $(\boldsymbol{x}_{i}^{g}, f(\boldsymbol{x}_{i}^{g}))$ do zbioru uczącego \mathcal{D}

Lokalna optymalizacja metamodelem znalazła zastosowanie w algorytmie M-GAPSO oraz związanym z nim SHADE-LM, co zostało szerzej opisane w rozdziale 6.

5.2.3 Preselekcja rozwiązań na podstawie wartości metamodelu

Preselekcja rozwiązań na podstawie wartości metamodelu ma za zadanie ocenić, które z rozwiązań powinny być ewaluowane z wykorzystaniem f.c., a które warto porzucić bez dokonywania ich ewaluacji. W tym celu generowanych jest więcej nowych osobników niż sugeruje to bieżący rozmiar populacji N^g , a następnie metamodel ponownie wskazuje N^g najlepszych z nich, które tworzą finalny zbiór rozwiązań poddawany ewaluacji z wykorzystaniem f.c.

Zaprezentowana preselekcja rozwiązań jest wykorzystywana w adaptacyjnych algorytmach z rodziny DE, takich jak SHADE, R-SHADE oraz L-SHADE, więc zakłada się, że multiplikacji podlegają wektory próby \boldsymbol{u}_i^g , generowane na podstawie wektora macierzystego \boldsymbol{x}_i^g .

Preselekcja rozwiązań opiera się o metamodel \hat{f} , który jest estymowany na podstawie zbioru rozwiązań dotychczas poddanych ewaluacji. Zakłada się wykorzystanie metamodeli z grupy RW. Rozmiar zbioru uczącego \mathcal{D} jest ograniczony z dwóch powodów. Po pierwsze ze względów wydajnościowych, celem szybszej estymacji parametrów metamodeli. Po drugie, żeby umożliwić lepsze dopasowanie metamodelu do obszaru przestrzeni przeszukiwań, jaki jest zajmowany przez bieżącą populację.

Preselekcja rozwiązań może występować w dwóch wariantach: lokalnym i globalnym. W wariancie lokalnym każdy *i*-ty osobnik \boldsymbol{x}_i^g generuje N_s wektorów próby $\boldsymbol{u}_i^{g,j}$, gdzie $j = 1, \ldots, N_s$. Następnie spośród nich wybierany jest najlepszy $\boldsymbol{u}_i^{g,best}$ ze względu na wartość \hat{f} i to on jest poddany ewaluacji z wykorzystaniem f.c.

Algorytm 7	7 Preselekcja	lokalna
------------	---------------	---------

1: for $i = 1 \text{ do } N^g$ do

- 2: Wygeneruj N_s wektorów próby $\boldsymbol{u}_i^{g,j}$ na podstawie bieżącego rozwiązania \boldsymbol{x}_i^g , zgodnie z logiką algorytmu
- 3: end for
- 4: Estymuj parametry metamodel
u \hat{f} w oparciu o zbiór uczący $\mathcal D$
- 5: Wyznacz wartość \hat{f} dla wszystkich $N_s\cdot N^g$ wektorów próby
- 6: for i = 1 do N^g do
- 7: Wybierz najlepszy wektor próby $\boldsymbol{u}_{i}^{g,best}$ spośród N_s wektorów próby $\boldsymbol{u}_{i}^{g,j}$
- 8: Ewaluuj $\boldsymbol{u}_i^{g,best}$ za pomocą f.c. f
- 9: Dodaj $(\boldsymbol{u}_i^{g,best}, f(\boldsymbol{u}_i^{g,best}))$ do zbioru uczącego \mathcal{D}

10: end for

W wariancie globalnym cała populacja P^g o rozmiarze N^g jest rozszerzana (multiplikowana) N^s razy do populacji $P_{ext}^g = [P^g, P^g, \dots, P^g]$ o rozmiarze $N_s \cdot N^g$. Następnie każdy osobnik \boldsymbol{x}_{k}^{g} , gdzie $k = 1, \ldots N_{s} \cdot N^{g}$ generuje swój wektor próby \boldsymbol{u}_{k}^{g} , lecz finalnie tylko N^{g} najlepszych wektorów próby $\boldsymbol{u}_{i}^{g,best}$, ze względu na wartość \hat{f} , jest poddawanych ewaluacji z wykorzystaniem f.c.

Alg	gorytm	8	Presel	ekcja	globa	alna
-----	--------	---	--------	-------	-------	------

- 1: Rozszerz populację P^g do populacji $P^g_{ext} = [P^g, P^g, \dots, P^g],$ gdzie $|P^g_{ext}| = N_s \cdot N^g$
- 2: Wygeneruj $N_s \cdot N^g$ wektorów próby $\boldsymbol{u}_i^{g,k}$ na podstawie bieżącego rozwiązania \boldsymbol{x}_i^g , zgodnie z logiką algorytmu
- 3: Estymuj parametry metamodel
u \hat{f} w oparciu o zbiór uczący $\mathcal D$
- 4: Wyznacz wartość \hat{f} dla wszystkich $N_s\cdot N^g$ wektorów próby
- 5: for i = 1 do N^g do
- 6: Wybierz najlepszy ze względu na \hat{f} , nie wybrany dotychczas, wektor próby $\boldsymbol{u}_i^{g,best}$ spośród $N_s \cdot N^g$ wektorów próby \boldsymbol{u}_k^g
- 7: Ewaluuj $\boldsymbol{u}_{i}^{g,best}$ za pomocą f.c. f
- 8: Dodaj $(\boldsymbol{u}_i^{g,best}, f(\boldsymbol{u}_i^{g,best}))$ do zbioru uczącego \mathcal{D}

9: end for

Ideę pojedynczej iteracji algorytmu zawierającej lokalną preselekcję rozwiązań przedstawiono za pomocą pseudokodu 7. Analogiczne spojrzenie na preselekcję globalną przedstawiono za pomocą pseudokodu 8.

Finalna integracja preselekcji rozwiązań z całością algorytmu, z uwzględnieniem jej wpływu na mechanizm adaptacji parametrów, została szerzej omówiona w rozdziale 7. Algorytmy populacyjne, które wykorzystują preselekcję rozwiązań to: LQ-R-SHADE (rozdział 7.1), psLSHADE (rozdział 7.2) oraz rmmLSHADE (rozdział 7.3).

5.2.4 Rekurencyjna estymacja parametrów metamodelu

Klasyczne wykorzystanie metamodelu RW w algorytmie populacyjnym zakłada cykliczną estymację jego parametrów z wykorzystaniem MNK, która wymaga istnienia zbioru uczącego \mathcal{D} . W przypadku RW, liczba obserwacji w zbiorze uczącym musi być co najmniej taka jak liczba stopni swobody modelu. Skutkiem tego minimalny rozmiar zbioru uczącego dla kwadratowej RW z interakcjami staje się znaczący wraz ze wzrostem liczby wymiarów.

Konieczność przechowywania oraz cyklicznego aktualizowania zbioru uczącego \mathcal{D} była główną motywacją poszukiwania alternatywnych metod estymacji parametrów w RW. Zastąpienie MNK pewną formą rekurencyjnej estymacji parametrów, tzn. w oparciu o pojedynczą obserwację (\boldsymbol{x}_i, y_i), pozwoliłoby osiągnąć następujące cele:

- 1. Zmniejszenie stopnia skomplikowania APWM poprzez eliminację mechanizmów konstruujących zbiór uczący.
- 2. Zwiększenie wydajności APWM poprzez mniej złożone obliczenia algebraiczne.
- 3. Zmniejszenie stopnia parametryzacji APWM poprzez zapewnienie czytelnej logiki douczania metamodelu bezpośrednio po każej ewaluacji rozwiązania z wykorzystaniem f.c.

Rekurencyjna estymacja parametrów w RW może być zapewnienia przez *rekursywny* filtr najmniejszych kwadratów (filtr RLS), inaczej zwany *rekurencyjną metodą najmniej*szych kwadratów.

Filtr RLS [194, równania (21.36) - (21.39)] jest adaptacyjnym algorytmem, który rekurencyjnie rozwiązuje problem najmniejszych kwadratów. Podaje procedurę obliczającą współczynniki \boldsymbol{w}_t na podstawie \boldsymbol{w}_{t-1} biorąc pod uwagę sygnał wejściowy \boldsymbol{a}_t , sygnał wyjściowy d_t i estymację sygnału wyjściowego $\hat{d}_t = \boldsymbol{a}_t^T \boldsymbol{w}_{t-1}$. $e_t = d_t - \boldsymbol{a}_t^T \boldsymbol{w}_{t-1}$ jest błędem sygnału wyjściowego. Suma kwadratów błędów jest funkcją kosztu podlegającą minimalizacji.

Aktualizacja parametrów \boldsymbol{w}_t metamodelu w chwili t na podstawie obserwacji $(\boldsymbol{x}_t, f(\boldsymbol{x}_t))$ może być zrealizowana poprzez przyjęcie następujących założeń: $\boldsymbol{a}_t = \tilde{\boldsymbol{x}}_t, d_t = f(\boldsymbol{x}_t)$ oraz $\hat{d}_t = \hat{f}(\boldsymbol{x}_t) = \tilde{\boldsymbol{x}}_t^\top \boldsymbol{w}_{t-1}$, gdzie $\tilde{\boldsymbol{x}}_t$ jest wektorem zmiennych objaśniających utworzonym na podstawie \boldsymbol{x}_t , zgodnie z przyjętą postacią RW. Całość jest opisana poniższym układem równań:

$$e_{t} = f(\boldsymbol{x}_{t}) - \tilde{\boldsymbol{x}}_{t}^{\top} \boldsymbol{w}_{t-1}$$
$$\boldsymbol{g}_{t} = \frac{Q_{t-1} \tilde{\boldsymbol{x}}_{t}}{\lambda + \tilde{\boldsymbol{x}}_{t}^{\top} Q_{t-1} \tilde{\boldsymbol{x}}_{t}}$$
$$Q_{t} = \frac{1}{\lambda} (Q_{t-1} - \boldsymbol{g}_{t} \tilde{\boldsymbol{x}}_{t}^{\top} Q_{t-1})$$
$$\boldsymbol{w}_{t} = \boldsymbol{w}_{t-1} + \boldsymbol{q}_{t} e_{t}$$
(5.3)

gdzie $\lambda \in (0, 1]$ jest czynnikiem zapominania (ang. forgetting factor), a Q_t jest macierzą o rozmiarze $|\tilde{\boldsymbol{x}}_t| \times |\tilde{\boldsymbol{x}}_t|$.

Złożoność obliczeniowa estymacji parametrów za pomocą filtru RLS wynosi $\mathcal{O}(L^2)$ [194], gdzie L jest liczbą parametrów RW. Stąd dla liniowej oraz kwadratowej RW otrzymuje się $\mathcal{O}(D^2)$, a dla kwadratowej RW z interakcjami odpowiednio $\mathcal{O}(D^4)$. Dla przypomnienia MNK posiada złożoność $\mathcal{O}(L^2K)$, gdzie K jest liczbą obserwacji w zbio-

rze uczącym, która w założeniu musi być co najmniej taka jak liczba parametrów (stopni swobody) metamodelu.

Zakładając, że rozmiar zbioru uczącego w MNK jest niewiele większy od liczby parametrów metamodelu, to wykorzystanie filtra RLS redukuje złożoność estymacji parametrów z $\mathcal{O}(D^3)$ do $\mathcal{O}(D^2)$ dla liniowej oraz kwadratowej RW. W przypadku kwadratowej RW z interakcjami złożoność maleje z $\mathcal{O}(D^6)$ do $\mathcal{O}(D^4)$.

Wykorzystanie filtru RLS w estymacji parametrów znalazło zastosowanie w algorytmie rmmLSHADE, omówionym w rozdziale 7.3.

ROZDZIAŁ 5. ZASTOSOWANIA METAMODELI W ALGORYTMACH POPULACYJNYCH

Rozdział 6

Lokalna optymalizacja metamodelem w optymalizacji taniej

W tym rozdziale przedstawiono badania autora rozprawy dotyczące zastosowania lokalnej optymalizacji metamodelem w wybranych algorytmach populacyjnych. Najpierw opisano środowisko GAPSO, które umożliwia integrację wielu algorytmów populacyjnych w obrębie jednej populacji. Następnie przedstawiono koncepcję integracji lokalnej optymalizacji metamodelem w algorytmie GAPSO, skutkującej powstaniem M-GAPSO.

W oparciu o algorytm M-GAPSO stworzono algorytm SHADE-LM, który został opisany w drugiej części tego rozdziału. SHADE-LM jest połączeniem R-SHADE oraz lokalnej optymalizacji metamodelem.

Zarówno M-GAPSO, jak i SHADE-LM zostały poddane eksperymentalnej ewaluacji, w której oceniono zysk z integracji lokalnej optymalizacji metamodelem. Na koniec przedstawiono analizę wpływu użycia metamodeli na całkowity czas obliczeń M-GAPSO oraz SHADE-LM.

6.1 Algorytm GAPSO

Hybrydowe algorytmy populacyjne nie stanowią głównego przedmiotu zainteresowania rozprawy. Mimo to, niektóre metody łączenia algorytmów populacyjnych mogą stanowić inspirację dla integracji metamodeli. Przykładem takiego algorytmu jest *Generalized Self-Adapting Particle Swarm Optimization* [221] (GAPSO), który jest hybrydowym algorytmem populacyjnym rozszerzającym PSO.

GAPSO jest motywowany obserwacją, że osobniki w populacji (w przypadku PSO

ROZDZIAŁ 6. LOKALNA OPTYMALIZACJA METAMODELEM W OPTYMALIZACJI TANIEJ

można mówić o cząsteczkach w roju) nie muszą zachowywać się identycznie, tzn. zasady wyznaczania kolejnych rozwiązań celem ewaluacji mogą być różne dla różnych osobników. W założeniu ma to umożliwić osiągnięcie lepszych wyników, niż gdyby populacja była homogeniczna. GAPSO zakłada, że osobniki znają swoje położenia i współdzielą zmienne oraz parametry w sposób charakterystyczny dla podejść niehybrydowych. Pierwotna koncepcja GAPSO wykorzystywała dwa algorytmy: PSO oraz DE, przy czym PSO zostało zaimplementowane w czterech wariantach, różniących się regułami wyznaczania kolejnych rozwiązań poddawanych ewaluacji. W ogólności, GAPSO może składać się z K różnych algorytmów. Wysokopoziomowe spojrzenie na algorytm przedstawiono za pomocą pseudokodu 9.

Algorytm 9 GAPSO

1:	Ustaw parametry algorytmu
2:	Wylosuj początkową populację $P^0 = [\boldsymbol{x}_1^0, \dots, \boldsymbol{x}_N^0]$
3:	while budżet (dostępna liczba ewaluacji f.c.) nie jest wyczerpany ${\bf do}$
4:	Dla każdego osobnika i wylosuj algoryt m k zgodnie z wektorem prawdopodobieństwa \pmb{w}_a
5:	for $i = 1 do N do$
6:	Wyznacz nowe rozwiązanie \boldsymbol{x}_{i}^{g} , zgodnie z przypisanym algorytmem k
7:	Wyznacz wartość f.c. $f(\boldsymbol{x}_{i}^{g})$
8:	if Osobnik i wymaga restartu then
9:	Ustaw \pmb{x}_i^g w losowym miejscu w przestrzeni rozwiązań
10:	end if
11:	end for
12:	(opcjonalnie) dokonaj adaptacji wektora prawdopodobieństwa \pmb{w}_a
13:	g = g + 1
14:	end while

Na początku każdej iteracji g każdy *i*-ty osobnik $\boldsymbol{x}_i^g = [x_{i,1}^g, \ldots, x_{i,D}^g]$ w populacji $P^g = [\boldsymbol{x}_1^g, \ldots, \boldsymbol{x}_N^g]$ ma przypisywane, w sposób losowy, swoje zachowanie, czyli rodzaj algorytmu, który będzie określał reguły wyznaczenia następnego rozwiązania \boldsymbol{x}_i^g w oparciu o poprzednie rozwiązanie \boldsymbol{x}_i^{g-1} .

Losowanie algorytmów jest dokonywane w oparciu o wektor wag prawdopodobieństwa \boldsymbol{w}_a wyboru konkretnego zachowania, który jest parametrem algorytmu. Przykładowo, wektor $\boldsymbol{w}_a = [0.6, 0.1, 0.1, 0.1, 0.1]$ oznacza, że pierwszy z K = 5 algorytmów będzie wybierany z prawdopodobieństwem 0.6, a pozostałe cztery z prawdopodobieństwem 0.1. Dodatkowo, logika GAPSO zapewnia, że każdy możliwy algorytm będzie przypisany co najmniej jednemu osobnikowi.

Wektor wag prawdopodobieństwa może być stały, tj. ustalany przed uruchomieniem

algorytmu albo podlegać adaptacji w trakcie procesu optymalizacji. Ponadto, GAPSO może dokonać restartu pojedynczego osobnika, tzn. zmienić jego rozwiązanie na losowe w przestrzeni rozwiązań w sytuacji gdy osobnik pozostaje w miejscu od dłuższego czasu lub wykryto, że znajduje się w lokalnym optimum. O szczegółach GAPSO, z uwzględnieniem szczegółowego opisu mechanizmu adaptacji oraz restartów, można przeczytać w pracy [221].

6.2 M-GAPSO: GAPSO z lokalną optymalizacją metamodelem

Skuteczne wykorzystanie heterogenicznej populacji w GAPSO sprawiło, że zaczęto poszukiwać sposobów na rozszerzenie algorytmu o lokalną optymalizację metamodelem (rozdział 5.2.2). Zauważono, że metamodel może być zaimplementowany jako jeden spośród K wykorzystywanych algorytmów, który podobnie jak DE oraz PSO, wskazuje nowe rozwiązanie \boldsymbol{x}_i^g , które jest poddawane ewaluacji z wykorzystaniem f.c. Skutkiem tego, grupa inicjalnych algorytmów (DE oraz PSO) w GAPSO została rozszerzona o lokalną optymalizację metamodelem. Rozszerzony algorytm GAPSO jest określany jako M-GAPSO [165]. Opis algorytmów, z wyłączeniem lokalnej optymalizacji metamodelem, jest przedstawiony w rozdziale 6.2.1. Metamodele stosowane w lokalnej optymalizaji metamodelem opisano w rozdziale 6.2.2.

Oprócz lokalnej optymalizacji metamodelem, M-GAPSO został rozbudowany względem GAPSO o wykorzystanie następujących komponentów:

- 1. Nowych reguł adaptacji wektora prawdopodobieństwa p.
- 2. Nowego mechanizmu restartów.
- 3. Archiwum próbek (par $(\pmb{x},f(\pmb{x})),$ uzyskanych wskutek dotych
czasowych ewaluacji f.c.
- 4. Inicjalizacji metamodelem (rozdział 5.2.1).

Adaptacja wektora wag \boldsymbol{w}_a , podobnie jak w przypadku GAPSO, jest wykonywana pod koniec każdej iteracji g.

Adaptacja polega na mierzeniu wpływu algorytmu k w iteracji g na przebieg optymalizacji, który jest określany jako wartość nagrody $r_{k,g}$. Nagroda bierze pod uwagę skuteczność wszystkich osobników, wykorzystujących dany algoryt
mk, w ciągu ostatnich G_a iteracji. Wartość nagrody
 $r_{k,g}$ jest opisana poniższym równaniem:

$$r_{k,g} = \frac{\sum_{g'=g-G_a}^{g-1} \left(\sum_{j \in \mathcal{K}_k^{g'}} \max\left(f(\boldsymbol{x}^{g',best}) - f(\boldsymbol{x}_j^{g'}), 0 \right) \right)}{\sum_{g'=g-G_a}^{g-1} |\mathcal{K}_k^{g'}|}$$
(6.1)

gdzie $\mathcal{K}_k^{g'}$ oznacza zbiór indeksów osobników używających algorytmu k w iteracji g', $f(\boldsymbol{x}_j^{g'})$ jest wartością f.c. *j*-tego osobnika po wykonaniu iteracji g', a $f(\boldsymbol{x}^{g',best})$ jest najlepszą wartością f.c. znalezioną do iteracji g' włącznie.

Finalny wektor wag prawdopodobieństwa p jest wyznaczany w oparciu o skalowanie wektora $[r_{1,g}, \ldots, r_{K,g}]$, tj.:

$$p = \left[\frac{r_{1,g}}{r^{g,sum}}, \dots, \frac{r_{K,g}}{r^{g,sum}}\right]$$
(6.2)

gdzie $r^{g,sum} = \sum_{k=1}^{K} r_{k,g}$.

Mechanizm restartów został zaimplementowany analogicznie jak w algorytmie JA-DE [177]. Restartowana jest cała populacja w sytuacji, gdy spełnione są poniższe dwa warunki. Po pierwsze, mierzone jest rozproszenie populacji $V(P^g)$ (równanie 6.3) i sprawdzane jest, czy jest ono mniejsze niż zadany próg V^{max} .

$$V(P^{g}) = \frac{1}{D} \sum_{d=1}^{D} Var(P_{d}^{g})$$
(6.3)

gdzie Var oznacza funkcję wariancji, a $P_d^g = [x_{1,d}, \ldots, x_{N,d}]$ jest wektorem współrzędnych d należących do wszystkich osobników populacji P^g .

Drugi warunek zakłada sprawdzenie, czy najlepsze znalezione dotychczas rozwiązanie nie zostało poprawione w ciągu ostatnich G^{max} iteracji.

Archiwum próbek, przechowuje pary $(\boldsymbol{x}, f(\boldsymbol{x}))$, które uzyskano wskutek dotychczasowych ewaluacji f.c. Dodanie nowej pary do archiwum odbywa się bezpośrednio po każdej ewaluacji f.c.

Wyszukiwanie elementów w archiwum jest realizowane z wykorzystaniem wielowymiarowej struktury R-drzewa [20]. Ze względów wydajnościowych rozmiar archiwum został ograniczony do stałej wartości N_a , która jest parametrem algorytmu. Po wypełnieniu archiwum, jest ono resetowane i zbieranie par $(\boldsymbol{x}, f(\boldsymbol{x}))$ zaczyna się od nowa.

Wysokopoziomowe spojrzenie na M-GAPSO, z uwzględnieniem lokalnej optymalizacji metamodelem zostało zawarte w pseudokodzie 10. O szczegółach algorytmu M-GAPSO można przeczytać w pracach [245, 165].

Alg	gorytm 10 M-GAPSO
1:	Ustaw parametry algorytmu
2:	while budżet optymalizacji nie jest wyczerpany do
3:	Wylosuj początkową populację $P^0 = [\pmb{x}_1^0, \dots, \pmb{x}_N^0]$, za pomocą inicjalizacji metamodelem (pseudo-
	kod 5)
4:	while budżet (dostępna liczba ewaluacji f.c.) nie jest wyczerpany ${\bf do}$
5:	Dla każdego osobnika i wylosuj algoryt m k zgodnie z wektorem prawdopodobieństwa \pmb{w}_a
6:	for $i = 1 do N do$
7:	Wyznacz nowe rozwiązanie \boldsymbol{x}_i^g , zgodnie z przypisanym algorytmem k
8:	Wyznacz wartość $f(\pmb{x}_i^g)$
9:	end for
10:	(opcjonalnie) dokonaj adaptacji wektora prawdopodobieństwa \pmb{w}_a
11:	if restart wymagany zgodnie z zasadami [177] then break
12:	end if
13:	g = g + 1
14:	end while
15:	end while

6.2.1 Opis algorytmów w M-GAPSO

W M-GAPSO zdecydowano się na pozostawienie tylko dwóch algorytmów (nie stanowiących lokalnej optymalizacji metamodelem), tj. PSO w wariancie SPSO-2007 [40] oraz DE w wariancie DE/best/1/bin.

PSO (wariant SPSO-2007) zakłada, że w iteracji g osobnik i posiada pewne położenie \boldsymbol{x}_i^g , najlepsze rozwiązanie \boldsymbol{p}_i^g znalezione dotychczas przez siebie oraz najlepsze rozwiązanie \boldsymbol{l}_i^g znalezione dotychczas przez swoich sąsiadów. Sąsiedztwo dla osobnika i jest definiowane jako trzy osobniki o współrzędnych $i-1 \mod(N)$, $i, i+1 \mod(N)$,.

W pierwszym kroku wyznaczana jest prędkość \boldsymbol{v}_i^g w następujący sposób:

$$\boldsymbol{v}_{i}^{g} = \omega \boldsymbol{v}_{i}^{g} + U(0, c_{1})(\boldsymbol{p}_{i}^{g} - \boldsymbol{x}_{i}^{g})) + U(0, c_{2})(\boldsymbol{l}_{i}^{g} - \boldsymbol{x}_{i}^{g})$$
(6.4)

gdzie ω , c_1 oraz c_2 są parametrami, odpowiednio czynnikiem bezwładności, czynnikiem poznawczym oraz czynnikiem społecznym. $U(0,c_1)$ oraz $U(0,c_2)$ są losowo (jednostajnie) generowanymi zmiennymi z przedziału odpowiednio $(0,c_1)$ oraz $(0,c_2)$.

W drugim kroku wyznaczana jest finalne położenie \boldsymbol{x}_i^{g+1} , które podlega ewaluacji z wykorzystaniem f.c. Położenie \boldsymbol{x}_i^{g+1} jest opisane następującym równaniem:

$$\boldsymbol{x}_i^{g+1} = \boldsymbol{x}_i^g + \boldsymbol{v}_i^g \tag{6.5}$$

Wariant DE/best/1/bin zakłada relatywnie prostą fazę mutacji, którą można opisać następującym równaniem:

$$\boldsymbol{v}_i^g = \boldsymbol{x}^{g,best} + F(\boldsymbol{x}_{r_1}^g - \boldsymbol{x}_{r_2}^g) \tag{6.6}$$

gdzie $\boldsymbol{x}^{g,best}$ jest najlepszym rozwiązaniem w populacji, F jest czynnikiem skalującym, losowanym z rozkładu jednostajnego z zadanego przedziału, będącego parametrem, a indeksy $r1_i, r2_i \in \{1, \ldots, N^g\}$, gdzie $r1_i \neq i$, $r2_i \neq i$ oraz $r1_i \neq r2_i$ oznaczają losowo wybranych z populacji dwóch osobników.

Pozostałe reguły algorytmu DE pozostają identyczne jak w przypadku wariantu DE/current-to-best/1, omówionego w rozdziale 3.3.

Hybrydyzacja algorytmów w M-GAPSO opiera się o wymianę inforamcji między osobnikami, stąd w przypadku PSO rozwiązanie l_i^g może należeć do innego algorytmu, w szczególności DE lub lokalnej optymalizacji metamodelem. Na podobnej zasadzie, w przypadku DE, rozwiązania $\boldsymbol{x}^{g,best}$, $\boldsymbol{x}_{r_1}^g$ oraz $\boldsymbol{x}_{r_2}^g$ mogą wskazywać na osobniki, które w danej iteracji g nie wykorzystują algorytmu DE.

6.2.2 Charakterystyka metamodeli w M-GAPSO

M-GAPSO zakłada wykorzystanie dwóch algorytmów lokalnej optymalizacji metamodelem, zwanych dalej dla uproszczenia metamodelem kwadratowym i metamodelem wielomianowym. Oba metamodele są metamodelami lokalnymi opartymi o RW, lecz różnią się zarówno swoją postacią, jak i sposobem konstrukcji zbioru uczącego \mathcal{D} . W obu przypadkach parametry metamodelu są estymowane za pomocą MNK (równanie 4.3).

Metamodel kwadratowy

Metamodel kwadratowy wykorzystuje zbiór uczący \mathcal{D} składający się k-najbliższych próbek z archiwum (względem zadanego punktu \boldsymbol{x}_i^g), gdzie jako miarę odległości przyjęto odległość Euklidesową. Postać metamodelu kwadratowego (kwadratowa RW) jest opisana następującym równaniem:

$$\hat{f}_{kw}(\boldsymbol{x}) = \sum_{d=1}^{D} \left(a_d x_d^2 + b_d x_d \right) + c$$
(6.7)

gdzie $\boldsymbol{x} = [x_1, \dots, x_D]$ jest wektorem D współrzędnych.

Znalezienie rozwiązania optymalnego \hat{x}_{kw}^* metamodelu \hat{f}_{kw} jest równoznaczne ze znalezieniem *D* niezależnych rozwiązań optymalnych następujących funkcji $\hat{f}_{kw,d}$, dla

 $d = 1, \ldots, D$:

$$\hat{f}_{kw,d}(x_d) = a_d x_d^2 + b_d x_d \tag{6.8}$$

Powyższa funkcja jest parabolą, więc swoje minimum osiąga w wierzchołku albo w jednym z dwóch punktów ograniczających przestrzeń przeszukiwań. Stąd, rozwiązanie optymalne funkcji $\hat{f}_{kw,d}(\boldsymbol{x})$ można wyznaczać w następujący sposób:

$$\hat{x}_{kw,d}^{*} = \begin{cases} -\frac{b_d}{2a_d}, & \text{if } a_d > 0 \land -\frac{b_d}{2a_d} \in [x_d^{min}, x_d^{min}] \\ x_d^{min}, & \text{if } \hat{f}(x_d^{min}) \leqslant \hat{f}(x_d^{min}) \\ x_d^{min}, & \text{if } \hat{f}(x_d^{min}) > \hat{f}(x_d^{min}) \end{cases}$$
(6.9)

gdzie x_d^{min} i x_d^{max} są dolnym i górnym ograniczeniem przestrzeni przeszukiwań w wymiarze d.

Finalnie, rozwiązanie optymalne \hat{x}_{kw}^* jest złożeniem D rozwiązań optymalnych $\hat{x}_{kw,d}^*$, co zostało pokazane w poniższym równaniu:

$$\hat{\boldsymbol{x}}_{kw}^* = [\hat{x}_{kw,1}^*, \dots, \hat{x}_{kw,D}^*] \tag{6.10}$$

Metamodel wielomianowy

Metamodel wielomianowy jest złożeniem D niezależnych metamodeli RW p-tego stopnia, które mają za zadanie odwzorować f.c. w danym wymiarze d. Każdy metamodel d wykorzystuje niezależny zbiór uczący \mathcal{D}_d , zbudowany w oparciu o k-najbliższych próbek z archiwum (względem zadanego punktu \boldsymbol{x}_i^g) w wymiarze d. Jako miarę odległości przyjęto odległość Euklidesową, lecz jest to odległość wyłącznie w jednym wymiarze (tzn. odległość między \boldsymbol{x}_i^g a $\boldsymbol{x}_{i'}^g$ w wymiarze d, to $|\boldsymbol{x}_{i,d}^g - \boldsymbol{x}_{i',d}^g|$). Porównywanie sposobu konstruowania zbiorów uczących w metamodelu kwadratowym oraz metamodelu wielomianowym zaprezentowano na rysunku 6.1.

Metamodel wielomianowy (RW p-tego stopnia) estymowany dla wymiaru d jest opisany następującym równaniem:

$$\hat{f}_{wiel,d}(x_d) = \sum_{i=1}^{p} a_{i,d} x_d^i + c_d$$
(6.11)

Rozwiązanie optymalne $\hat{x}^*_{wiel,d}$ metamodelu $\hat{f}_{wiel,d}$ jest wyznaczane za pomocą przeszukiwania siatkowego (ang. grid search). Przedział $[x^{min}_d, x^{max}_d]$ jest dzielony na 1000



Rysunek 6.1: Porównywanie sposobu konstruowania zbiorów uczących w metamodelu kwadratowym oraz metamodelu wielomianowym.

równomiernie rozłożonych punktów $x_{j,d}$ i dla każdego z tych punktów jest liczona wartość $\hat{f}_{wiel,d}(x_{j,d})$. Punkt $x_{j,d}$, dla którego wartość $\hat{f}_{wiel,d}(x_{j,d})$ była najmniejsza stanowi rozwiązanie optymalne $\hat{x}^*_{wiel,d}$ dla wymiaru d. Analogicznie jak w przypadku metamodelu kwadratowego finalne rozwiązanie optymalne \hat{x}^*_{wiel} jest złożeniem D rozwiązań optymalnych $\hat{x}^*_{wiel,d}$, co zostało pokazane w poniższym równaniu:

$$\hat{\boldsymbol{x}}_{wiel}^* = [\hat{x}_{wiel,1}^*, \dots, \hat{x}_{wiel,D}^*]$$
(6.12)

6.2.3 Eksperymentalna ewaluacja

Eksperymentalnej ewaluacji M-GAPSO dokonano przy użyciu zbioru testowego COCO oraz towarzyszącej mu procedury testowej (rozdział 2.4.4).

Trzy warianty M-GAPSO wzięły udział w porównaniu: DE, PD oraz PDLP. Wariant DE, to M-GAPSO z jednym algorytmem, jakim jest DE. Brak w nim inicjalizacji metamodelem. Wariant PD, to hybryda DE oraz PSO bez adaptacji wektora wag p oraz bez inicjalizacji metamodelem. Wariant PDLP rozszerza wariant PD o inicjalizację metamodelem oraz dodaje dwa algorytmy wykorzystujące lokalną optymalizację metamodelem, odpowiednio kwadratowym i wielomianowym (p = 4). W obrębie wspólnych funkcjonalności dla wszystkich wariantów M-GAPSO zachowano tę samą parametryzację. Pełna parametryzacja, tj. dla wariantu PDLP, jest przedstawiona w tabeli 6.1

Tabela 6.1: Parametry M-GAPSO.

Główne parametry M-GAPSO

Rozmiar populacji N	10D
Waga wyboru PSO	1000
Waga wyboru DE	1000
Waga wyboru metamodelu kwadratowego	1
Waga wyboru metamodelu wielomianowego	1
Adaptacja wag wyboru w_a	Nie
Parametry poszczególnych algorytmów	
PSO czynnik poznawczy c_1	1.4
PSO czynnik społeczny c_2	1.4
PSO czynnik bezwładności ω	0.64
DE prawdopodobieństwo krzyżowani a ${\cal CR}$	0.9
DE czynnik skalujący ${\cal F}$	0.0 - 1.4
Rozmiar archiwum próbek ${\cal N}_a$	$2\cdot 10^5$
Liczba próbek k w metamodelu kwadratowym	5D
Stopień wielomianu p	4
Liczba próbek k w metamodelu wielomianowym	4D + 1

Wyniki eksperymentalnej ewaluacji dla wszystkich 24 funkcji (f1 - f24) łącznie, w podziale na liczbę wymiarów $D \in \{2, 3, 5, 10, 20, 40\}$, są zaprezentowane na rysunku 6.2. Brak jest wyników wariantu DE dla D = 40, lecz nie umniejsza to wartości przedstawionego porównania, ponieważ DE pełni jedynie punkt odniesienia dla właściwego porównania PD oraz PDLP. Eksperymenty dla wariantów PD oraz PDLP wykorzystywały budżet optymalizacji $10^6 \cdot D$ ewaluacji f.c., a dla wariantu DE jest to $10^5 \cdot D$ ewaluacji f.c.

Dla wszystkich wymiarów biorących udział w zestawieniu, PD okazał się lepszy od DE, co pokazuje, że zasadność połączenia dwóch algorytmów znalazła swoje potwierdzenie w finalnych rezultatach.

Wyniki dla wymiarów D = 2 oraz D = 3 są zbliżone dla wszystkich wariantów M-GAPSO, choć należy zauważyć wyraźną poprawę wyników w pierwszej fazie optymalizacji dzięki wykorzystaniu inicjalizacji metamodelem w wariancie PDLP. Inicjalizacja



Rysunek 6.2: Wyniki (na zbiorze testowym COCO) M-GAPSO w wariantach: DE, PD oraz PDLP dla wszystkich 24 funkcji dla $D \in \{2, 3, 5, 10, 20, 40\}$, w budżecie optymalizacji $10^6 \cdot D$ ewaluacji f.c.
metamodelem poprawia wyniki również dla pozostałych wymiarów biorących udział w zestawieniu. Biorąc pod uwagę, że ewaluacja miała miejsce w budżecie optymalizacji taniej, większe znaczenie mają wyniki uzyskane w ostatniej fazie optymalizacji.

W pozostałych wymiarach, tj. $D \in \{5, 10, 20, 40\}$ PDLP jest lepszy od PD, zwłaszcza kiedy porównać ostatnią fazę procesu optymalizacji. Pokazuje to, że lokalna optymalizacja metamodelem sprawdza się w sytuacji, gdy populacja znajduje się w okolicy minimum globalnego. Różnica na korzyść PDLP względem PD jet szczególnie widoczna dla D = 10 oraz D = 20.

Wyniki w podziale na grupy funkcji

Zaprezentowane łączne (dla funkcji $f_1 - f_{24}$) wyniki potwierdziły przypuszczenie, ze lokalna optymizacja metamodelem znajdzie zastosowanie w optymalizacji taniej. Postanowiono wykonać dodatkowe porównanie, pokazujące wyniki dla D = 10 w podziale na kategorie funkcji, co zostało zaprezentowane na rysunku 6.3.

Przedstawione wykresy pokazują, że zysk z lokalnej optymalizacji metamodelem jest szczególnie widoczny dla funkcji separowalnych $(f_1 - f_5)$. Biorąc pod uwagę konstrukcje metamodelu kwadratowego oraz wielomianowego, tj. modelowanie f.c. niezależnie dla wszystkich wymiarów, jest to zgodne z oczekiwaniami.

Nieduża przewaga wariantu PDPL nad wariantem PD jest jeszcze widoczna dla funkcji ze słabym lub średnim uwarunkowaniem ($f_6 - f_9$). Również jest to zrozumiałe, ponieważ funkcje $f_6 - f_9$ charakteryzują się silną strukturą globalną, a minima lokalne, które w nich występują, nie mają relatywnie dużej amplitudy.

Brak jakiejkolwiek poprawy wyniku, ze wskazaniem na lekkie jego pogorszenie, jest obserwowany dla funkcji f_{15} - f_{24} , które charakteryzują się wielomodalnością, a liczba minimów lokalnych jest znacząca.

Alternatywny sposób konstrukcji zbioru uczącego w metamodelu wielomianowym

Postanowiono wykonać dodatkowy eksperyment sprawdzający wyniki wariantu PDLP przy założeniu, że w metamodelu wielomianowym D zbiorów uczących \mathcal{D}_d byłoby tworzonych analogicznie jak w metamodelu kwadratowym, tj. za pomocą klasycznej miary odległości Euklidesowej w przestrzeni \mathbb{R}^D . Wyniki dla D = 10 dla wszystkich funkcji łącznie (lewy wykres) oraz dla funkcji separowalnych (prawy wykres) przedstawiono na rysunku 6.4.



Rysunek 6.3: Wyniki (na zbiorze testowym COCO) M-GAPSO w wariantach: DE, PD oraz PDLP dla wszystkich 24 funkcji w podziale na kategorie funkcji dla D = 10, w budżecie optymalizacji $10^6 \cdot D$ ewaluacji f.c.



Rysunek 6.4: Wyniki (na zbiorze testowym COCO) M-GAPSO w wariancie PDLP oraz PDLPk dla wszystkich 24 funkcji (wykres lewy) oraz funkcji separowalnych f_1 - f_5 (wykres prawy) dla D = 10 w budżecie optymalizacji $10^6 \cdot D$ ewaluacji f.c.

Zaprezentowane zestawienie pokazuje, że uproszczenie sposobu konstrukcji zbioru uczącego w metamodelu wielomianowym pogarsza całościowe wyniki dla D = 10. Co więcej, dla funkcji separowalnych, różnica jest znacząca na korzyść przedstawionego pierwotnie sposobu konstrukcji zbiorów uczących.

6.3 SHADE-LM: R-SHADE z lokalną optymalizacją metamodelem

Obiecujące wyniki lokalnej optymalizacji metamodelem w M-GAPSO sprawiły, że zaczęto poszukiwać jej zastosowania w adaptacyjnych algorytmach z rodziny DE. Skutkiem tego powstał SHADE-LM [164], czyli R-SHADE (rozdział 3.3.5) rozszerzonym o lokalną optymalizację metamodelem. SHADE-LM jest zbudowany w oparciu o M-GAPSO, tzn. zarówno R-SHADE, jak i metamodel są algorytmami, które są losowo przypisywane do każdego osobnika i, w każdej iteracji g.

SHADE-LM, podobnie jak PDLP wykorzystuje mechanizm inicjacji metamodelem (rozdział 5.2.1). Reguły restartowania populacji pochodzą z R-SHADE. Adaptacja wektora wag w_a nie jest stosowana. Faza selekcji dotyczy zarówno osobników należących do R-SHADE, jak i metamodelu.

Analogicznie jak w przypadku algorytmu DE w PDLP, R-SHADE nie dokonuje rozróżnienie osobników ze względu na rodzaj algorytmu, jaki mają przypisany w danej iteracji. W następstwie, wylosowane w fazie mutacji rozwiązania $\boldsymbol{x}_{pbest_i}^g$, $\boldsymbol{x}_{r1_i}^g$ oraz $\boldsymbol{x}_{r2_i}^g$ (równanie 3.4) mogą wskazywać na osobników, które w danej iteracji g stosują algorytm metamodelu. Odpowiednio, osobniki, które poprawiły swoje rozwiązanie w oparciu o wskazanie metamodelu trafiają do zewnętrznego archiwum A, wykorzystywanego przez SHADE (rozdział 3.3.3). Wysokopoziomowe spojrzenie na SHADE-LM jest zaprezentowane za pomocą pseudokodu 11.

Algorytm 11 SHADE-LM
$\frac{11 \text{gory our } 11 \text{ Simular Barrier}}{1: \text{ Ustaw parametry } N, M_F, M_{CB}, p, a, H}$
2: while budżet optymalizacji nie jest wyczerpany do
3: Inicializuj wpisy w pamieci M_{EL}^0 oraz M_{CRL}^0 za pomoca domyślnych wartości M_F oraz M_{CR}
4: Wylosuj początkową populację $P^0 = [\boldsymbol{x}_1^0, \dots, \boldsymbol{x}_{N^0}^0]$, gdzie $N^0 = N$, za pomocą inicjalizacji meta-
modelem (pseudokod 5)
5: $g = 0$
6: while budżet (dostępna liczba ewaluacji f.c.) nie jest wyczerpany do
7: Dla każdego osobnika i wylosuj algoryt m k zgodnie z wektorem prawdopodobieństwa \pmb{w}_a
8: for $i = 1 \text{ do } N^g \mathbf{do}$
9: if Osobnik <i>i</i> ma algorytm SHADE then
10: Wyznacz nowy wektor próby \boldsymbol{u}_i^g , zgodnie logiką SHADE
11: end if
12: if Osobnik <i>i</i> ma algorytm metamodelu then
13: Wyznacz nowy wektor próby $\boldsymbol{u}_i^g = \hat{\boldsymbol{x}}^*$, zgodnie logiką metamodelu
14: end if
15: Wyznacz wartość f.c. $f(\boldsymbol{u}_i^g)$
16: Dokonaj selekcji \boldsymbol{u}_i^g , zgodnie z logiką SHADE 3.6
17: end for
18: Zaktualizuj archiwum A wykorzystując zastąpione wektory macierzyste \boldsymbol{x}_{i}^{g}
19: Zaktualizuj wpis w pamięci za pomocą $M_{F,k}^g$ oraz $M_{CB,k}^g$ zgodnie z równaniem (3.7)
20: if restart wymagany then break
21: end if
$22: N^{g+1} = N^g$
23: g = g + 1
24: end while
25: end while

6.3.1 Charakterystyka metamodelu w SHADE-LM

W SHADE-LM metamodel jest uczony w oparciu o bieżącą populację P^g oraz odpowiednie wartości f.c. Wobec tego, nie ma potrzeby stosowania archiwum próbek. SHADE-LM wykorzystuje kaskadę dwóch metamodeli: kwadratowej RW oraz kwadratowej RW z interakcjami (tabela 6.2). W przypadku, gdy rozmiar populacji umożliwia stosowanie kwadratowej RW z interakcjami $(N^g \ge (D^2 + 3D)/2 + 1)$, to jest on stosowany. Jeśli rozmiar populacji nie jest wystarczający, to mniej złożony metamodel (kwadratowa RW) jest stosowany. Celem uproszczenia nazewnictwa, kwadratowa RW jet nazywana metamodelem kwadratowym, a kwadratowa RW z interakcjami, metamodelem pełnym.

Tabela 6.2: Opis kaskady metamodeli stosowanych w SHADE-LM. Wykorzystywane metamodele to: kwadratowa RW (ozn. jako kw. RW) oraz kwadratowa RW z interakcjami (ozn. jako kw. RW + in.). Estymowane współczynniki przy każdej ze zmiennych zostały pominięte dla zwiększenia czytelności.

Nazwa	Postać	Stopnie swobody
Kw. RW	$\tilde{\boldsymbol{x}}_{kw} = [\tilde{\boldsymbol{x}}_{lin}, x_1^2, \dots, x_D^2]$	$df_{kw} = 2D + 1$
Kw. RW $+$ in.	$\tilde{\boldsymbol{x}}_{kwin} = [\tilde{\boldsymbol{x}}_{kw}, x_1 x_2, x_1 x_3, \dots, x_{D-1} x_D]$	$df_{kwin} = \frac{D^2 + 3D}{2} + 1$

Wyznaczenie rozwiązania optymalnego \hat{x}_{kw}^* metamodelu kwadratowego jest realizowane tak samo jak w wariancie PDLP algorytmu M-GAPSO (równania 6.8 oraz 6.10).

Wyznaczenie rozwiązania optymalnego \hat{x}_{pel}^* metamodelu pełnego wymaga wyznaczenia pochodnych pierwszego rzędu \hat{f}_{pel} . W tym celu metamodel pełny jest przedstawiony jako:

$$\hat{f}_{pel}(\boldsymbol{x}) = \sum_{d=1}^{D} \left(b_d x_d + \sum_{d'=1}^{d} a_{d,d'} x_d x_{d'} \right)$$
(6.13)

gdzie $\boldsymbol{x} = [x_1, \dots, x_D]$ jest wektorem D współrzędnych.

Parametry $a_{d,d'}$ oraz b_d są obliczane z wykorzystaniem MNK (równanie 4.3). Następnie, wyznaczany jest punkt stacjonarny $\boldsymbol{x}^{\theta} = [x_1^{\theta}, \dots x_D^{\theta}]$, jako rozwiązanie poniższego układu równań:

$$\begin{bmatrix} 2a_{1,1} & \dots & 2a_{D,1} \\ \vdots & \ddots & \vdots \\ 2a_{D,1} & \dots & 2a_{D,D} \end{bmatrix} \begin{bmatrix} x_1^{\theta} \\ \vdots \\ x_D^{\theta} \end{bmatrix} = \begin{bmatrix} -b_1 \\ \vdots \\ -b_D \end{bmatrix}$$
(6.14)

Wyznaczony punkt stacjonarny \boldsymbol{x}^{θ} jest przyjmowany za wektor próby \boldsymbol{u}_{i}^{g} , który jest poddawany selekcji. W ogólności, punkt \boldsymbol{x}^{θ} nie musi stanowić rozwiązania optymalnego metamodelu \hat{f}_{pel} . Niemniej, nie jest to sprawdzane, ze względów wydajnościowych oraz dla zapewnienia przejrzystości implementacji.

6.3.2 Eksperymentalna ewaluacja

Analogicznie jak w przypadku M-GAPSO, eksperymentalna ewaluacja jest wykonana za pomocą zbioru testowego COCO oraz towarzyszącej mu procedury testowej (rozdział 2.4.4). Porównano SHADE-LM oraz bazowy R-SHADE. Parametryzacja R-SHADE została zaczerpnięta z [216]. SHADE-LM dzieli z nią wszystkie parametry dotyczące bazowego R-SHADE, z wyłączeniem rozmiaru populacji, który musiał być zwiększony ze względu na obecność lokalnej optymalizacji metamodelem. Pełna SHADE-LM jest przedstawiona w tabeli 6.3

Tabela 6.3: Parametry SHADE-LM.

Parametry ogólne	
Rozmiar populacji ${\cal N}$	10D
Waga wyboru SHADE	0.95
Waga wyboru metamodelu	0.05
Adaptacja wag wyboru w_a	Nie
Parametry SHADE	
Inicjalny M_F	0.38
Inicjalny M_{CR}	0.9
Mnożnik najlepszych \boldsymbol{p}	0.11
Mnożnik archiwum \boldsymbol{a}	0.12
Rozmiar pamięci ${\cal H}$	11

Wyniki eksperymentalnej ewaluacji dla wszystkich 24 funkcji (f1 - f24) łącznie, w podziale na liczbę wymiarów $D \in \{2, 5, 10, 20\}$ są zaprezentowane na rysunku 6.5. Założony budżet ewaluacji to $10^6 \cdot D$. Wyniki dla bazowego R-SHADE są ograniczone do $10^5 \cdot D$.

Inicjalizacja metamodelem poprawia wynik pierwszej fazy optymalizacji dla wszystkich zaprezentowanych wymiarów. Patrząc na cały przebieg procesu optymalizacji, dla D = 2 oba algorytmy osiągają zbliżone wyniki. W przypadku D = 5 SHADE-LM jest już nieznacznie lepszy od R-SHADE. Większe liczby wymiarów, tj. D = 10 oraz D = 20 pokazują, że zysk z użycia lokalnej optymalizacji metamodelem staje się znaczący. SHADE-LM pod koniec budżetu optymalizacji osiąga założone cele istotnie szybciej niż R-SHADE.

Warto zaznaczyć, że SHADE-LM, bazujący na M-GAPSO, prezentuje bardzo prosty sposób integracji lokalnej optymalizacji metamodelem. Uzyskane wyniki są satysfakcjo-



Rysunek 6.5: Wyniki (na zbiorze testowym COCO) SHADE-LM w zestawieniu z wynikami bazowego R-SHADE dla wszystkich 24 funkcji w podziale na kategorie funkcji dla $D \in \{2, 5, 10, 20\}$ w budżecie optymalizacji $10^6 \cdot D$ ewaluacji f.c.

nujące. Dodanie archiwum próbek oraz udoskonalenie sposobu wyznaczania rozwiązania optymalnego metamodelu pełnego może jeszcze zwiększyć skuteczność SHADE-LM.

6.4 Narzut obliczeniowy lokalnej optymalizacji metamodelem

Zgodnie z procedurą pomiaru empirycznej złożoności obliczeniowej, znaną z CEC2021 (rozdział 2.4.3) dokonano pomiarów czasów działania M-GAPSO. Dodatkowo, dla algorytmu SHADE-LM dokonano pomiaru czasu obliczeń przypadającego na pojedynczą ewaluację f.c. Badania wykonano za pomocą oprogramowania napisanego w języku Java, działającym na systemie Windows 10 z wykorzystaniem CPU Intel Core i7-9750H (2.60Ghz). W tabeli 6.4 zaprezentowano wyniki pomiaru empirycznej złożoności obliczeniowej M-GAPSO w wariantach: DE, PDa oraz PDLPa. Warianty PDa oraz PDLPa odpowiadają wariantom PD oraz PDLP (rozdział 6.2.3, lecz wykorzystują adaptację wag \boldsymbol{w}_a , dzięki czemu zmierzono wariant bardziej złożony obliczeniowo. Sprawdzono empiryczną złożoność obliczeniową dla wymiarów $D \in \{10, 30, 50, 100\}$.

W wynikowej tabeli, T_0 jest czasem działania testowego programu, T_1 jest czasem $2 \cdot 10^5$ ewaluacji funkcji f_{18} z CEC2017, T_2 jest średnim czasem działania algorytmu obejmującego $2 \cdot 10^5$ ewaluacji f.c, a $T_3 = (T_2 - T_1)/T_0$ jest finalną empiryczną złożonością obliczeniową.

Tabela 6.4: Empiryczna złożoność obliczeniowa M-GAPSO w wariantach DE, PDa oraz PDLPa, mierzona zgodnie z procedurą CEC2021 (rozdział 2.4.3). T_0 jest czasem działania testowego programu, T_1 jest czasem $2 \cdot 10^5$ ewaluacji funkcji f_{18} z CEC2017. T_2 jest średnim czasem działania algorytmu obejmującego $2 \cdot 10^5$ ewaluacji f.c. $T_3 = (T_2 - T_1)/T_0$ jest finalną empiryczną złożonością obliczeniową.

D	Algorytm	$T_0[s]$	$T_1[s]$	$T_2[s]$	T_3
	DE			1.15	330
10	PDa	0.00205	0.91	1.39	415
10	PDLPa	0.00265	0.21	8.48	2903
	DE			1.58	385
20	PDa	0.00205	0.40	1.71	458
30	PDLPa	0.00265	0.49	9.35	3113
	DE			2.82	991
50	PDa	0.00905	0.00	3.81	1020
90	PDLPa	0.00285	0.90	17.12	5692
	DE			13.39	3702
100	PDa		0.04	12.95	3548
	PDLPa	0.00280	2.84	51.45	17061

Zaprezentowane wyniki pokazują, że czasy działania DE oraz PDa należy uznać za zbliżone. Zastosowanie lokalnej optymalizacji metamodelem (PDLPa) zwiększa czas działania algorytmu, jednak nie jest to różnica uniemożliwiająca przeprowadzenie procesu optymalizacji w racjonalnym czasie. Dla wszystkich badanych wymiarów D obserwuje się wzrosty czasu działania w przybliżeniu między 4 a 6 razy. Analizując najbardziej złożony przypadek (D = 100) można zauważyć, że czas całego procesu optymalizacji wzrósł z blisko 13s do ponad 51s. Czas samych obliczeń PDLPa, rozumiany jako $T_2 - T_1$ wynosi w tym przypadku prawie 49s. Przeliczając ten czas na pojedynczą ewaluację f.c. otrzymuje się wartość $2.4 \cdot 10^{-4}s$. W przypadku braku wykorzystanie metamodeli (PDa) dostajemy odpowiednio $5.0 \cdot 10^{-5}s$. W przypadku najmniej złożonego problemu, tj. dla D = 10, czas samych obliczeń PDL-Pa przypadający na pojedynczą ewaluację f.c. wynosi $4.1 \cdot 10^{-5}s$. Zdaniem autora, taki wzrost czasu obliczeń jest w pełni akceptowalny, nawet przy założeniu relatywnie wysokich budżetów optymalizacji.

Dodatkowe badanie SHADE-LM pokazało, że średni czas (rozumiany jako T_c z równania 5.2) przypadający na pojedynczą ewaluację f.c. wynosi dla D = 2,3,5,10,20 odpowiednio $1.70 \cdot 10^{-5}s$, $1.08 \cdot 10^{-5}s$, $1.49 \cdot 10^{-5}s$, $2.42 \cdot 10^{-5}s$ oraz $5.20 \cdot 10^{-5}s$. Uzyskane wyniki czasów obliczeń dla SHADE-LM są zbliżone do tych uzyskanych dla M-GAPSO. Należy je uznać za w pełni akceptowalne w kontekście optymalizacji taniej.

ROZDZIAŁ 6. LOKALNA OPTYMALIZACJA METAMODELEM W OPTYMALIZACJI TANIEJ

Rozdział 7

Preselekcja rozwiązań w optymalizacji semikosztownej

W tym rozdziale przedstawiono badania autora rozprawy dotyczące zastosowania zastosowania preselekcji rozwiązań na podstawie wartości metamodelu w adaptacyjnych algorytmach R-SHADE oraz L-SHADE. Skutkiem tego otrzymano następujące trzy algorytmy: LQ-R-SHADE, psLSHADE oraz rmmLSHADE. Wszystkie trzy zaprezentowane algorytmy za metamodel przyjmują regresję wielomianową lub jej rozszerzenie. LQ-R-SHADE oraz rmmLSHADE zakładają użycie kwadratowej regresji wielomianowej z interakcjami. Algorytm psLSHADE dodatkowo wykorzystuje nielinowe transformacje zmiennej objaśniającej w postaci $\frac{1}{x}$ oraz $\frac{1}{x^2}$. LQ-R-SHADE oprócz preselekcji rozwiązań wykorzystuje mechanizm inicjalizacji metamodelem. Wyróżnikiem rmmLSHADE jest estymacja parametrów metamodelu po każdej ewaluacji f.c. z wykorzystaniem rekursywnego filtru najmniejszych kwadratów.

Wszystkie zaproponowane algorytmy populacyjne wspierane metamodelem zostały poddane eksperymentalnej ewaluacji. Zaprezentowano oraz przedyskutowano wpływ preselekcji rozwiązań na podstawie wartości metamodelu na czas obliczeń algorytmu.

7.1 LQ-R-SHADE: R-SHADE wspierany globalnym metamodelem

Algorytm LQ-R-SHADE [242] jest rozszerzeniem algorytmu R-SHADE, opisanego w rozdziale 3.3.5, o mechanizm lokalnej preselekcji rozwiązań. Lokalna preselekcja rozwiązań jest oparta, podobnie jak lq-CMA-ES [77], o kaskadę metamodeli: liniową RW, kwadratową RW oraz kwadratową RW z interakcjami. Dodatkowo, inicjalna populacja jest generowana zgodnie z mechanizmem inicjalizacji metamodelem. Wysokopoziomowe spojrzenie na LQ-R-SHADE jest zaprezentowane za pomocą pseudokodu 12.

Alg	gorytm 12 LQ-R-SHADE
1:	Ustaw parametry $N, M_F, M_{CR}, p, a, H, N_a, N_s$
2:	while budżet optymalizacji nie jest wyczerpany do
3:	Inicjalizuj wpisy w pamięci $M_{F,k}^0$ oraz $M_{CR,k}^0$ za pomocą domyślnych wartości M_F oraz M_{CR}
4:	Wylosuj początkową populację $P^0 = [\boldsymbol{x}_1^0, \dots, \boldsymbol{x}_{N^0}^0]$, gdzie $N^0 = N$, za pomocą inicjalizacji meta-
	modelem (pseudokod 5)
5:	g = 1
6:	while budżet optymalizacji nie jest wyczerpany \mathbf{do}
7:	Generuj $N^g \cdot N_s$ zmutowanych wektorów $\boldsymbol{v}_i^{g,j}$ zgodnie z równaniem (7.1)
8:	Generuj $N^g \cdot N_s$ wektorów próby $\boldsymbol{u}_i^{g,j}$ zgodnie z równaniem (7.2)
9:	Estymuj parametry metamodel u \hat{f} za pomocą MNK (równanie 4.3)
10:	Dla każdego osobnika i wyznacz najlepszy wektor próby $\boldsymbol{u}_{i}^{g,best}$
11:	$\mathbf{for} \ \mathbf{i} = 1 \ \mathbf{do} \ N^g \ \mathbf{do}$
12:	Dokonaj selekcji $\boldsymbol{u}_i^{g,best}$ za pomocą równania 7.3
13:	Dodaj $(\boldsymbol{u}_i^{g,best}, f(\boldsymbol{u}_i^{g,best}))$ do archiwum próbek
14:	end for
15:	Zaktualizuj archiwum A wykorzystując zastąpione wektory macierzyste \pmb{x}_i^g
16:	Zaktualizuj w pis w pamięci za pomocą $M_{F,k}^g$ oraz $M_{CR,k}^g$ zgodnie z równaniem (3.7)
17:	if restart wymagany then break
18:	end if
19:	$N^{g+1} = N^g$
20:	g = g + 1
21:	end while
22:	end while

7.1.1 Lokalna preselekcja rozwiązań w LQ-R-SHADE

Przypomnijmy, że w fazie mutacji każdy osobnik *i* losowo generuje N_s zmutowanych wektorów $\boldsymbol{v}_i^{g,j}$, gdzie $j \in \{1, \ldots, N_s\}$. N_s jest parametrem algorytmu oznaczającym mnożnik osobników. Każdy zmutowany wektor $\boldsymbol{v}_i^{g,j}$, jest tworzony w oparciu o inne wartości $F_i^{g,j}$, $r1_i$, $r2_i$ oraz $pbest_i$, zgodnie z logiką SHADE (rozdział 3.3.3). Całość opisuje poniższe równanie:

$$\boldsymbol{v}_{i}^{g,j} = \boldsymbol{x}_{i}^{g} + F_{i}^{g,j}(\boldsymbol{x}_{pbest_{i}}^{g,j} - \boldsymbol{x}_{i}^{g}) + F_{i}^{g,j}(\boldsymbol{x}_{r1_{i}}^{g,j} - \boldsymbol{x}_{r2_{i}}^{g,j})$$
(7.1)

Następnie każdy zmutowany wektor $\boldsymbol{v}_i^{g,j}$ jest przekształcany w wektor próby $\boldsymbol{u}_i^{g,j}$, przy czym wszystkie wektory próby pochodzące od *i*-tego osobnika wykorzystują te same wartości CR_i^g (równanie 3.9). Stąd, fazę krzyżowania można przedstawić w następujący sposób:

$$u_{i,d}^{g,j} = \begin{cases} v_{i,d}^{g,j}, & \text{jeśli } rand(0,1) \leqslant CR_i^g \text{ lub } d = d_{rand} \\ x_{i,d}^g, & \text{w p.p.} \end{cases}$$
(7.2)

gdzie $\forall_{j \in \{1,...,N_s\}} rand(0,1) = const.$ oraz $d_{rand} = const.$

Metamodel jest estymowany bezpośrednio przed fazą selekcji. Dla każdego osobnika i wyznaczana jest wartość $\hat{f}(\boldsymbol{u}_i^{g,j})$ dla wszystkich N_s wektorów próby, które są mu przypisane. Następnie dla każdego osobnika i wybierany jest tylko jeden wektor próby $\boldsymbol{u}_i^{g,best}$, ze względu na wartość $\hat{f}(\boldsymbol{u}_i^{g,j})$, celem ewaluacji z wykorzystaniem f. Całość sprowadza się do poniższego równania:

$$\boldsymbol{x}_{i}^{g+1} = \begin{cases} \boldsymbol{u}_{i}^{g,best}, & \text{jeśli } f(\boldsymbol{u}_{i}^{g,best}) < f(\boldsymbol{x}_{i}^{g}) \\ \boldsymbol{x}_{i}^{g}, & \text{w p.p} \end{cases}$$
(7.3)

Wykorzystanie lokalnej preselekcji rozwiązań nie wpływa ma mechanizm adaptacji parametrów: czynnika skalującego oraz prawdopodobieństwa krzyżowania. Wartości $F_i^{g,best}$ oraz $CR_i^{g,best}$, które posłużyły do wygenerowania wektora próby $\boldsymbol{u}_i^{g,best}$, są traktowane w taki sam sposób jak F_i^g oraz CR_i^g w bazowym R-SHADE. Innymi słowy, wszystkie wartości $F_i^{g,best}$ oraz $CR_i^{g,best}$, które skutkowały poprawą rozwiązania w fazie selekcji trafiają, odpowiednio, do zbiorów S_F oraz S_{CR} (rozdział 3.3.3). Pozostałe kroki są analogiczne do tych zastosowanych w mechanizmie adaptacji parametrów w SHADE.

Analogicznie, lokalna preselekcja rozwiązań nie ma wpływu na konstrukcję zewnętrznego archiwum A, ponieważ trafiają do niego wektory macierzyste \boldsymbol{x}_i^g , jeśli tylko zostały zastąpione w fazie selekcji przez wektor próby $\boldsymbol{u}_i^{g,best}$.

Archiwum próbek

Archiwum przechowuje próbki, rozumiane jako pary $(\boldsymbol{x}, f(\boldsymbol{x}))$, które uzyskano wskutek dotychczasowych ewaluacji f.c. Archiwum posiada ograniczony rozmiar N_a . Jeśli archiwum jest pełne i nowa próbka ma być do niego dodana, to najgorsza próbka, ze względu na wartość $f(\boldsymbol{x})$ jest z niego usuwana. W przypadku gdy nowa próbka jest gorsza od najgorszej próbki w archiwum, to aktualizacja archiwum nie jest wykonywana. Podobnie, jeżeli w archiwum istnieje próbka posiadająca to samo rozwiązanie (\boldsymbol{x}) co nowa próbka, to archiwum nie jest aktualizowane.

7.1.2 Charakterystyka metamodelu w LQ-R-SHADE

Lokalna preselekcja rozwiązań jest oparta o kaskadę metamodeli, zaprezentowaną w tabeli 7.1. Parametry metamodeli są estymowane za pomocą MNK (równanie 4.3). Najprostszy metamodel to liniowa RW, która wymaga do estymacji parametrów co najmniej df_{lin} obserwacji w archiwum. Następnie wykorzystywana jest kwadratowa RW, która wymaga minimum df_{kw} obserwacji. Najbardziej złożony metamodel to kwadratowa RW z interakcjami, która wymaga df_{kwin} obserwacji.

Tabela 7.1: Opis kaskady metamodeli stosowanych w preselekcji w LQ-R-SHADE. Wykorzystywane metamodele to: liniowa RW (ozn. jako lin. RW), kwadratowa RW (ozn. jako kw. RW) oraz kwadratowa RW z interakcjami (ozn. jako kw. RW + in.). Estymowane współczynniki przy każdej ze zmiennych zostały pominięte dla zwiększenia czytelności.

Nazwa	Postać	Stopnie swobody
Lin. RW	$\tilde{\boldsymbol{x}}_{lin} = [1, x_1, \dots, x_D]$	$df_{lin} = D + 1$
Kw. RW	$ ilde{m{x}}_{kw} = [ilde{m{x}}_{lin}, x_1^2, \dots, x_D^2]$	$df_{kw} = 2D + 1$
Kw. RW $+$ in.	$\tilde{\boldsymbol{x}}_{kwin} = [\tilde{\boldsymbol{x}}_{kw}, x_1 x_2, x_1 x_3, \dots, x_{D-1} x_D]$	$df_{kwin} = \frac{D^2 + 3D}{2} + 1$

7.1.3 Eksperymentalna ewaluacja

Eksperymentalnej ewaluacji LQ-R-SHADE dokonano przy użyciu zbioru testowego COCO oraz towarzyszącej mu procedury testowej (rozdział 2.4.4). Wykorzystano scenariusz kosztowny (w rozumieniu nomenklatury rozprawy - semikosztowny), który ogranicza budżet optymalizacji do $10^3 \cdot D$ ewaluacji f.c.

Dokonano porównania proponowanego LQ-R-SHADE oraz bazowego R-SHADE. Celem zapewnienia rzetelnego porównania, wszystkie parametry nie związane z lokalną preselekcją rozwiązań są takie same w obu algorytmach i ustalone na podstawie sugerowanej parametryzacji dla R-SHADE [216]. Parametryzację R-SHADE oraz LQ-R-SHADE przedstawiono w tabeli 7.2. Dodatkowo, w porównaniu uwzględniono wariant init-R-SHADE, który jest algorytmem R-SHADE z inicjalizacją metamodelem, ale bez lokalnej preselekcji rozwiązań ($N_s = 1$).

Tabela 7.2: Parametry R-SHADE i LQ-R-SHADE. $r(\cdot)$ zwraca argument zaokrąglony do najbliższej liczby całkowitej.

R-SHADE oraz LQ-R-SHADERozmiar populacji N $r(3.96 \cdot D)$ Inicjalny M_F 0.38Inicjalny M_{CR} 0.94Mnożnik najlepszych p0.09Mnożnik archiwum a0.12Rozmiar pamięci H11

Tylko LQ-R-SHADE

Rozmiar archiwum ${\cal N}_a$	$\max(N, 2df_{full})$
Mnożnik osobników ${\cal N}_s$	10

Wyniki eksperymentalnej ewaluacji dla wszystkich 24 funkcji (f1 - f24) łącznie, w podziale na liczbę wymiarów $D \in \{2,3,5,10,20,40\}$ są zaprezentowane na rysunku 7.1. W kontekście całego budżetu ewaluacji $(10^3 \cdot D)$ LQ-R-SHADE okazał się lepszy od pozostałych algorytmów dla 5, 10, 20 oraz 40 wymiarów. Dla 2 oraz 3 wymiarów wszystkie algorytmy finalnie osiągnęły podobny rezultat. Co ważniejsze, LQ-R-SHADE jest zauważalnie bardziej wydajny we wszystkich wymiarach w budżecie optymalizacji pomiędzy $10^1 \cdot D$ a $10^2 \cdot D$. Sama inicjalizacja metamodelem (wariant init-R-SHADE) poprawia wyniki w pierwszej fazie optymalizacji, jeśli porównać ją z bazowym R-SHADE. Jednakże, zastosowanie preselekcji lokalnej jest kluczowym czynnikiem poprawy skuteczności algorytmu w całym budżecie optymalizacji.

7.2 psLSHADE: LSHADE wspierany globalnym metamodelem

Algorytm psLSHADE [241] jest efektem kontynuacji prac nad zastosowaniem lokalnej preselekcji rozwiązań w adaptacyjnych algorytmach z rodziny DE. Stąd, można dostrzec pewne podobieństwa psLSHADE do LQ-R-SHADE.

Mechanizm lokalnej preselekcji rozwiązań w psLSHADE został zaimplementowany analogicznie, jak w LQ-R-SHADE (rozdział 7.1.1), więc nie będzie szerzej dyskutowany. Jednakże, w przeciwieństwie do LQ-R-SHADE, psLSHADE stanowi rozszerzenie algoryt-



Rysunek 7.1: Wyniki (na zbiorze testowym COCO) LQ-R-SHADE (ozn. jako LQ-R-SH) oraz init-R-SHADE (ozn. jako init-R-SH) w zestawieniu z wynikami R-SHADE (ozn. jako R-SH) dla wszystkich 24 funkcji dla $D \in \{2, 3, 5, 10, 20, 40\}$, w budżecie optymalizacji $10^3 \cdot D$ ewaluacji f.c.

mu L-SHADE, więc rozmiar jego populacji zmniejsza się w czasie. Ponadto, w psLSHADE nie ma mechanizmu restartów, lecz zastosowano odpowiednio większy rozmiar inicjalnej populacji, celem powstrzymania algorytmu przed przedwczesnym zbiegnięciem do minimum lokalnego. W psLSHADE zrezygnowano z kaskady metamodeli na rzecz jednego metamodelu bazującego na RW. Wykorzystano kwadratową RW z interakcjami rozszerzoną o dodatkowe dwa nieliniowe komponenty. Zrezygnowano również z mechanizmu inicjalizacji metamodelem, ponieważ eksperymentalna ewaluacja LQ-R-SHADE pokazała, że ma on marginalne znaczenie biorąc pod uwagę cały budżet optymalizacji semikosztownej. Zamiast tego inicjalna populacja jest tworzona w oparciu o generator *Latin Hypercube Sampling* [84] (LHS).

Patrząc całościowo na psLSHADE, jest to algorytm mniej złożony od LQ-R-SHADE, lecz z bardziej zaawansowanym metamodelem. Jest to zgodne z ideą poszukiwania efektywnych zastosowań metamodeli, tj. poprawiających wyniki algorytmu populacyjnego przy zachowaniu jego uniwersalności.

Warto podkreślić, że uproszczona logika psLSHADE pozwoliła na wykonania dodatkowych badań nad skutecznością mechanizmu preselekcji, co zostało przedstawione w rozdziale 7.2.3.

Dla zapewnienia czytelności, wysokopoziomowe spojrzenie na psLSHADE zaprezentowano za pomocą pseudokodu 12.

7.2.1 Charakterystyka metamodelu w psLSHADE

Lokalna preselekcja rozwiązań jest oparta o metamodel będący rozszerzoną wersją RW, co zostało zaprezentowane w tabeli 7.3. Parametry metamodeli są w typowy sposób estymowane za pomocą MNK (równanie 4.3). psLSHADE wymaga dostatecznie dużego rozmiaru inicjalnej populacji, tj. $N_{init} \ge (D^2 + 7D)/2 + 1$, by móc już w pierwszej iteracji algorytmu estymować parametry metamodelu.

7.2.2 Eksperymentalna ewaluacja

Eksperymentalnej ewaluacji psLSHADE dokonano przy użyciu zbioru testowego CEC2021 oraz towarzyszącej mu procedury testowej (rozdział 2.4.3). Założono trzy scenariusze semikosztowne poprzez ograniczenie budżetu optymalizacji do $10^2 \cdot D$, $10^3 \cdot D$ oraz $10^4 \cdot D$ ewaluacji f.c. Budżet $10^3 \cdot D$ jest podstawowym scenariuszem semikosztownym, a pozostałe dwa budżety mają reprezentować odpowiednio dolne oraz górne ograniczenie budżetu

ROZDZIAŁ 7. PRESELEKCJA ROZWIĄZAŃ W OPTYMALIZACJI SEMIKOSZTOWNEJ

Algorytm 13 psLSHADE
1: Ustaw parametry $N_{init}, N_{min}, M_F, M_{CR}, p, a, H, N_a, N_s$
2: Inicjalizuj wpisy w pamięci $M_{F,k}^0$ oraz $M_{CR,k}^0$ za pomocą domyślnych wartości M_F oraz M_{CR}
3: Wylosuj początkową populację $P^0 = [\boldsymbol{x}_1^0, \dots, \boldsymbol{x}_{N^0}^0]$, gdzie $N^0 = N_{init}$, za pomocą LHS [84]
4: $g = 1$
5: while budżet optymalizacji nie jest wyczerpany do
6: Generuj $N^g \cdot N_s$ zmutowanych wektorów $\boldsymbol{v}_i^{g,j}$ zgodnie z równaniem (7.1)
7: Generuj $N^g \cdot N_s$ wektorów próby $\boldsymbol{u}_i^{g,j}$ zgodnie z równaniem (7.2)
8: Estymuj parametry metamodel u \hat{f} za pomocą MNK (równanie 4.3)
9: Dla każdego osobnika i wyznacz najlepszy wektor próby $\boldsymbol{u}_i^{g,best}$
10: for $i = 1$ do N^g do
11: Dokonaj selekcji $\boldsymbol{u}_i^{g,best}$ za pomocą równania 7.3
12: Dodaj $(\boldsymbol{u}_i^{g,best}, f(\boldsymbol{u}_i^{g,best}))$ do archiwum próbek
13: end for
14: Zaktualizuj archiwum A wykorzystując zastąpione wektory macierzyste \pmb{x}_i^g
15: Zaktualizuj wybrany wpis w pamięci za pomocą $M_{F,k}^g$ oraz $M_{CR,k}^g$ zgodnie z równaniem (3.7)
16: Ustal nowy rozmiar populacji N^{g+1} zgodnie z równaniem 3.10
17: $g = g + 1$
18: end while

semikosztownego, czyli być na granicy optymalizacji kosztownej oraz taniej.

psLSHADE porównano z bazowym L-SHADE oraz MadDE [26]. Implementacja (wraz z parametryzacją) L-SHADE oraz MadDE została pobrana z [214]. Warto podkreślić, że MadDE, wykorzystując ten sam zbiór testowy CEC2021, tylko że z domyślnym budżetem optymalizacji, pokonał takie algorytmy jak AGSK [153], LSHADE [217], LSHA-DE_cnEpSin [17], j2020 [31] oraz IMODE [193] w porównaniu przedstawionym przez autorów [26].

Podobnie jak w przypadku ewaluacji LQ-R-SHADE, zadbano o rzetelne porównanie algorytmów poprzez zapewnienie identycznej parametryzacji psLSHADE oraz L-SHADE, z wyłączeniem parametrów związanych z lokalną preselekcją rozwiązań. Finalną parametryzację psLSHADE oraz L-SHADE przedstawiono w tabeli 7.4.

Wyniki dla różnych wartości mnożnika osobników

W pierwszym kroku eksperymentalnej ewaluacji sprawdzono jak różne wartości mnożnika osobników (parametr N_s) wpływają na wyniki algorytmu w podstawowym budżecie optymalizacji semikosztownej $10^3 \cdot D$. Wyniki są przedstawione w tabeli 7.5.

Najlepszą wartością okazało się $N_s = 5$. psLSHADE z $N_s = 2$ osiągnął gorszy wynik,

Tabela 7.3: Opis transformacji oraz finalnej postaci metamodelu stosowanych w psLSHADE. Estymowane współczynniki przy każdej ze zmiennych zostały pominięte dla zwiększenia czytelności.

Nazwa	Postać	Stopnie swobody
Stała	$ ilde{m{x}}_c = [1]$	$df_c = 1$
Liniowa	$\tilde{\boldsymbol{x}}_l = [x_1, \dots, x_D]$	$df_l = D$
Kwadratowa	$ ilde{m{x}}_k = [x_1^2, \dots, x_D^2]$	$df_k = D$
Interakcje	$\tilde{\boldsymbol{x}}_i = [x_1 x_2, \dots, x_{D-1} x_D]$	$df_i = \frac{D(D-1)}{2}$
Odwrotna liniowa	$ ilde{oldsymbol{x}}_{ol} = [rac{1}{x_1}, \dots, rac{1}{x_D}]$	$df_{ol} = D$
Odwrotna kwadratowa	$ ilde{oldsymbol{x}}_{ok} = [rac{1}{x_1^2}, \dots, rac{1}{x_D^2}]$	$df_{ok} = D$
Finalny metamodel	$ ilde{oldsymbol{x}}_{mm} = [ilde{oldsymbol{x}}_c + ilde{oldsymbol{x}}_l + ilde{oldsymbol{x}}_k + ilde{oldsymbol{x}}_l + ilde{oldsymbol{x}}_{ol} + ilde{oldsymbol{x}}_{ok}]$	$df_{mm} = \frac{D^2 + 7D}{2} + 1$

podobnie jak wartości $N_s = 10$ oraz $N_s = 20$. Sugeruje to, że zwiększanie liczby wektorów próby generowanych przez jednego osobnika do pewnego punktu (w tym przypadku $N_s = 5$) pozwala na poprawę wyników, lecz zbyt wysokie wartości N_s prowadzą do pogorszenia wyników algorytmu. Najgorszy w całym zestawieniu, wynik dla $N_s = 20$ pokazuje trend pogarszania się wyników w miarę wzrostu wartości N_s powyżej pewnego progu.

Należy zaznaczyć, że być może istnieją inne, lepsze wartości N_s niż te, które zostały uwzględnione w porównaniu. Niemniej, celem eksperymentu nie było strojenie parametrów pod konkretny zbiór testowy, a jedynie uchwycenie pewnej zależności osiąganych wyników od wartości mnożnika osobników N_s .

Finalnie, do porównań z L-SHADE oraz Mad
DE w trzech budżetach optymalizacji użyto wartości $N_s = 5$.

Porównanie z L-SHADE oraz MadDE

Porównanie psLSHADE ($N_s = 5$) z L-SHADE oraz MadDE zostało przeprowadzone dla budżetów optymalizacji $10^2 \cdot D$, $10^3 \cdot D$ oraz $10^4 \cdot D$ ewaluacji f.c. Wyniki zostały zaprezentowane w tabeli 7.6.

W najmniejszym budżecie $10^2 \cdot D$, psLSHADE osiągnął wynik lepszy niż L-SHADE oraz MadDE. Osiągnięta przewaga psLSHADE nad konkurentami jest znacząca, szczególnie jeśli spojrzeć na cząstkowe miary SNE oraz SR. Co więcej, wysoce skuteczny w

L-SHADE oraz psLSHADE		
Rozmiar populacji N_{init}	$18 \cdot D$	
Inicjalny M_F	0.5	
Inicjalny M_{CR}	0.5	
Mnożnik najlepszych \boldsymbol{p}	0.11	
Mnożnik archiwum \boldsymbol{a}	1.4	
Rozmiar pamięci ${\cal H}$	5	

Tabela 7.4: Parametry L-SHADE i psLSHADE.

Tylko psLSHADE

Rozmiar archiwum N_a	$\max(N, 2df_{full})$
Mnożnik osobników ${\cal N}_s$	2, 5, 10, 20

Tabela 7.5: Wyniki (na zbiorze testowym CEC2021) ps LSHADE dla różnych wartości mnożnika osobników
($N_s \in \{2, 5, 10, 20\}$) w budżecie optymalizacji 10³ · D ewaluacji f.c.

Algo Score	$N_s = 2$	$N_s = 5$	$N_{s} = 10$	$N_s = 20$
SNE	34.87	30.80	33.59	41.42
SR	101.25	93.75	133.75	171.25
Score 1	44.17	50.00	45.84	37.18
Score 2	46.30	50.00	35.05	27.37
Score	90.46	100.00	80.89	64.56

optymalizacji taniej MadDE w tym przypadku był gorszy niż bazowy L-SHADE. Mocno ograniczony budżet optymalizacji może skutkować relatywnie słabymi wynikami MadDE z powodu niestabilności niektórych z jego mechanizmów adaptacji parametrów.

W podstawowym scenariuszu zakładającym budżet $10^3 \cdot D$ ewaluacji f.c. wyniki są zbliżone do tych dla budżetu $10^2 \cdot D$. psLSHADE wciąż jest lepszy od obu konkurentów, lecz różnica nie jest już tak znacząca. MadDE pozostaje najgorszym algorytmem w zestawieniu.

W przypadku budżetu $10^4 \cdot D$ ewaluacji f.c., który jest na granicy optymalizacji semikosztownej oraz taniej, wszystkie algorytmy w zestawieniu osiągnęły podobne rezultaty. Najlepszy okazał się L-SHADE, a psLSHADE był nieznacznie gorszy. MadDE ponownie osiągnął najsłabszy rezultat, lecz różnica w finalnej wartości *Score* nie była już tak duża jak przy mniejszych budżetach optymalizacji. Tabela 7.6: Wyniki (na zbiorze testowym CEC2021) psLSHADE ($N_s = 5$) w zestawieniu z wynikami L-SHADE oraz MadDE w budżetach optymalizacji $10^2 \cdot D$, $10^3 \cdot D$ oraz $10^4 \cdot D$ ewaluacji f.c.

Algo O. b.	Score	MadDE	LSHADE	psLSHADE
	SNE	41.72	32.38	19.58
	SR	131.00	110.50	58.50
$10^2 \cdot D$	Score 1	23.46	30.23	50.00
	Score 2	22.33	26.47	50.00
	Score	45.79	56.70	100.00
	SNE	37.92	25.38	20.36
	SR	125.75	104.50	69.75
$10^3 \cdot D$	Score 1	26.85	40.10	50.00
	Score 2	27.73	33.37	50.00
	Score	54.58	73.48	100.00
	SNE	28.50	26.81	26.37
	SR	105.50	94.00	100.50
$10^4 \cdot D$	Score 1	46.26	49.18	50.00
	Score 2	44.55	50.00	46.77
	Score	90.81	99.18	96.77

Dodatkowo, dla scenariusza podstawowego $(10^3 \cdot D \text{ ewaluacji f.c.})$ wykonano test statystyczny *Manna–Whitneya*. Ocenie podlegała istotność różnicy błędów ϵ , uzyskanych przez psLSHADE oraz L-SHADE. Ocenie podlegało 100 przypadków (10 funkcji × 5 transformacji × 2 wymiary), dla których zebrano 30 wartości błędów ϵ . W 77 na 100 przypadkach różnica na korzyść psLSHADE była istotna (*p-value*=0.05). W żadnym przypadku psL-SHADE nie był istotnie gorszy niż L-SHADE.

7.2.3 Badanie skuteczności preselekcji

Zestawienie psLSHADE z L-SHADE oraz MadDE pokazało, że zysk z użycia preselekcji lokalnej zmniejsza się wraz ze wzrostem budżetu optymalizacji. Dlatego też, postanowiono

wykonać dodatkowe badania eksperymentalne sprawdzające wpływ preselekcji rozwiązań na wyniki algorytmu. W tym celu wybrano 4 funkcje (f_1 , f_2 , f_3 , f_8) ze zbioru CEC2021, różniące się strukturą. Funkcja f_1 posiada silną strukturę globalną oraz jedno minimum lokalne. Funkcja f_2 nie ma globalnej struktury oraz występuje w niej wiele minimów lokalnych o dużej amplitudzie. Funkcja f_3 charakteryzuje się silną strukturę globalną, która zawiera wiele minimów lokalnych o małej amplitudzie. Funkcja f_8 stanowi złożenie dwóch funkcji: pierwszej o silnej strukturze globalnej oraz drugiej bez struktury globalnej i z wieloma minimami lokalnymi o dużej amplitudzie. Trójwymiarowe wizualizacje wybranych funkcji (D = 2) są zaprezentowane na rysunku 7.2.



Rysunek 7.2: Wizualizacje 3D funkcji (D = 2) ze zbioru testowego CEC2021: f_1 (a) , f_2 (b), f_3 (c) oraz f_8 (d). Rysunki pochodzą z pracy [152].

Utrzymano parametryzację psLSHADE, która brała udział w porównaniu z L-SHADE oraz MadDE. Dla wszystkich wybranych funkcji (f_1, f_2, f_3, f_8) w wersjach D = 20 wygenerowano wykresy zbieżności, tzn. zarejestrowano wartości błędów ϵ w 16 punktach procesu optymalizacji w budżecie optymalizacji $10^3 \cdot D$. Następnie zebrane błędy zostały uśrednione w taki sposób, że pojedyncza obserwacja jest średnią ze 150 wartości (5 transformacji × 30 powtórzeń). Wykresy zbieżności dla psLSHADE (kolor niebieski) oraz L-SHADE



(kolor czerwony) są przedstawione na rysunku 7.3.

Rysunek 7.3: Uśredniony błąd psLSHADE (niebieska linia) oraz L-SHADE (czerwona linia) dla funkcji f_1 , f_2 , f_3 oraz f_8 (D = 20) w budżecie optymalizacji $10^3 \cdot D$. Na osi x zaznaczono liczbę ewaluacji f.c., a na osi y średni błąd ϵ .

Wykresy zbieżności pokazują, zgodnie z oczekiwaniami, że użycie preselekcji rozwiązań szczególne wyraźnie poprawia wyniki dla funkcji f_1 , która jest jednomodalna. W przypadku funkcji f_2 nie obserwuje się poprawy zbieżności. Brak globalnej struktury funkcji powoduje, że metamodel nie jest w stanie dobrze przybliżyć f.c. Dla funkcji f_3 preselekcja pozwala na znaczącą poprawę zbiegania algorytmu w pierwszej fazie optymalizacji. Potem oba badane algorytmy mają podobne przebiegi. Można to wyjaśnić tym, że w miarę kolejnych ewaluacji f.c. populacja zbiega do pewnego obszaru lokalnego, w którym relacja struktury globalnej do amplitudy minimów lokalnych mocno się zmniejsza. Wyniki dla funkcji f_8 są zbliżone do wyników dla funkcji f_3 . Preselekcja pozwala na szybsze zbiegnięcie w kierunku obiecującego obszaru, lecz ostatecznie zysk z jej zastosowania nie jest znaczący.

Hiperobjętość populacji

Mierzenie zbieżności algorytmów w oparciu o uzyskany błąd jest wiodącym podejściem w ich ewaluacji. Niemniej, przebieg błędu w czasie nie tłumaczy jak mechanizm preselekcji wpływa na populację. Z tego względu przeprowadzono eksperyment, w którym mierzono hiperobjętość populacji w każdej iteracji g. Hiperobjętość h^g jest zdefiniowana jako objętość przedziału wielowymiarowego opisanego na bieżącej populacji, co zostało zaprezentowane w poniższym równaniu:

$$h^g = \prod_d^D \left(max(x_{i,d}^g) - min(x_{i,d}^g) \right)$$
(7.4)

Rysunek 7.4 przedstawia wykres uśrednionej hiperobjętości dla psLSHADE (kolor niebieski) oraz L-SHADE (kolor czerwony), uzyskany w sposób analogiczny jak wykresy zbieżności, tzn. są to uśrednione wartości ze 150 przebiegów dla funkcji f_1 , f_2 , f_3 oraz f_8 dla D = 20.

Przebiegi hiperobjętości są silnie skorelowane z przebiegami wartości błędów (rysunek 7.3). Dla funkcji f_1 , f_3 oraz f_8 widać, że uzyskiwaniu mniejszych wartości błędów przez psLSHADE w relacji do L-SHADE towarzyszy zbiegnięcie populacji, w sensie jej hiperobjętości. Widać również, że dla funkcji f_1 obserwuje się bardzo szybko zmniejszającą się hiperobjętość populacji, tak że finalnie jest ona bliska zeru. Przeciwne zjawisko można zaobserwować dla funkcji f_2 , która nie poosiada globalnej struktury, więc preselekcja rozwiązań ani nie powoduje szybszego uzyskiwania mniejszego błędu, ani nie wpływa na hiperobjętość populacji.

Precyzja metamodelu

Kolejny eksperyment miał za zadane zbadać jak zmienia się jakość preselekcji rozwiązań w psLSHADE, rozumianej jako precyzja modelu. W tym celu sprawdzano w każdej iteracji g, dla każdego osobnika i czy wskazany przez metamodel wektor próby $\boldsymbol{u}_i^{g,best}$ jest faktycznie najlepszym wektorem próby spośród wszystkich N_s wektorów próby $\boldsymbol{u}_i^{g,j}$, ze względu na wartość f.c. Stąd, w każdej iteracji g wygenerowano N^g wartości prawda albo fałsz. Biorąc pod uwagę fakt, że każda funkcja posiada 5 transformacji oraz że proces optymalizacji jest powtarzany 30 razy, to dla każdej iteracji g zebrano $N^g \times 5 \times 30$ obserwacji (prawda albo fałsz).

Udział wartości prawda we wszystkich obserwacjach otrzymanych w iteracji g jest średnią precyzją modelu. Precyzja wynosząca 1 oznacza, że metamodel zawsze wskazuje



Rysunek 7.4: Uśredniona hiperobjętość psLSHADE (niebieska linia) oraz L-SHADE (czerwona linia) dla funkcji f_1 , f_2 , f_3 oraz f_8 (D = 20) w budżecie optymalizacji $10^3 \cdot D$. Na osi x zaznaczono liczbę ewaluacji f.c., a na osi y średnią hiperobjętość.

najlepszy wektor próby. Na rysunku 7.5 przedstawiono przebieg średniej precyzji metamodelu w psLSHADE, analogicznie dla funkcji f_1 , f_2 , f_3 oraz f_8 (D = 20) w budżecie optymalizacji $10^3 \cdot D$.

Dla funkcji f_1 średnia precyzja jest bliska 1, co oznacza, że metamodel bardzo dobrze odwzorowuje f.c. W przypadku funkcji f_2 precyzja metamodelu jest słaba (około 0.2), ponieważ odwzorowanie f.c. jest dalece niedokładne. Poprzednie obserwacje dla funkcji f_3 również znajdują potwierdzenie na wykresie średniej precyzji. Metamodel początkowo jest w stanie skutecznie wskazywać wektory próby o najmniejszej wartości f.c, lecz kiedy populacja zbiega do obszarów, gdzie stosunek amplitudy minimów lokalnych do globalnej struktury jest duży, to jego precyzja drastycznie spada. W przypadku f_8 obserwuje się w przybliżeniu losowe zachowanie preselekcji rozwiązań na początku procesu optymalizacji, które potem zaczyna rosnąć i utrzymuje się na zadowalającym poziomie przez większość iteracji. Należy podkreślić, że funkcja f_8 jest funkcją złożoną, więc wzrost precyzji meta-



ROZDZIAŁ 7. PRESELEKCJA ROZWIĄZAŃ W OPTYMALIZACJI SEMIKOSZTOWNEJ

Rysunek 7.5: Uśredniona precyzja metamodelu w psLSHADE dla funkcji f_1 , f_2 , f_3 oraz f_8 (D = 20), w budżecie optymalizacji $10^3 \cdot D$. Na osi x zaznaczono liczbę ewaluacji f.c. jaka została wykonana do iteracji g włącznie, a na osi y średnią precyzję.

modelu jest prawdopodobnie spowodowany zbiegnięciem populacji do części przestrzeni rozwiązań, która posiada silną strukturę globalną. Warto zwrócić uwagę, że pod koniec procesu optymalizacji funkcji f_8 obserwuje się spadek precyzji metamodelu do wartości mniejszych niż 0.2.

Ocena skuteczności preselekcji w czasie rzeczywistym

Zaprezentowana powyżej miara precyzji metamodelu pozwala na lepsze zrozumienie działania preselekcji rozwiązań. Jednakże nie może być w praktyce stosowana do mierzenia skuteczność preselekcji w czasie rzeczywistym, ponieważ wymagałaby wykonywania nadmiernej liczby ewaluacji f.c. (ewaluacji każdego wektora próby $\boldsymbol{u}_i^{g,j}$, a nie tylko jednego $\boldsymbol{u}_i^{g,best}$). Stąd postanowiono sprawdzić, czy jest możliwe stosowanie innych miar, które pozwalają ocenić stopień dopasowania metamodelu, a tym samym jakość preselekcji, w czasie rzeczywistym bez narzutu w postaci dodatkowych ewaluacji f.c. Za pierwsze kryterium oceny przyjęto dobrze znany współczynnik determinacji $R^2 \in [0,1]$. W każdej iteracji g jego wartość jest wyznacza bezpośrednio po estymacji parametrów metamodelu. Za drugie kryterium przyjęto korelację τ -Kendalla ($\tau \in [0,1]$) mierzącą rankingową współzależność N^g wartości $f(\boldsymbol{u}_i^{g,best})$ oraz odpowiadających im oszacowań metamodelu $\hat{f}(\boldsymbol{u}_i^{g,best})$. τ -Kendalla, podobnie jak R^2 , jest wyznaczana w każdej iteracji g. Co ważniejsze, obie miary są bezkosztowe, tzn. do ich kalkulacji nie są wymagane żadne nadmiarowe ewaluacje f.c.

Przebiegi obu miar, dla badanych funkcji f_1 , f_2 , f_3 oraz f_8 (D = 20) są przedstawione na rysunku 7.6. Kolorem niebieskim oznaczono przebieg dla psLSHADE, a czerwonym dla L-SHADE. Podobnie jak przy poprzednich eksperymentach wartości przedstawione na wykresie są uśrednione, w tym przypadku na podstawie 150 (5 transformacji × 30 powtórzeń) obserwacji.



Rysunek 7.6: Uśredniona wartość R^2 (kolor czerwony) oraz τ -Kendalla (kolor niebieski) dla funkcji f_1 , f_2 , f_3 oraz f_8 (D = 20) w budżecie optymalizacji $10^3 \cdot D$. Na osi x zaznaczono liczbę ewaluacji f.c., jaka została wykonana do iteracji g włącznie, a na osi y średnie wartości R^2 oraz τ -Kendalla.

ROZDZIAŁ 7. PRESELEKCJA ROZWIĄZAŃ W OPTYMALIZACJI SEMIKOSZTOWNEJ

Przedstawione przebiegi uśrednionych miar R^2 oraz τ -Kendalla wykazują silną zależność z zaprezentowaną wcześniej miarą precyzji (rysunek 7.5). Warto zauważyć, że τ -Kendalla wydaje się być lepszą miarą, ponieważ bada faktyczną, tj. obserwowaną, skuteczność modelu, a nie jedynie stopień jego dopasowania do danych uczących, jak ma to miejsce w przypadku R^2 . Skutkiem tego, na zaprezentowanych wykresach obserwuje się wzrosty wartości R^2 , które nie idą w parze ze wzrostem τ -Kendalla, ale także innymi, wcześniej zaprezentowanymi zależnościami. Dobrym przykładem tego zjawiska jest funkcja f_8 , w której precyzja metamodelu jest przez cały czas bliska 0.2, a R^2 w końcowej fazie optymalizacji znacząco się zwiększa i finalnie osiąga wartości około 0.8.

Przedstawione miary, ze wskazaniem na τ -Kendalla wydają się być użyteczne z punktu widzenia możliwego aktywowania lub dezaktywowania mechanizmu preselekcji rozwiązań w zależności od bieżącej oceny jego skuteczności. Jednakże włączenie takiego mechanizmu wiązałoby się z koniecznością określenia dodatkowego parametru, jakim jest pewien próg skuteczności, od którego uzależniane jest wykorzystanie preselekcji.

7.3 rmmLSHADE: LSHADE wspierany rekurencyjnie estymowanym globalnym metamodelem

Algorytm rmmLSHADE [243] jest kolejnym po LQ-R-SHADE oraz psLSHADE algorytmem wykorzystującym preselekcję rozwiązań. rmmLSHADE, podobnie jak psLSHADE stanowi rozwinięcie L-SHADE. W odróżnieniu od psLSHADE preselekcja rozwiązań jest realizowana w sposób globalny, co jest motywowane dalszym upraszczaniem logiki preselekcji rozwiązań. Za metamodel przyjęto kwadratową RW wraz z interakcjami. Estymacja parametrów metamodelu jest realizowana za pomocą filtra RLS (rozdział 5.2.4), skutkiem czego algorytm nie wykorzystuje archiwum próbek. W zakresie reszty funkcjonalności rmmLSHADE jest taki sam jak psLSHADE.

Wysokopoziomowe spojrzenie na rmmLSHADE jest zaprezentowane za pomocą pseudokodu 14.

7.3.1 Globalna preselekcja rozwiązań w rmmLSHADE

Przypomnijmy, że w algorytmie rmmLSHADE bazowa populacja zawiera N^g osobników $\boldsymbol{x}_i^g = [x_{i,1}^g, \ldots, x_{i,D}^g]$, gdzie $i = 1, \ldots, N^g$. Przed fazą mutacji populacja P^g jest rozszerzana N_s razy (N_s jest parametrem - mnożnikiem osobników) do populacji P_{ext}^g w następujący

7.3. RMMLSHADE: LSHADE WSPIERANY REKURENCYJNIE ESTYMOWANYM GLOBALNYM METAMODELEM

Algorytm 14 rmmLSHADE

1:	Ustaw parametry $N_{init}, N_{min}, M_F, M_{CR}, p, a, H, N_a, N_s$
2:	Inicjalizuj wpisy w pamięci $M_{F,k}^0$ oraz $M_{CR,k}^0$ za pomocą domyślnych wartości M_F oraz M_{CR}
3:	Wylosuj początkową populację $P^0 = [\boldsymbol{x}_1^0, \dots, \boldsymbol{x}_{N^0}^0]$, gdzie $N^0 = N_{init}$, za pomocą LHS [84]
4:	g = 0
5:	while budżet optymalizacji nie jest wyczerpany do
6:	Rozszerz populację P^g do populacj i $P^g_{ext} = [P^g, P^g, \ldots, P^g],$ gdzie $ P^g_{ext} = N_s \cdot N^g$ zgodnie z
	równaniem (7.5)
7:	Generuj $N^g \cdot N_s$ zmutowanych wektorów $\boldsymbol{v}_i^{g,j}$ zgodnie z równaniem (7.6)
8:	Generuj $N^g \cdot N_s$ wektorów próby $\boldsymbol{u}_i^{g,j}$ zgodnie z równaniem (7.7)
9:	for $i = 1$ to N^g do
10:	Oblic z $N^g \cdot N_s$ wartości $\hat{f}(\boldsymbol{u}_k^g)$
11:	Wybierz najlepszy ze względu na \hat{f} , nie wybrany dotych czas, wektor próby $\boldsymbol{u}_i^{g,best}$ spośród
	$N_s \cdot N^g$ wektorów próby $oldsymbol{u}_k^g$
12:	Ewaluuj $oldsymbol{u}_i^{g,best}$ za pomocą f.c. f
13:	Aktualizuj parametry filtra RLS $\pmb{w}_g,$ zgodnie z równaniem 5.3
14:	end for
15:	Dokonaj selekcji wszystkich N^g wybranych wektorów próby $\pmb{u}_i^{g,best}$ za pomocą równania 7.8
16:	Zaktualizuj archiwum A wykorzystując zastąpione wektory macierzyste \pmb{x}_i^g
17:	Zaktualizuj wybrany wpis w pamięci za pomocą $M_{F,k}^g$ oraz $M_{CR,k}^g$ zgodnie z równaniem (3.7)
18:	Ustal nowy rozmiar populacji ${\cal N}^{g+1}$ zgodnie z równaniem 3.10
19:	g = g + 1
20:	end while

sposób:

$$P_{ext}^{g} = [P^{g}, \dots, P^{g}] = [\boldsymbol{x}_{1}^{g}, \dots, \boldsymbol{x}_{N^{g}}^{g}, \dots, \boldsymbol{x}_{1}^{g}, \dots, \boldsymbol{x}_{N^{g}}^{g}]$$
(7.5)

gdzie $|P_{ext}^g| = N_s \cdot N^g$. W takim sam sposób rozszerza się wektor zawierający wartości f.c. odpowiadające poszczególnym rozwiązaniom.

Następnie, faza mutacji jest stosowana, tak jak w bazowym L-SHADE, niezależnie dla każdego osobnika o indeksie $k = 1, ..., N_m \cdot N^g$. Innymi słowy, każdy zmutowany wektor \boldsymbol{v}_k^g , jest tworzony w oparciu o inne wartości F_k^g , $r1_k$, $r2_k$ oraz $pbest_k$, zgodnie z logiką SHADE (rozdział 3.3.3). Całość przedstawia poniższe równanie:

$$\boldsymbol{v}_{k}^{g} = \boldsymbol{x}_{k}^{g} + F_{k}^{g}(\boldsymbol{x}_{pbest_{k}}^{g} - \boldsymbol{x}_{k}^{g}) + F_{k}^{g}(\boldsymbol{x}_{r1_{k}}^{g} - \boldsymbol{x}_{r2_{k}}^{g})$$
(7.6)

Następnie każdy zmutowany wektor \boldsymbol{v}_k^g jest przekształcany w wektor próby \boldsymbol{u}_k^g , używając niezależnie wygenerowanej wartości d_{rand} oraz CR_k^g (równanie 3.9):

$$u_{k,d}^g = \begin{cases} v_{k,d}^g, & \text{jeśli } rand(0,1) \leqslant CR_i^g \text{ lub } d = d_{rand} \\ x_{k,d}^g, & \text{w p.p.} \end{cases}$$
(7.7)

Po fazie krzyżowania następuje pętla zawierająca cztery kroki i wykonująca się N^g razy celem ewaluacji N^g kolejnych najlepszych wektorów próby $\boldsymbol{u}_i^{g,best}$ $(i = 1, ..., N^g)$. W pierwszym kroku wyznaczane są, przy udziale metamodelu, wartości \hat{f} wszystkich wektorów próby \boldsymbol{u}_k^g . W drugim kroku wybierany jest najlepszy, lecz nie wybrany dotychczas wektor próby $\boldsymbol{u}_i^{g,best}$. W trzecim kroku jest on poddawany ewaluacji z wykorzystaniem f.c. W czwartym kroku parametry metamodelu są aktualizowane za pomocą filtra RLS (równanie 5.3), wykorzystując nową obserwację $(\boldsymbol{u}_i^{g,best}, f(\boldsymbol{u}_i^{g,best}))$.

Faza selekcji przebiega analogicznie jak w L-SHADE. Jej wynikiem jest ponownie populacja P^g o rozmiarze N^g . Dla porządku fazę selekcji przedstawiono poniżej:

$$\boldsymbol{x}_{i}^{g+1} = \begin{cases} \boldsymbol{u}_{i}^{g,best}, & \text{jeśli } f(\boldsymbol{u}_{i}^{g,best}) < f(\boldsymbol{x}_{i}^{g}) \\ \boldsymbol{x}_{i}^{g}, & \text{w p.p.} \end{cases}$$
(7.8)

Podobnie jak w przypadku psLSHADE preselekcja rozwiązań nie wypływa na mechanizm adaptacji parametrów. Niewielka różnica względem bazowego L-SHADE dotyczy zewnętrznego archiwum A. W przypadku rmmLSHADE, powiększona populacja P_{ext}^g jest rozszerzana o archiwum A o rozmiarze $a \cdot N^g$, gdzie a jest parametrem. Pozostałe reguły wykorzystania zewnętrznego archiwum pozostają identyczne. Składa się ono z wektorów macierzystych \boldsymbol{x}_k^g , które odpowiadają za wygenerowanie wektorów próby $\boldsymbol{u}_i^{g,best}$, które poprawiły rozwiązanie w fazie selekcji. Losowe wybrane elementy są z niego usuwane w przypadku gdy jest pełne oraz jego rozmiar jest zmniejszany w przypadku redukcji rozmiaru populacji.

7.3.2 Charakterystyka metamodelu w rmmLSHADE

Algorytm rmmLSHADE wykorzystuje kwadratową RW z interakcjami. Żadne inne transformacje zmiennej objaśniającej nie są stosowane. Rezygnacja z transformacji $\frac{1}{x}$ oraz $\frac{1}{x^2}$, zawartych w metamodelu w psLSHADE jest motywowana ideą stosowania metamodeli w możliwie prosty sposób oraz obserwacją, że takie przekształcenia mogą powodować błędy numeryczne w filtrze RLS. Takie zjawisko było obserwowane w sytuacji, gdy wartości zmiennej objaśniającej są bliskie zeru.

7.3. RMMLSHADE: LSHADE WSPIERANY REKURENCYJNIE ESTYMOWANYM GLOBALNYM METAMODELEM

Postać metamodelu kwadratowej RW z interakcjami, stosowanego w rmmLSHADE, została dla czytelności zaprezentowana w tabeli 7.7. Inicjalne parametry metamodelu są wyznaczane w oparciu o MNK (równanie 4.3), wykonywaną na inicjalnej populacji P^0 oraz odpowiadających jej wartościach f.c. Stąd, inicjalny rozmiar populacji N_{init} musi wynosić co najmniej $(D^2 + 3D)/2 + 1$.

Uzyskany za pomocą MNK wektor parametrów metamodelu stanowi inicjalny wektor \boldsymbol{w}_0 w filtrze RLS. Inicjalna macierz Q_0 jest macierzą jednostkową.

Warto podkreślić, że w przeciwieństwie do LQ-R-SHADE oraz psLSHADE, metamodel jest douczany każdą nową próbką pochodzącą z ewaluacji f.c. Dzieje się tak nawet w przypadku, gdy ewaluowane rozwiązanie uzyskuje relatywnie słabą wartość f.c. Niemniej, stanowi to kolejny krok w kierunku uproszczeniu sposobu integracji metamodelu.

Tabela 7.7: Opis metamodelu kwadratowej RW z interakcjami (ozn. jako kw. RW + in.) używanego w rmmLSHADE. Estymowane współczynniki przy każdej ze zmiennych zostały pominięte dla zwiększenia czytelności.

Nazwa	Postać	Stopnie swobody
Lin. RW	$\tilde{\boldsymbol{x}}_{lin} = [1, x_1, \dots, x_D]$	$df_{lin} = D + 1$
Kw. RW	$\tilde{\boldsymbol{x}}_{kw} = [\tilde{\boldsymbol{x}}_{lin}, x_1^2, \dots, x_D^2]$	$df_{kw} = 2D + 1$
Kw. $RW + in$.	$\tilde{\boldsymbol{x}}_{kwin} = [\tilde{\boldsymbol{x}}_{kw}, x_1 x_2, x_1 x_3, \dots, x_{D-1} x_D]$	$df_{kwin} = \frac{D^2 + 3D}{2} + 1$

7.3.3 Eksperymentalna ewaluacja

Algorytm rmmLSHADE został poddany standardowej ewaluacji przy użyciu zbioru testowego CEC2021 (rozdział 2.4.3). Przyjęto budżet optymalizacji wynoszący $10^3 \cdot D$ ewaluacji f.c. rmmLSHADE został porównany z bazowym L-SHADE oraz psLSHADE.

W typowy sposób zapewniono identyczną parametryzację rmmLSHADE oraz L-SHADE, z wyłączeniem parametrów związanych z globalną preselekcją rozwiązań. Utrzymano taki sam zbiór parametrów dla L-SHADE oraz psLSHADE, jak w przypadku ich porównania (rozdział 7.2.2). Finalną parametryzację rmmLSHADE oraz L-SHADE przedstawiono w tabeli 7.8.

Porównanie rmmLSHADE z L-SHADE oraz psLSHADE zostało przeprowadzone dla budżetu optymalizacji $10^3 \cdot D$ ewaluacji f.c. W porównaniu uwzględniono 3 warianty

L-SHADE oraz rillinghade				
Rozmiar populacji N_{init}	$18 \cdot D$			
Inicjalny M_F	0.5			
Inicjalny M_{CR}	0.5			
Mnożnik najlepszych p	0.11			
Mnożnik archiwum a	1.4			
Rozmiar pamięci H	5			

-----TOTLADE

Tabela 7.8: Parametry L-SHADE i rmmLSHADE.

Tylko rmmLSHADE

Czynnik zapominania λ	0.98, 0.99, 1.0
Mnożnik osobników ${\cal N}_s$	5

rmmLSHADE, różniące się szybkością adaptacji parametrów metamodelu. Sprawdzono 3 różnie wartości czynnika zapominania λ , tj, 0.98, 0.99 oraz 1.0. Im mniejsza wartość λ , tym większy stopień zapominania starych obserwacji (nowe obserwacje mają relatywnie większą wagę). Dodatkowo sprawdzono wariant rmmLSHADE, bez adaptacji parametrów, oznaczony jako Ø. W tym wariancie parametry metamodelu \boldsymbol{w}_0 są wyznaczane tylko raz, po utworzeniu inicjalnej populacji P^0 i pozostają niezmienne ($\boldsymbol{w} = \boldsymbol{w}_0$) przez cały czas trwania procesu optymalizacji. Wyniki eksperymentalnej ewaluacji rmmLSHADE zostały zaprezentowane w tabeli 7.5.

Tabela 7.9: Wyniki (na zbiorze testowym CEC2021) rmmLSHADE z $\lambda \in \{0.98, 0.99, 1.0\}$ w zestawieniu z wynikami L-SHADE oraz psLSHADE w budżecie optymalizacji $10^3 \cdot D$. $\lambda = \emptyset$ oznacza wariant bez adaptacji parametrów metamodelu za pomocą filtra RLS.

Algo Score	L-SHADE	psLSHADE	$\lambda = \emptyset$	$\lambda = 0.98$	$\lambda = 0.99$	$\lambda = 1.0$
SNE	30.06	17.86	36.72	15.75	14.81	21.62
SR	217.00	154.25	245.50	138.50	122.75	172.00
Score 1	24.64	41.47	20.16	47.02	50.00	34.25
Score 2	28.28	39.79	25.00	44.31	50.00	35.68
Score	52.92	81.26	45.16	91.33	100.00	69.93

Wariant rmmLSHADE z czynnikiem zapominania $\lambda = 0.99$ okazał się być najlepszym spośród wariantów $\lambda \in \{0.98, 0.99, 1.0\}$. Pokazuje to, że wybór odpowiedniego stopnia adaptacji parametrów metamodelu jest ważny. Za mała wartość czynnika zapominania (w

tym przypadku $\lambda = 0.98$) prowadzi do zbyt szybkiej adaptacji parametrów metamodeli, czego skutkiem jest złe odwzorowanie f.c. Można powiedzieć, że za mała wartość λ prowadzi do przeuczania się metamodelu w oparciu o najnowsze obserwacje. Z kolei, zbyt duży czynnik zapominania (w tym przypadku $\lambda = 1.0$) prowadzi do zbyt wolnej adaptacji parametrów, które nie nadążają za odwzorowaniem zmieniającego się kształtu f.c. w związku z przemieszczaniem się populacji po przestrzeni przeszukiwań. Obserwacja dotycząca zbyt wolnej adaptacji parametrów do zmieniającego się kształtu f.c. znajduje potwierdzenie w przypadku wariantu $\lambda = \emptyset$, który osiągnął najgorszy wynik w zestawieniu. Potwierdza to przypuszczenie, że używanie preselekcji lokalnej w oparciu o źle dopasowany metamodel sprawia, że uzyskane wyniki są jeszcze gorsze niż dla bazowego L-SHADE.

Zestawienie najlepszego wariantu rmmLSHADE ($\lambda = 0.99$) z psLSHADE pokazuje znaczą przewagę na korzyść rmmLSHADE. Różnica względem L-SHADE jest jeszcze większa.

Test statystyczny *Manna–Whitneya* pokazał, że w 57 na 100 przypadków testowych (10 funkcji \times 5 transformacji \times 2 wymiary) różnica na korzyść rmmLSHADE względem psLSHADE była istotna (*p-value*=0.05).

Należy podkreślić, że prawdopodobnie można znaleźć inną wartość λ z przedziału (0.98,1.0), która pozwoliłaby na uzyskanie jeszcze lepszego wyniku, lecz pokazanie jak największej przewagi nad konkurentami nie było celem zaprezentowanej eksperymentalnej ewaluacji.

7.3.4 Wpływ czynnika zapominania na zbieżność algorytmu

Eksperymentalna ewaluacja rmmLSHADE z wykorzystaniem standardowej procedury testowej CEC2021 pokazała, jak duży wpływ na wyniki ma czynnik zapominania λ . Z tego powodu wykonano dodatkowe badanie eksperymentalne. Analogicznie, jak w przypadku psLSHADE (rysunek 7.3), wygenerowano wykresy zbieżności dla funkcji f_1 , f_2 , f_3 oraz f_8 . W zestawieniu uwzględniono bazowy L-SHADE, warianty rmmLSHADE z $\lambda = 0.98, 0.99, 1.0$ oraz rmmLSHADE bez adaptacji (oznaczony wcześniej jako Ø). Wszystkie wykresy zbieżności są przedstawione na rysunku 7.7.

Zaprezentowane porównywanie zbieżności posiada podobną charakterystykę do tego obserwowanego dla psLSHADE. Wykorzystanie preselekcji rozwiązań, z $\lambda = 0.99$ jest użyteczne dla wszystkich funkcji.

W przypadku funkcji f_1 przez większość procesu optymalizacji wszystkie warianty rmmLSHADE zachowują się podobnie. Pokazuje to bardzo dobrą zdolność do odwzorowania f.c. przez inicjalne estymowany metamodel, nawet jak populacja zaczyna zbiegać



ROZDZIAŁ 7. PRESELEKCJA ROZWIĄZAŃ W OPTYMALIZACJI SEMIKOSZTOWNEJ

Rysunek 7.7: Uśredniony błąd rmmLSHADE w wariantach $\lambda = 0.98, 0.99, 1.0$, rmmLSHA-DE bez adaptacji (ozn. jako rmmLSHADE-no-adapt) oraz L-SHADE dla funkcji f_1, f_2, f_3 oraz f_8 (D = 20) w budżecie optymalizacji $10^3 \cdot D$. Na osi x zaznaczono liczbę ewaluacji f.c., a na osi y średni błąd ϵ .

w przestrzeni rozwiązań w sensie hiperobjętości. Co więcej, warianty z największa dynamiką adaptacji ($\lambda \in \{0.98, 0.99\}$) ostatecznie zaczynają wypadać gorzej od pozostałych wariantów rmmLSHADE. Prawdopodobnie jest to spowodowane dużymi błędami numerycznymi, które powstają w sytuacji gdy rozproszenie rozwiązań oraz towarzyszących im wartości f.c. jest niewielkie

Przypadki funkcji f_2 , f_3 oraz f_8 potwierdzają ogólną interpretację wyników otrzymanych za pomocą procedury testowej CEC2021. Adaptacja parametrów dla $\lambda = 0.99$ pozwala osiągać najlepsze wyniki, choć wariant $\lambda = 0.98$ nie wydaje się być znacząco gorszy. Brak adaptacji lub wariant z $\lambda = 1.0$ osiągają najgorsze wyniki i widać, że od pewnego momentu najlepsze znalezione rozwiązanie przestaje się poprawiać. Jest to podyktowane tym, że źle dopasowany metamodel wybiera, wskutek preselekcji, te rozwiązania która są blisko rozwiązania optymalnego \hat{f} , które nie pokrywa się z rozwiązaniem optymalnym f. Podobnie, jak w przypadku psLSHADE oraz LQ-R-SHADE, zysk z użycia metamodelu jest szczególnie widoczny w pierwszej fazie optymalizacji.

7.4 Narzut obliczeniowy preselekcji rozwiązań

Zgodnie z procedurą pomiaru empirycznej złożoności obliczeniowej, znaną z CEC2021 (rozdział 2.4.3), dokonano pomiarów czasów działania L-SHADE, psLSHADE oraz rmmL-SHADE. W zaprezentowanym w tabeli 7.10 zestawieniu T_0 jest czasem działania testowego programu, T_1 jest czasem $2 \cdot 10^5$ ewaluacji funkcji f_2 z CEC2021, T_2 jest średnim czasem działania algorytmu obejmującego $2 \cdot 10^5$ ewaluacji f.c, a $T_3 = (T_2 - T_1)/T_0$ jest finalną empiryczną złożonością obliczeniową.

Dla zwiększenie czytelności zaprezentowano dwie dodatkowe miary: T'_3 oraz n^{mm} . T'_3 jest czasem T_3 w relacji do czasu T_3 uzyskanego przez L-SHADE. n^{mm} oznacza liczbę estymacji parametrów metamodelu podczas kalkulacji czasu T_2 .

Tabela 7.10: Empiryczna złożoność obliczeniowa rmmLSHADE, psLSHADE oraz L-SHADE mierzona zgodnie z procedurą CEC2021 (rozdział 2.4.3). T_0 jest czasem działania testowego programu, T_1 jest czasem $2 \cdot 10^5$ ewaluacji funkcji f_2 z CEC2021. T_2 jest średnim czasem działania algorytmu obejmującego $2 \cdot 10^5$ ewaluacji f.c. $T_3 = (T_2 - T_1)/T_0$ jest finalną empiryczną złożonością obliczeniową. T'_3 jest czasem T_3 w relacji do czasu T_3 uzyskanego przez L-SHADE. n^{mm} oznacza liczbę estymacji parametrów metamodelu podczas kalkulacji czasu T_2 .

D	Algorytm	$T_0[s]$	$T_1[s]$	$T_2[s]$	T_3	T'_3	n^{mm}
10	rmmLSHADE	0.002323	10.4592	83.3448	31364.19	36.2	199821
	psLSHADE	0.002323	10.4592	48.0409	16172.15	18.7	4329
	LSHADE	0.002323	10.4592	12.4722	866.23	1	0
20	rmmLSHADE	0.002323	11.8945	279.9270	115339.96	169.9	199641
	psLSHADE	0.002323	11.8945	67.2213	23808.24	35.1	2528
	LSHADE	0.002323	11.8945	13.4723	678.94	1	0

Badanie wykonano w środowisku *MATLAB R2022a* działającym na systemie Windows 10 z wykorzystaniem sprzętu o następujących parametrach: CPU: Intel Core i7-4700MQ (2.40Ghz), RAM: 16GB.

ROZDZIAŁ 7. PRESELEKCJA ROZWIĄZAŃ W OPTYMALIZACJI SEMIKOSZTOWNEJ

Zaprezentowane wyniki pokazują, że psLSHADE, wykorzystujący preselekcję lokalną i metamodel estymowany za pomocą MNK, jest wolniejszy od bazowego L-SHADE blisko 19 razy dla D = 10 oraz około 35 razy dla D = 20. W przypadku rmmLSHADE, czas działania względem L-SHADE rośnie około 36 razy dla D = 10 oraz około 170 razy D = 20.

Należy zwrócić szczególną uwagę na bezwzględny czasu obliczeń algorytmu, rozumiany w tym przypadku jako $T_2 - T_1$, który odpowiada sumie czasów T_{ap} oraz T_{mm} z równania 5.2. Zgodnie z tym, możliwe jest wyznaczenie narzutu obliczeniowego metamodelu, rozumianego jako T_{mm} . Za T_{ap} przyjęto czas działania $T_2 - T_1$ dla L-SHADE. Finalnie, przyjmując budżet optymalizacji $10^3 \cdot D$ ewaluacji f.c, narzut obliczeniowy, wynikający z użycia metamodelu, w przypadku psLSHADE wynosi około 1.26s dla D = 10 oraz 4.19s dla D = 20. W przypadku rmmLSHADE jest to około 3.02s dla D = 10 oraz 25.46s dla D = 20

Przedstawiając powyższą kalkulację narzutu obliczeniowego (T_{mm}) jako przypadającego na pojedynczą ewaluację f.c., otrzymujemy dla psLSHADE wartości $6.3 \cdot 10^{-6}s$ oraz $2.1 \cdot 10^{-5}s$, odpowiednio dla D = 10 oraz D = 20. Analogicznie, rmmLSHADE uzyskuje $1.5 \cdot 10^{-5}s$ oraz $1.3 \cdot 10^{-4}s$, odpowiednio dla D = 10 oraz D = 20.

Powyższe wartości pokazują, że wykorzystanie preselekcji rozwiązań oraz towarzyszące temu obliczenia związane z metamodelem zwiększają czas obliczeń bazowego algorytmu L-SHADE. Niemniej, kiedy zestawi się bezwzględne wartości uzyskanego narzutu, to wyniki należy uznać za dalece satysfakcjonujące, zwłaszcza jeśli mowa o optymalizacji semikosztownej.

Dłuższy czas obliczeń rmmLSHADE, w porównaniu do psLSHADE wynika przede wszystkim z częstości estymacji parametrów metamodelu. W przypadku psLSHADE ma ona miejsce raz na iterację, a w przypadku rmmLSHADE po każdej ewaluacji (za wyjątkiem inicjalnej estymacji parametrów). Patrząc na liczbę estymacji modelu (n^{mm}) widzimy, że dla D = 10 rmmLSHADE wykonał ponad 46 razy więcej estymacji parametrów niż psLSHADE. Dla D = 20 jest to wartość ponad 79 razy. Stąd, blisko pięciokrotny wzrost czasu działania rmmLSHADE względem psLSHADE dla D = 20 jest w pełni wytłumaczalny i pokazuje jak wydajna jest estymacja parametrów za pomocą filtra RLS.
Rozdział 8

Podsumowanie

W rozprawie został rozważony problem efektywnego zastosowania metamodeli w algorytmach populacyjnych. W tym celu przedyskutowano problem kosztu ewaluacji funkcji celu oraz jego relacji z dostępnym budżetem optymalizacji. Dokonano klasyfikacji budżetów optymalizacji, przypisując je do kategorii optymalizacji kosztownej ($\leq 10^2 \cdot D$ ewaluacji f.c.), semikosztownej ($10^2 \cdot D < ... < 10^4 \cdot D$ ewaluacji f.c.) oraz taniej ($\geq 10^4 \cdot D$ ewaluacji f.c.).

Ponadto, rozprawie omówiono i usystematyzowano wybrane algorytmy populacyjne, które są uznawane za wiodące metody rozwiązywania problemów czarnoskrzynkowej optymalizacji ciągłej. Przedstawiono problematykę modelowania funkcji celu, ze szczególnym uwzględnieniem zastępowania jej ewaluacji wartością metamodelu. Omówiono popularne grupy metamodeli oraz przedyskutowano ich złożoność obliczeniową. Odniesiono się do znanych strategii rozwiązywania problemów optymalizacji kosztownej, a następnie wykazano ich nieefektywność w budżetach optymalizacji semikosztownej oraz taniej. Przeanalizowano znane APWM oraz ich użycie w budżetach optymalizacji większych niż kosztowne.

W rozprawie zdefiniowano pojęcie efektywnego zastosowania metamodelu. Za pierwsze kryterium przyjęto zdolność wynikowego algorytmu do wykorzystanie go w budżetach optymalizacji semikosztownej lub taniej. Drugie kryterium było oparte o akceptowalny narzut obliczeniowy. Trzecie kryterium dotyczyło zadowalających wyników, które są potwierdzone rzetelną analiza eksperymentalną. Ostatnie kryterium dotyczyło uniwersalności algorytmu oraz towarzyszącego jej relatywnie niskiego stopnia parametryzacji. Zaproponowane kryteria posłużyły do dyskusji nad znanymi APWM oraz oceny efektywności zastosowanych w nich metamodeli.

W dysertacji zaproponowano cztery efektywne zastosowania metamodeli: inicjalizację

metamodelem, lokalną optymalizacje metamodelem, preselekcje rozwiązań na podstawie wartości metamodelu oraz rekurencyjną estymację parametrów metamodelu. Wszystkie ww. metody są oparte o regresję wielomianową lub jej rozszerzenie.

Wszystkie sugerowane zastosowania metamodeli zostały zintegrowane ze znanymi oraz skutecznymi algorytmami populacyjnymi, takimi jak GAPSO, R-SHADE oraz L-SHADE. Skutkiem tego, zaprezentowanych oraz poddanych eksperymentalnej ewaluacji zostało pięć autorskich algorytmów: M-GAPSO, SHADE-LM, LQ-R-SHADE, psLSHADE oraz rmmLSHADE.

Inicjalizację metamodelem wykorzystano w M-GAPSO, SHADE-LM oraz LQ-R-SHADE. Lokalną optymalizację metamodelem wykorzystano w M-GAPSO oraz SHADE-LM, dedykowanych rozwiązywaniu problemów optymalizacji taniej. Prese-lekcja rozwiązań znalazła zastosowanie w LQ-R-SHADE, psLSHADE oraz rmmLSHADE, przeznaczonych do rozwiązywania problemów optymalizacji semikosztownej. Przy okazji ewaluacji psLSHADE przeanalizowano szerzej zachowanie metamodelu. Sprawdzono jak często preselekcja rozwiązań wybiera faktycznie najlepszych kandydatów do ewaluacji za pomocą f.c i zestawiono to z miarami jakości metamodelu, które mogą być wyznaczane w czasie rzeczywistym (współczynnik determinacji R^2 oraz korelacja τ -Kendalla).

Estymacja parametrów metamodelu w rmmLSHADE została zrealizowana w sposób rekurencyjny za pomocą filtra RLS. Skutkiem tego rmmLSHADE nie posiada archiwum próbek. W porównaniach wykorzystano znane zbiory testowe COCO oraz CEC2021.

8.1 Dyskusja wyników badań

Zaproponowane zastosowania metamodeli w różnym stopniu modyfikują bazowe algorytmy populacyjne. Zarówno lokalna optymalizacja metamodelem, jak i preselekcja rozwiązań wpływają na działanie algorytmu populacyjnego przez cały czas procesu optymalizacji. Inicjacja metamodelem pełni rolę wspomagającą poprzez skierowanie inicjalnej populacji w obiecujący obszar przestrzeni rozwiązań. Jest to więc wysoce sprawny sposób poprawy wyników w pierwszej fazie optymalizacji. Rekurencyjna estymacja parametrów metamodelu przede wszystkim pozwala na częstsze uczenie metamodelu przy zachowaniu zadowalającego narzutu obliczeniowego. Co więcej, wykorzystanie filtra RLS pozwoliło na eliminację archiwum próbek oraz uproszczenie logiki algorytmu.

Ocena efektywności zaproponowanych rozwiązań wymaga spojrzenia na finalne, poddane eksperymentalnej ewaluacji, APWM. Na wstępie należy przypomnieć, że podczas wszystkich eksperymentalnych ewaluacji zadbano o rzetelną weryfikację wpływu użycia metamodelu na wyniki. Unikano strojenia parametrów pod dany zbiór testowy, ale także zachowano stałą parametryzację w obrębie mechanizmów bazowego algorytmu populacyj-nego.

8.1.1 Dyskusja efektywności lokalnej optymalizacji metamodelem

Lokalna optymalizacja metamodelem znalazła zastosowanie w optymalizacji taniej. Algorytm M-GAPSO, stanowiący rozszerzenie GAPSO, pokazał, że zastosowanie lokalnej optymalizacji metamodelem pozwala na istotną poprawę wyników w ostatniej fazie optymalizacji. Różnica była znacząca dla większych wymiarów, tj. $D \in \{10, 20, 40\}$. Wzrost skuteczności M-GAPSO względem GAPSO był widoczny we wszystkich wymiarach funkcji separowanych. Eksperymentalna ewaluacja pokazała też, że nowatorski sposób konstrukcji zbioru uczącego w metamodelu wielomianowym pozwala na osiągnięcie lepszych wyników niż za pomocą znanych metod.

Wyjaśnienia wymaga posiadanie przez M-GAPSO relatywnie dużej liczby parametrów. Wynika to z faktu, że jest to algorytm hybrydowy i każdy z wykorzystywanych algorytmów posiada własną parametryzację. W przypadku samych metamodeli, wykorzystywany jest jeden parametr na dany metamodel, który oznacza wielkość zbioru uczącego.

Podobnie jak w przypadku M-GAPSO, wykorzystanie lokalnej optymalizacji metamodelem w R-SHADE pozwoliło na poprawę wyników dla wszystkich wymiarów problemów testowych. Jednakże wyraźna przewaga była widoczna w wymiarach D = 10 oraz D = 20. Warto zauważyć, że bazowy R-SHADE jest skutecznym algorytmem z rodziny DE wykorzystującym mechanizm adaptacji parametrów, skutkiem czego poprawa wyników była wyzwaniem.

Wzrost czasu obliczeń po zastosowaniu lokalnej optymalizacji metamodelem w przypadku M-GAPSO wyniósł między 4 a 6 razy, w zależności od wielkości problemu. Najbardziej złożony przypadek (D = 100) skutkuje czasem $2.4 \cdot 10^{-4}s$ przypadającym na pojedynczą ewaluację f.c. Najmniej złożony przypadek (D = 10) uzyskał odpowiednio czas $4.1 \cdot 10^{-5}s$. W przypadku SHADE-LM badane narzutu obliczeniowego potwierdziło wyniki pierwowzoru, jakim jest M-GAPSO. Najbardziej złożony problem w zestawieniu (D = 20) skutkował całkowitym czasem obliczeń przypadającym na jedną ewaluację f.c. wynoszącym $5.2 \cdot 10^{-5}s$. Powyższe wyniki wydajności należy uznać za dalece satysfakcjonujące. Zestawiając wszystkie aspekty analizy oraz eksperymentalnej ewaluacji M-GAPSO oraz SHADE-LM, należy uznać, że wykorzystanie lokalnej optymalizacji metamodelem jest efektywne.

8.1.2 Dyskusja efektywności preselekcji rozwiązań

Preselekcja rozwiązań znalazła zastosowanie w optymalizacji semikosztownej. Zaprezentowane algorytmy oparte o preselekcję rozwiązań na podstawie wartości metamodelu, tj. LQ-R-SHADE, psLSHADE oraz rmmLSHADE, stanowią pewien ciąg prac badawczych związanych z poszukiwaniem efektywnych zastosowań metamodeli.

Algorytm LQ-R-SHADE pokazał, że zastosowanie kaskady metamodeli, znanej z algorytmu lq-CMA-ES może być wykorzystane w algorytmie R-SHADE. Konstrukcja algorytmów z rodziny DE jest istotnie inna niż algorytmów z rodziny CMA-ES, więc sposób integracji metamodelu też jest znacząco odmienny. Eksperymentalna ewaluacja pokazała, że preselekcja rozwiązań w wydaniu lokalnym pozwoliła na poprawę wyników w środkowym budżecie optymalizacji semikosztownej, tj. $10^3 \cdot D$ ewaluacji f.c. Niemniej zaobserwowano, że zysk z wykorzystania metamodelu spada w końcowej fazie optymalizacji.

W kolejnym algorytmie, tj. psLSHADE zrezygnowano z kaskady metamodeli, celem poszukiwania rozwiązań bardziej uniwersalnych i prostych w swojej strukturze. Jednocześnie rozszerzono metamodel o nieliniowe transformacje zmiennej objaśniającej. Tym razem za bazowy algorytm przyjęto L-SHADE obserwując, że mechanizm restartów można wyeliminować za pomocą większego rozmiaru populacji. Wyniki ponownie okazały się satysfakcjonujące. psLSHADE okazał się być lepszy nie tylko od L-SHADE, ale także od MadDE, który wykazuje się dużą skutecznością w budżetach optymalizacji taniej.

Ostatni z algorytmów, tj. rmmLSHADE pokazał, że zastosowanie filtra RLS do estymacji parametrów regresji wielomianowej pozwala na uproszczenie logiki algorytmu, poprzez eliminację archiwum. Ponadto zastąpiono preselekcje lokalna za pomocą preselekcji globalnej. Porównano się z bazowym L-SHADE oraz uprzednio zaprezentowanymi psLSHADE, uzyskując wyniki jednoznacznie lepsze od obu konkurentów.

We wszystkich zaprezentowanych algorytmach preselekcja rozwiązań nie wpływa na mechanizm adaptacji parametrów. Na tej samej zasadzie jest ona transparentna dla mechanizmu restartów oraz mechanizm redukcji rozmiaru populacji.

Badanie narzutu obliczeniowego preselekcji rozwiązań pokazało, że stanowi ona efektywną formę zastosowania metamodelu. Algorytm psLSHADE, dla najbardziej złożonego przypadku w zestawieniu (D = 20), był w przybliżeniu 35 razy wolniejszy od bazowego L- SHADE. Algorytm rmmLSHADE był odpowiednio blisko 170 razy wolniejszy. Niemniej, zakładając średni budżet optymalizacji semikosztownej ($10^d \cdot D$ ewaluacji f.c.) całkowity narzut obliczeniowy wynikający z używania metamodelu wyniósł 4.19*s* dla psLSHADE oraz 25.46*s* dla rmmLSHADE. Należy przypomnieć, że rmmLSHADE estymuje parametry metamodelu co ewaluację, a nie co iterację jak psLSHADE. W skutego tego dla D = 20 wykonał ponad 79 razy więcej estymacji parametrów, a był wolniejszy w przybliżeniu mniej niż 5 razy.

Biorąc pod uwagę że optymalizacja semikosztowna zakłada zauważalny koszt pojedynczej ewaluacji f.c., to uzyskane wyniki czasu obliczeń są bardzo dobre oraz bez porównania lepsze względem klasycznych metod opartych o optymalizację bayesowską.

Podsumowując, wszystkie trzy algorytmy należy uznać za efektywnie integrujące metamodel, ponieważ spełnione zostały wszystkie wymagane kryteria. Szczególną uwagę należy zwrócić na rmmLSHADE, który estymuje parametry metamodelu w sposób istotnie różny niż jest to na ogół praktykowane w APWM.

8.2 Weryfikacja hipotez badawczych

W rozdziale 1.3 przedstawiono cztery hipotezy badawcze, które miały być poddane weryfikacji wskutek prac badawczych. Treść rozprawy oraz zaprezentowana wyżej dyskusja pozwala stwierdzić, że zostały one pozytywnie potwierdzone, lecz dla czytelności poniżej przedstawiono stosowne komentarze do każdej z hipotez.

Hipotezę H1 mówiącą, że jest możliwe zastosowanie w algorytmie populacyjnym metamodeli służących bezpośredniemu wyznaczaniu rozwiązań poddawanych ewaluacji z wykorzystaniem funkcji celu, co poprawi wyniki algorytmu w optymalizacji taniej, uznaje się za potwierdzoną, czego dowodem jest efektywny mechanizm lokalnej optymalizacji metamodelem zastosowany w algorytmie M-GAPSO oraz SHADE-LM.

Hipotezę H2 mówiącą że jest możliwe zastosowanie globalnego metamodelu służącego do preselekcji obiecujących rozwiązań w algorytmie populacyjnym opartym o adaptacyjną ewolucję różnicową w sposób umożliwiający osiąganie lepszych wyników w optymalizacji semikosztownej, uznaje się za potwierdzoną, czego dowodem są algorytmy LQ-R-SHADE, psLSHADE oraz rmmLSHADE, oparte o preselekcję rozwiązań z wykorzystaniem metamodelu globalnego.

Hipotezę H3 mówiącą, że jest możliwe zaprojektowanie mechanizmu prese-

lekcji obiecujących rozwiązań w sposób niewymagający modyfikacji mechanizmów adaptacji parametrów., uznaje się za potwierdzoną, czego dowodem ponownie są LQ-R-SHADE, psLSHADE oraz rmmLSHADE, które są oparte o algorytm SHADE, zakładający adaptację parametrów: czynnika skalującego oraz prawdopodobieństwa krzyżowania. Zastosowanie preselekcji rozwiązań, w wariancie lokalnym i globalnym, nie wypłynęło na konstrukcję mechanizmu adaptacji parametrów.

Hipotezę H4 mówiącą, że jest możliwe zintegrowanie metamodelu z algorytmem populacyjnym, w taki sposób, że będzie on estymowany rekurencyjnie, a tym samym użycie archiwum będzie niepotrzebne, uznaje się za potwierdzoną, czego dowodem jest algorytm rmmLSHADE, w którym estymacja parametrów regresji wielomianowej odbywa się za pomocą filtra RLS.

8.3 Dalsze kierunki badań

Przedstawione w rozprawie wyniki otwierają drogę do dalszych badań nad zastosowaniem metamodeli w optymalizacji semikosztownej oraz taniej.

W przypadku optymalizacji semikosztownej zasadnym wydaje się poszukiwanie możliwość rozszerzania mechanizmu preselekcji rozwiązań o adaptację jej parametrów. Adaptacja może być dokonywana w oparciu o bieżącą skuteczność metamodelu i jeśli jest ona duża mnożnik osobników mógłby być zwiększany. Analogicznie, gdy jakość preselekcji spada, należy ograniczyć jej wykorzystanie, celem unikania kierowania populacji do nieobiecujących obszarów.

Wyzwaniem dla stosowania takiego rodzaju adaptacji jest konieczność przyjęcia pewnych założeń i parametrów, co może stać w sprzeczności z uniwersalnością i czytelnością projektowanych rozwiązań. Niemniej, wpływ takiego rozwiązania na wyniki proponowanych APWM jest warty zbadania.

W przypadku optymalizacji taniej możliwe jest stosowanie lokalnej optymalizacji metamodelem w miejscach przestrzeni rozwiązań, w których metamodel jest dobrze dopasowany, a nie jak ma to miejsce teraz - określonych z góry. Takie rozwiązanie pozwoliłoby na poprawę wyników poprzez zdolność algorytmu do lepszej eksploatacji oraz ograniczyłoby wzrost narzutu obliczeniowego.

W obrębie wszystkich budżetów optymalizacji zasadne jest sprawdzenie skuteczności rozszerzenia regresji wielomianowej o większa liczbę transformacji zmiennej objaśniającej. Ponadto, warto zbadać możliwość użycia mechanizmu automatycznej selekcji używanych transformacji celem wyboru tylko tych najlepiej przybliżających f.c.

8.4 Autorski wkład w dziedzinę

Autor rozprawy deklaruje, że jego autorski wkład w dziedzinę stanowi:

- Koncepcja, opracowanie, implementacja oraz badania eksperymentalne prowadzone nad algorytmem MRFO [246].
- Koncepcja, opracowanie oraz implementacja metamodelu wielomianowego (metody lokalnej optymalizacji metamodelem) w algorytmie GAPSO¹, skutkująca powstaniem M-GAPSO [165] (rozdział 6.2).
- Analiza działania metamodeli (kwadratowego i wielomianowego) w M-GAPSO oraz towarzyszące jej badania eksperymentalne, zawarte w pracach [244, 245].
- Wsparcie w implementacji lokalnej optymalizacji metamodelem w algorytmie R-SHADE, prowadzące do powstania algorytmu SHADE-LM [164] (rozdział 6.3).
- Koncepcja, opracowanie, implementacja oraz badania eksperymentalne prowadzone nad algorytmami LQ-R-SHADE [242], psLSHADE [241] oraz rmmLSHADE [243] (rozdział 7).

Ponadto autor rozprawy dokonał:

- Podziału optymalizacji ze względu na jej budżety (rysunek 2.5).
- Klasyfikacji wybranych metod rozwiązujących problemy optymalizacji ciągłej z jednym celem (rysunek 3.1).
- Gradacji sposobów pozyskania informacji o wartości funkcji celu w danym punkcie (rysunek 4.1).
- Zdefiniowania efektywnego zastosowania metamodelu (rozdział 5.1).
- Eksperymentalnego badania czasu estymacji parametrów metamodeli (rozdział 5.1).

¹Algorytm GAPSO został zaimplementowany przez doktorantów Wydziału MiNI Politechniki Warszawskiej (Adama Żychowskiego oraz Mateusza Ulińskiego) według projektu dr. Michała Okulewicza.

Bibliografia

- [1] IEEE Congress on Evolutionary Computation, CEC 2019, Wellington, New Zealand, June 10-13, 2019. IEEE, 2019.
- [2] IEEE Congress on Evolutionary Computation, CEC 2020, Glasgow, United Kingdom, July 19-24, 2020. IEEE, 2020.
- [3] IEEE Congress on Evolutionary Computation, CEC 2021, Kraków, Poland, June 28 - July 1, 2021. IEEE, 2021.
- [4] Kamal Abboud and Marc Schoenauer. Surrogate deterministic mutation: Preliminary results. In International Conference on Artificial Evolution (Evolution Artificielle), pages 104–116. Springer, 2001.
- [5] Hisham ME Abdelsalam and Han P Bao. A simulation-based optimization framework for product development cycle time reduction. *IEEE Transactions on Engineering Management*, 53(1):69–85, 2006.
- [6] Michael Affenzeller, Stefan Wagner, and Stephan Winkler. Evolutionary systems identification: New algorithmic concepts and applications. INTECH Open Access Publisher, 2008.
- [7] Ouassim Ait Elhara, Anne Auger, and Nikolaus Hansen. A median success rule for non-elitist evolution strategies: Study of feasibility. In *Proceedings of the 15th* annual conference on Genetic and evolutionary computation, pages 415–422, 2013.
- [8] Sinem Akyol and Bilal Alatas. Plant intelligence based metaheuristic optimization algorithms. Artificial Intelligence Review, 47(4):417–462, 2017.
- [9] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician, 46(3):175–185, 1992.

- [10] Kurt S Anderson and YuHong Hsu. Genetic crossover strategy using an approximation concept. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), volume 1, pages 527–533. IEEE, 1999.
- [11] Asma Atamna. Benchmarking ipop-cma-es-tpa and ipop-cma-es-msr on the bbob noiseless testbed. In Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, pages 1135–1142, 2015.
- [12] Charles Audet and Warren Hare. Derivative-free and blackbox optimization, volume 2. Springer, 2017.
- [13] Anne Auger and Nikolaus Hansen. A restart cma evolution strategy with increasing population size. In 2005 IEEE congress on evolutionary computation, volume 2, pages 1769–1776. IEEE, 2005.
- [14] Anne Auger, Marc Schoenauer, and Nicolas Vanhaecke. Ls-cma-es: A second-order algorithm for covariance matrix adaptation. In *International Conference on Parallel Problem Solving from Nature*, pages 182–191. Springer, 2004.
- [15] N.H. Awad, M.Z. Ali, P.N. Suganthan, Liang J.J., and Qu B.Y. Problem definitions and evaluation criteria for the CEC 2017 special session competition on constrained real-parameter optimization. https://github.com/P-N-Suganthan/ CEC2017-BoundContrained/.
- [16] Noor H Awad, Mostafa Z Ali, Rammohan Mallipeddi, and Ponnuthurai N Suganthan. An improved differential evolution algorithm using efficient adapted surrogate model for numerical optimization. *Information Sciences*, 451:326–347, 2018.
- [17] Noor H Awad, Mostafa Z Ali, and Ponnuthurai N Suganthan. Ensemble sinusoidal differential covariance matrix adaptation with euclidean neighborhood for solving cec2017 benchmark problems. In 2017 IEEE Congress on Evolutionary Computation (CEC), pages 372–379. IEEE, 2017.
- [18] Lukáš Bajer, Zbyněk Pitra, and Martin Holeňa. Benchmarking gaussian processes and random forests surrogate models on the bbob noiseless testbed. In Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, pages 1143–1150, 2015.

- [19] Lukáš Bajer, Zbyněk Pitra, Jakub Repický, and Martin Holeňa. Gaussian process surrogate models for the cma evolution strategy. *Evolutionary computation*, 27(4):665–697, 2019.
- [20] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In Proceedings of the 1990 ACM SIGMOD international conference on Management of data, pages 322–331, 1990.
- [21] Maumita Bhattacharya. Evolutionary approaches to expensive optimisation. arXiv preprint arXiv:1303.2745, 2013.
- [22] Atharv Bhosekar and Marianthi Ierapetritou. Advances in surrogate based modeling, feasibility analysis, and optimization: A review. Computers & Chemical Engineering, 108:250–267, 2018.
- [23] Rafał Biedrzycki. A version of ipop-cma-es algorithm with midpoint for cec 2017 single objective bound constrained problems. In 2017 IEEE Congress on Evolutionary Computation (CEC), pages 1489–1494. IEEE, 2017.
- [24] Christopher M Bishop and Nasser M Nasrabadi. Pattern recognition and machine learning, volume 4. Springer, 2006.
- [25] J Mark Bishop. Stochastic searching networks. In 1989 First IEE international conference on artificial neural networks, (Conf. Publ. No. 313), pages 329–331. IET, 1989.
- [26] Subhodip Biswas, Debanjan Saha, Shuvodeep De, Adam D Cobb, Swagatam Das, and Brian A Jalaian. Improving differential evolution through bayesian hyperparameter optimization. In 2021 IEEE Congress on Evolutionary Computation (CEC), pages 832–840. IEEE, 2021.
- [27] George EP Box and Kenneth B Wilson. On the experimental attainment of optimum conditions. In *Breakthroughs in statistics*, pages 270–310. Springer, 1992.
- [28] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. Convex optimization. Cambridge university press, 2004.
- [29] Jürgen Branke and Christian Schmidt. Faster convergence by means of fitness estimation. Soft Computing, 9(1):13–20, 2005.

- [30] Janez Brest, Sao Greiner, Borko Boskovic, Marjan Mernik, and Viljem Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE transactions on evolutionary computation*, 10(6):646–657, 2006.
- [31] Janez Brest, Mirjam Sepesy Maučec, and Borko Bošković. Differential evolution algorithm for single objective bound-constrained optimization: Algorithm j2020. In 2020 IEEE Congress on Evolutionary Computation (CEC), pages 1–8. IEEE, 2020.
- [32] Larry Bull. On model-based evolutionary computation. Soft Computing, 3(2):76–82, 1999.
- [33] Xiwen Cai, Liang Gao, Xinyu Li, and Haobo Qiu. Surrogate-guided differential evolution algorithm for high dimensional expensive problems. *Swarm and Evolutionary Computation*, 48:288–311, 2019.
- [34] Amrita Chakraborty and Arpan Kumar Kar. Swarm intelligence: A review of algorithms. *Nature-inspired computing and optimization*, pages 475–494, 2017.
- [35] Rachid Chelouah and Patrick Siarry. A continuous genetic algorithm designed for the global optimization of multimodal functions. *Journal of Heuristics*, 6(2):191– 213, 2000.
- [36] Qin Chen, B Liu, Qinfu Zhang, Jing Liang, P Suganthan, and Boyang Qu. Problem definitions and evaluation criteria for CEC 2015 special session on bound constrained single-objective computationally expensive numerical optimization. https://github.com/P-N-Suganthan/CEC2015/.
- [37] E Ward Cheney and David R Kincaid. Numerical mathematics and computing. Cengage Learning, 2012.
- [38] Tinkle Chugh, Yaochu Jin, Kaisa Miettinen, Jussi Hakanen, and Karthik Sindhya. A surrogate-assisted reference vector guided evolutionary algorithm for computationally expensive many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 22(1):129–142, 2016.
- [39] Tinkle Chugh, Karthik Sindhya, Jussi Hakanen, and Kaisa Miettinen. A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms. *Soft Computing*, 23(9):3137–3166, 2019.

- [40] Maurice Clerc. Standard particle swarm optimisation, 2012.
- [41] Andrew R Conn, Katya Scheinberg, and Luis N Vicente. Introduction to derivativefree optimization. SIAM, 2009.
- [42] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. ACM computing surveys (CSUR), 45(3):1–33, 2013.
- [43] Jean-Jerôme Da Costa, Fabien Chainet, Benoît Celse, Marion Lacoue-Nègre, Cyril Ruckebusch, Noémie Caillol, and Didier Espinat. Kriging modeling to predict viscosity index of base oils. *Energy & fuels*, 32(2):2588–2597, 2018.
- [44] Charles Darwin. On the origin of species, 1859. Routledge, 2004.
- [45] Ashraf Darwish, Aboul Ella Hassanien, and Swagatam Das. A survey of swarm and evolutionary computing approaches for deep learning. *Artificial Intelligence Review*, 53(3):1767–1812, 2020.
- [46] Swagatam Das, Ajith Abraham, and Amit Konar. Particle swarm optimization and differential evolution algorithms: technical analysis, applications and hybridization perspectives. In Advances of computational intelligence in industrial systems, pages 1–38. Springer, 2008.
- [47] Swagatam Das, Sankha Subhra Mullick, and Ponnuthurai N Suganthan. Recent advances in differential evolution-an updated survey. Swarm and evolutionary computation, 27:1–30, 2016.
- [48] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31, 2010.
- [49] Siva Krishna Dasari, Abbas Cheddad, and Petter Andersson. Random forest surrogate models to support design space exploration in aerospace use-case. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 532–544. Springer, 2019.
- [50] Javier Del Ser, Eneko Osaba, Daniel Molina, Xin-She Yang, Sancho Salcedo-Sanz, David Camacho, Swagatam Das, Ponnuthurai N Suganthan, Carlos A Coello Coello,

and Francisco Herrera. Bio-inspired computation: Where we stand and what's next. Swarm and Evolutionary Computation, 48:220–250, 2019.

- [51] Alan Díaz-Manríquez, Gregorio Toscano-Pulido, and Wilfrido Gómez-Flores. On the selection of surrogate models in evolutionary optimization algorithms. In 2011 IEEE congress of evolutionary computation (CEC), pages 2155–2162. IEEE, 2011.
- [52] Jinliang Ding, Cuie Yang, Yaochu Jin, and Tianyou Chai. Generalized multitasking for evolutionary optimization of expensive problems. *IEEE Transactions on Evolutionary Computation*, 23(1):44–58, 2017.
- [53] Ding-Zhu Du and Ker-I Ko. Theory of computational complexity, volume 58. John Wiley & Sons, 2011.
- [54] David Eby, RC Averill, William F Punch, and Erik D Goodman. Evaluation of injection island ga performance on flywheel design optimisation. In Adaptive Computing in Design and Manufacture, pages 121–136. Springer, 1998.
- [55] Saber M Elsayed, Tapabrata Ray, and Ruhul A Sarker. A surrogate-assisted differential evolution algorithm with dynamic parameters selection for solving expensive optimization problems. In 2014 IEEE congress on evolutionary computation (CEC), pages 1062–1068. IEEE, 2014.
- [56] Michael Emmerich, Alexios Giotis, Mutlu Ozdemir, Thomas Bäck, and Kyriakos Giannakoglou. Metamodel—assisted evolution strategies. In International Conference on parallel problem solving from nature, pages 361–370. Springer, 2002.
- [57] Michael TM Emmerich, Kyriakos C Giannakoglou, and Boris Naujoks. Single-and multiobjective evolutionary optimization assisted by gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation*, 10(4):421–439, 2006.
- [58] Maria Ferrara, Enrico Fabrizio, Joseph Virgone, and Marco Filippi. A simulationbased optimization method for cost-optimal analysis of nearly zero energy buildings. *Energy and Buildings*, 84:442–457, 2014.
- [59] Steffen Finck, Nikolaus Hansen, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical report, Citeseer, 2010.

- [60] J Michael Fitzpatrick and John J Grefenstette. Genetic algorithms in noisy environments. *Machine learning*, 3(2):101–120, 1988.
- [61] Peter J Fleming and Robin C Purshouse. Evolutionary algorithms in control systems engineering: a survey. Control engineering practice, 10(11):1223–1241, 2002.
- [62] Christodoulos A Floudas and Chrysanthos E Gounaris. A review of recent advances in global optimization. *Journal of Global Optimization*, 45(1):3–38, 2009.
- [63] Alexander IJ Forrester and Andy J Keane. Recent advances in surrogate-based optimization. Progress in aerospace sciences, 45(1-3):50–79, 2009.
- [64] Peter I Frazier. Bayesian optimization. In Recent advances in optimization and modeling of contemporary problems, pages 255–278. Informs, 2018.
- [65] Jerome H Friedman. Multivariate adaptive regression splines. The annals of statistics, 19(1):1–67, 1991.
- [66] C Fu, C Fu, and Michael Michael. Handbook of simulation optimization. Springer, 2015.
- [67] Amir Hossein Gandomi and Amir Hossein Alavi. Krill herd: a new bio-inspired optimization algorithm. Communications in nonlinear science and numerical simulation, 17(12):4831–4845, 2012.
- [68] José García-Nieto, Enrique Alba, and Javier Apolloni. Noiseless functions black-box optimization: evaluation of a hybrid particle swarm with differential operators. In Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, pages 2231–2238, 2009.
- [69] Tushar Goel, Raphael T Haftka, Wei Shyy, and Nestor V Queipo. Ensemble of surrogates. Structural and Multidisciplinary Optimization, 33(3):199–216, 2007.
- [70] David E Goldberg and Manohar P Samtani. Engineering optimization via genetic algorithm. In *Electronic computation*, pages 471–482. ASCE, 1986.
- [71] Wenyin Gong, Aimin Zhou, and Zhihua Cai. A multioperator search strategy based on cheap surrogate models for evolutionary optimization. *IEEE transactions on Evolutionary Computation*, 19(5):746–758, 2015.

- [72] Yue-Jiao Gong, Wei-Neng Chen, Zhi-Hui Zhan, Jun Zhang, Yun Li, Qingfu Zhang, and Jing-Jing Li. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing*, 34:286–300, 2015.
- [73] Abhijit Gosavi et al. Simulation-based optimization. Springer, 2015.
- [74] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36:114–144, 2021.
- [75] Nikolaus Hansen. Benchmarking a bi-population cma-es on the bbob-2009 function testbed. In Proceedings of the 11th annual conference companion on genetic and evolutionary computation conference: late breaking papers, pages 2389–2396, 2009.
- [76] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.
- [77] Nikolaus Hansen. A global surrogate assisted cma-es. In Proceedings of the Genetic and Evolutionary Computation Conference, pages 664–672, 2019.
- [78] Nikolaus Hansen, Asma Atamna, and Anne Auger. How to assess step-size adaptation mechanisms in randomised search. In *International Conference on Parallel Problem Solving from Nature*, pages 60–69. Springer, 2014.
- [79] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). Evolutionary computation, 11(1):1–18, 2003.
- [80] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized selfadaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.
- [81] Ivana Hartmann Tolić, Goran Martinović, and Dominika Crnjac Milić. Optimization methods in modern transportation systems. *Tehnički vjesnik*, 25(2):627–634, 2018.
- [82] Trevor J Hastie and Robert J Tibshirani. Generalized additive models. Routledge, 2017.
- [83] Mark Hauschild and Martin Pelikan. An introduction and survey of estimation of distribution algorithms. Swarm and evolutionary computation, 1(3):111–128, 2011.

- [84] Jon C Helton and Freddie Joe Davis. Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems. *Reliability Engineering & System* Safety, 81(1):23-69, 2003.
- [85] Michael Hinze, René Pinnau, Michael Ulbrich, and Stefan Ulbrich. Optimization with PDE constraints, volume 23. Springer Science & Business Media, 2008.
- [86] Martin Holeňa, David Linke, Uwe Rodemerck, and Lukáš Bajer. Neural networks as surrogate models for measurements in optimization algorithms. In International Conference on Analytical and Stochastic Modeling Techniques and Applications, pages 351–366. Springer, 2010.
- [87] John H Holland. Adaptation in natural and artificial systems. an introductory analysis with applications to biology, control and artificial intelligence. Ann Arbor: University of Michigan Press, 1975.
- [88] Abdollah Homaifar, Charlene X Qi, and Steven H Lai. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–253, 1994.
- [89] Reiner Horst, Panos M Pardalos, and Nguyen Van Thoai. Introduction to global optimization. Springer Science & Business Media, 2000.
- [90] Gao Huang, Guang-Bin Huang, Shiji Song, and Keyou You. Trends in extreme learning machines: A review. *Neural Networks*, 61:32–48, 2015.
- [91] Guang-Bin Huang, Dian Hui Wang, and Yuan Lan. Extreme learning machines: a survey. International journal of machine learning and cybernetics, 2(2):107–122, 2011.
- [92] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An evaluation of sequential model-based optimization for expensive blackbox functions. In *Proceedings of the* 15th annual conference companion on Genetic and evolutionary computation, pages 1209–1216, 2013.
- [93] James P Ignizio and Tom M Cavalier. *Linear programming*. Prentice-Hall, Inc., 1994.
- [94] Alexandros Iosifidis, Anastastios Tefas, and Ioannis Pitas. On the kernel extreme learning machine classifier. *Pattern Recognition Letters*, 54:11–17, 2015.

- [95] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.
- [96] M Janga Reddy and D Nagesh Kumar. Evolutionary algorithms, swarm intelligence methods, and their applications in water resources engineering: a state-of-the-art review. *H2Open Journal*, 3(1):135–188, 2020.
- [97] Thomas Jansen and Christine Zarges. Analysis of evolutionary algorithms: From computational complexity analysis to algorithm engineering. In Proceedings of the 11th workshop proceedings on Foundations of genetic algorithms, pages 1–14, 2011.
- [98] Tim Janus, Anne Lüubbers, and Sebastian Engell. Neural networks for surrogateassisted evolutionary optimization of chemical processes. In 2020 IEEE Congress on Evolutionary Computation (CEC), pages 1–8. IEEE, 2020.
- [99] Jin Jin, Chuan Yang, and Yi Zhang. An improved cma-es for solving large scale optimization problem. In *International Conference on Swarm Intelligence*, pages 386–396. Springer, 2020.
- [100] Yaochu Jin. A comprehensive survey of fitness approximation in evolutionary computation. Soft computing, 9(1):3–12, 2005.
- [101] Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. Swarm and Evolutionary Computation, 1(2):61–70, 2011.
- [102] Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on evolutionary computation*, 6(5):481–494, 2002.
- [103] Yaochu Jin, Handing Wang, Tinkle Chugh, Dan Guo, and Kaisa Miettinen. Datadriven evolutionary optimization: An overview and case studies. *IEEE Transactions* on Evolutionary Computation, 23(3):442–458, 2018.
- [104] 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.
- [105] Hubertus Th Jongen, Klaus Meer, and Eberhard Triesch. Optimization theory. Springer Science & Business Media, 2007.

- [106] June Young Jung, Gary Blau, Joseph F Pekny, Gintaras V Reklaitis, and David Eversdyk. A simulation based optimization approach to supply chain management under demand uncertainty. *Computers & chemical engineering*, 28(10):2087–2106, 2004.
- [107] Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39(3):459–471, 2007.
- [108] Takeaki Kariya and Hiroshi Kurata. Generalized least squares. John Wiley & Sons, 2004.
- [109] Anezka Kazikova, Michal Pluhacek, Roman Senkerik, and Adam Viktorin. Proposal of a new swarm optimization method inspired in bison behavior. In 23rd International Conference on Soft Computing, pages 146–156. Springer, 2017.
- [110] James Kennedy and Russell Eberhart. Particle swarm optimization. In Proceedings of ICNN'95-international conference on neural networks, volume 4, pages 1942– 1948. IEEE, 1995.
- [111] Stefan Kern, Nikolaus Hansen, and Petros Koumoutsakos. Fast quadratic local meta-models for evolutionary optimization of anguilliform swimmers. In EUROGEN 2007, 2004.
- [112] Stefan Kern, Nikolaus Hansen, and Petros Koumoutsakos. Local meta-models for optimization using evolution strategies. In *Parallel Problem Solving from Nature-PPSN IX*, pages 939–948. Springer, 2006.
- [113] Pascal Kerschke and Heike Trautmann. Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evolutionary computation*, 27(1):99–127, 2019.
- [114] Thanh Tung Khuat and My Hanh Le. A novel hybrid abc-pso algorithm for effort estimation of software projects using agile methodologies. *Journal of Intelligent* Systems, 27(3):489–506, 2018.
- [115] Joshua Knowles and Evan J Hughes. Multiobjective optimization on a budget of 250 evaluations. In International Conference on Evolutionary Multi-criterion Optimization, pages 176–190. Springer, 2005.

- [116] Oliver Kramer, David Echeverría Ciaurri, and Slawomir Koziel. Derivative-free optimization. In *Computational optimization, methods and algorithms*, pages 61–83. Springer, 2011.
- [117] Mohanarangam Krithikaa and Rammohan Mallipeddi. Differential evolution with an ensemble of low-quality surrogates for expensive optimization problems. In 2016 IEEE Congress on Evolutionary Computation (CEC), pages 78–85. IEEE, 2016.
- [118] Abhishek Kumar, Kenneth V Price, Ali Wagd Mohamed, Anas A Hadi, and P.N Suganthan. Problem definitions and evaluation criteria for the CEC 2022 special session and competition on single objective bound constrained numerical optimization. https://github.com/P-N-Suganthan/2022-SO-BCO/.
- [119] Kenneth Lange. Optimization, volume 95. Springer Science & Business Media, 2013.
- [120] Antonio LaTorre, Daniel Molina, Eneko Osaba, Javier Poyatos, Javier Del Ser, and Francisco Herrera. A prescription of methodological guidelines for comparing bioinspired optimization algorithms. Swarm and Evolutionary Computation, 67:100973, 2021.
- [121] Fan Li, Xiwen Cai, and Liang Gao. Ensemble of surrogates assisted particle swarm optimization of medium scale expensive problems. *Applied Soft Computing*, 74:291– 305, 2019.
- [122] Fan Li, Xiwen Cai, Liang Gao, and Weiming Shen. A surrogate-assisted multiswarm optimization algorithm for high-dimensional computationally expensive problems. *IEEE transactions on cybernetics*, 51(3):1390–1402, 2020.
- [123] Fan Li, Yingli Li, Xiwen Cai, and Liang Gao. A surrogate-assisted hybrid swarm optimization algorithm for high-dimensional computationally expensive problems. *Swarm and Evolutionary Computation*, page 101096, 2022.
- [124] Fan Li, Weiming Shen, Xiwen Cai, Liang Gao, and G Gary Wang. A fast surrogateassisted particle swarm optimization algorithm for computationally expensive problems. *Applied Soft Computing*, 92:106303, 2020.
- [125] Jiahang Li, Yuelin Gao, Kaiguang Wang, and Ying Sun. A dual opposition-based learning for differential evolution with protective mechanism for engineering optimization problems. *Applied Soft Computing*, 113:107942, 2021.

- [126] Jian-Yu Li, Zhi-Hui Zhan, and Jun Zhang. Evolutionary computation for expensive optimization: A survey. *Machine Intelligence Research*, 19(1):3–23, 2022.
- [127] Zhi Li, Shu-Chuan Chu, Jeng-Shyang Pan, Pei Hu, and Xingsi Xue. A mahalanobis surrogate-assisted ant lion optimization and its application in 3d coverage of wireless sensor networks. *Entropy*, 24(5):586, 2022.
- [128] Jing J Liang, Bo Y Qu, and Ponnuthurai N Suganthan. Problem definitions and evaluation criteria for the CEC 2014 special session competition on constrained real-parameter optimization. https://github.com/P-N-Suganthan/CEC2014/.
- [129] Dudy Lim, Yew-Soon Ong, Yaochu Jin, and Bernhard Sendhoff. Trusted evolutionary algorithm. In 2006 IEEE International Conference on Evolutionary Computation, pages 149–156. IEEE, 2006.
- [130] Ming-Hua Lin, Jung-Fa Tsai, and Chian-Son Yu. A review of deterministic optimization methods in engineering and management. *Mathematical Problems in Engineering*, 2012, 2012.
- [131] Bo Liu, Qingfu Zhang, and Georges GE Gielen. A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems. *IEEE Transactions on Evolutionary Computation*, 18(2):180–192, 2013.
- [132] Jiao Liu, Yong Wang, Guangyong Sun, and Tong Pang. Multisurrogate-assisted ant colony optimization for expensive optimization problems with continuous and categorical variables. *IEEE Transactions on Cybernetics*, 2021.
- [133] Junhong Liu and Jouni Lampinen. A fuzzy adaptive differential evolution algorithm. Soft Computing, 9(6):448–462, 2005.
- [134] Marco Locatelli and Fabio Schoen. (global) optimization: Historical notes and recent developments. EURO Journal on Computational Optimization, 9:100012, 2021.
- [135] Søren Nymand Lophaven, Hans Bruun Nielsen, Jacob Søndergaard, et al. DACE: a Matlab kriging toolbox, volume 2. Citeseer, 2002.
- [136] Ilya Loshchilov. Surrogate-assisted evolutionary algorithms. PhD thesis, Université Paris Sud-Paris XI; Institut national de recherche en ..., 2013.

- [137] Ilya Loshchilov and Tobias Glasmachers. Black-box optimization competition. https://www.ini.rub.de/PEOPLE/glasmtbl/projects/bbcomp/.
- [138] Ilya Loshchilov, Marc Schoenauer, and Michele Sebag. Alternative restart strategies for cma-es. In International Conference on Parallel Problem Solving from Nature, pages 296–305. Springer, 2012.
- [139] Ilya Loshchilov, Marc Schoenauer, and Michele Sebag. Self-adaptive surrogateassisted covariance matrix adaptation evolution strategy. In *Proceedings of the 14th* annual conference on Genetic and evolutionary computation, pages 321–328, 2012.
- [140] Ilya Loshchilov, Marc Schoenauer, and Michèle Sebag. Intensive surrogate model exploitation in self-adaptive surrogate-assisted cma-es (saacm-es). In Proceedings of the 15th annual conference on Genetic and evolutionary computation, pages 439– 446, 2013.
- [141] David G Luenberger, Yinyu Ye, et al. Linear and nonlinear programming, volume 2. Springer, 1984.
- [142] Philipp Mall, Alexander Fidlin, Arne Krüger, and Heiko Groß. Simulation based optimization of torsional vibration dampers in automotive powertrains. *Mechanism* and Machine Theory, 115:244–266, 2017.
- [143] Rammohan Mallipeddi and Minho Lee. Surrogate model assisted ensemble differential evolution algorithm. In 2012 IEEE congress on evolutionary computation, pages 1–8. IEEE, 2012.
- [144] Rammohan Mallipeddi and Minho Lee. An evolving surrogate model-based differential evolution algorithm. Applied Soft Computing, 34:770–787, 2015.
- [145] M Mareli and B Twala. An adaptive cuckoo search algorithm for optimisation. Applied computing and informatics, 14(2):107–115, 2018.
- [146] Georges Matheron. Principles of geostatistics. *Economic geology*, 58(8):1246–1266, 1963.
- [147] Farrukh Mazhar, Abdul Munem Khan, Imran Ali Chaudhry, and Mansoor Ahsan. On using neural networks in uav structural design for cfd data fitting and classification. Aerospace Science and Technology, 30(1):210–225, 2013.

- [148] Temesgen Mengistu and Wahid Ghaly. Aerodynamic optimization of turbomachinery blades using evolutionary methods and ann-based surrogate models. Optimization and Engineering, 9(3):239–255, 2008.
- [149] Zbigniew Michalewicz, Dipankar Dasgupta, Rodolphe G Le Riche, and Marc Schoenauer. Evolutionary algorithms for constrained engineering problems. *Computers* & Industrial Engineering, 30(4):851–870, 1996.
- [150] Jonas Mockus. Bayesian approach to global optimization: theory and applications, volume 37. Springer Science & Business Media, 2012.
- [151] Ali Wagdy Mohamed, Anas A Hadi, Prachi Agrawal, Karam M Sallam, and Ali Khater Mohamed. Gaining-sharing knowledge based algorithm with adaptive parameters hybrid with imode algorithm for solving cec 2021 benchmark problems. In 2021 IEEE Congress on Evolutionary Computation (CEC), pages 841–848. IEEE, 2021.
- [152] Ali Wagdy Mohamed, Anas A Hadi, Ali Khater Mohamed, Prachi Agrawal, Abhishek Kumar, and P.N Suganthan. Problem definitions and evaluation criteria for the CEC 2021 special session and competition on single objective bound constrained numerical optimization. https://github.com/P-N-Suganthan/2021-SO-BCO/.
- [153] Ali Wagdy Mohamed, Anas A Hadi, Ali Khater Mohamed, and Noor H Awad. Evaluating the performance of adaptive gainingsharing knowledge based algorithm on cec 2020 benchmark problems. In 2020 IEEE Congress on Evolutionary Computation (CEC), pages 1–8. IEEE, 2020.
- [154] Hossein Mohammadi, Rodolphe Le Riche, and Eric Touboul. Making ego and cmaes complementary for global optimization. In International Conference on Learning and Intelligent Optimization, pages 287–292. Springer, 2015.
- [155] Daniel Molina, Javier Poyatos, Javier Del Ser, Salvador García, Amir Hussain, and Francisco Herrera. Comprehensive taxonomies of nature-and bio-inspired optimization: Inspiration versus algorithmic behavior, critical analysis recommendations. *Cognitive Computation*, 12(5):897–939, 2020.
- [156] Kumar Muthuraman and Haining Zha. Simulation-based portfolio optimization for large portfolios with transaction costs. Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics, 18(1):115–134, 2008.

- [157] Raymond H Myers, Douglas C Montgomery, and Christine M Anderson-Cook. Response surface methodology: process and product optimization using designed experiments. John Wiley & Sons, 2016.
- [158] Raymond H Myers and Raymond H Myers. Classical and modern regression with applications, volume 2. Duxbury press Belmont, CA, 1990.
- [159] Anh-Tuan Nguyen, Sigrid Reiter, and Philippe Rigo. A review on simulation-based optimization methods applied to building performance analysis. *Applied energy*, 113:1043–1058, 2014.
- [160] Duc Manh Nguyen and Nikolaus Hansen. Benchmarking cmaes-apop on the bbob noiseless testbed. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, pages 1756–1763, 2017.
- [161] Kouhei Nishida and Youhei Akimoto. Psa-cma-es: Cma-es with population size adaptation. In Proceedings of the Genetic and Evolutionary Computation Conference, pages 865–872, 2018.
- [162] Jorge Nocedal and Stephen J Wright. Numerical optimization. Springer, 1999.
- [163] Michał Okulewicz. Zastosowanie populacyjnych metaheurystyk uwzgledniajacych rozkład danych problemu do rozwiazywania problemu dynamicznej marszrutyzacji.
 PhD thesis, Department of Artificial Intelligence and Computational Methods, 2017.
- [164] Michał Okulewicz and Mateusz Zaborski. Benchmarking shade algorithm enhanced with model based optimization on the bbob noiseless testbed. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, pages 1259–1266, 2021.
- [165] Michał Okulewicz, Mateusz Zaborski, and Jacek Mańdziuk. Self-adapting particle swarm optimization for continuous black box optimization. Applied Soft Computing, 131:109722, 2022.
- [166] Ricardo A Olea. Geostatistics for engineers and earth scientists. Springer Science & Business Media, 2012.
- [167] Yew S Ong, Prasanth B Nair, and Andrew J Keane. Evolutionary optimization of computationally expensive problems via surrogate modeling. AIAA journal, 41(4):687–696, 2003.

- [168] Mark JL Orr et al. Introduction to radial basis function networks, 1996.
- [169] Carolina Osorio and Michel Bierlaire. A simulation-based optimization framework for urban transportation problems. *Operations Research*, 61(6):1333–1345, 2013.
- [170] Jeng-Shyang Pan, Nengxian Liu, Shu-Chuan Chu, and Taotao Lai. An efficient surrogate-assisted hybrid optimization algorithm for expensive optimization problems. *Information Sciences*, 561:304–325, 2021.
- [171] Quan-Ke Pan, M Fatih Tasgetiren, Ponnuthurai N Suganthan, and Tay Jin Chua. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *Information sciences*, 181(12):2455–2468, 2011.
- [172] MD Parno, T Hemker, and KR Fowler. Applicability of surrogates to improve efficiency of particle swarm optimization for simulation-based problems. *Engineering optimization*, 44(5):521–535, 2012.
- [173] Donald A Pierre. Optimization theory with applications. Courier Corporation, 1986.
- [174] Zbyněk Pitra, Lukáš Bajer, and Martin Holeňa. Doubly trained evolution control for the surrogate cma-es. In International Conference on Parallel Problem Solving from Nature, pages 59–68. Springer, 2016.
- [175] Rama Mohan Pokhrel, Jiro Kuwano, and Shinya Tachibana. A kriging method of interpolation used to map liquefaction potential over alluvial ground. *Engineering* geology, 152(1):26–37, 2013.
- [176] Dawid Połap and Marcin Woźniak. Polar bear optimization algorithm: Metaheuristic with fast population movement and dynamic birth and death mechanism. Symmetry, 9(10):203, 2017.
- [177] Petr Pošík and Václav Klemš. Jade, an adaptive differential evolution algorithm, benchmarked on the bbob noiseless testbed. In *Proceedings of the 14th annual* conference companion on Genetic and evolutionary computation, pages 197–204, 2012.
- [178] C Praveen and Regis Duvigneau. Low cost pso using metamodels and inexact preevaluation: Application to aerodynamic shape design. Computer Methods in Applied Mechanics and Engineering, 198(9-12):1087–1096, 2009.

- [179] A Kai Qin, Vicky Ling Huang, and Ponnuthurai N Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2):398–417, 2008.
- [180] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. Numerical mathematics, volume 37. Springer Science & Business Media, 2010.
- [181] Nestor V Queipo, Raphael T Haftka, Wei Shyy, Tushar Goel, Rajkumar Vaidyanathan, and P Kevin Tucker. Surrogate-based analysis and optimization. *Progress in aerospace sciences*, 41(1):1–28, 2005.
- [182] Khaled Rasheed and Haym Hirsh. Informed operators: Speeding up geneticalgorithm-based design optimization using reduced models. In *Proceedings of the* 2nd Annual Conference on Genetic and Evolutionary Computation, pages 628–635. Citeseer, 2000.
- [183] Alain Ratle. Accelerating the convergence of evolutionary algorithms by fitness landscape approximation. In International Conference on Parallel Problem Solving from Nature, pages 87–96. Springer, 1998.
- [184] Rommel G Regis. Particle swarm with radial basis function surrogates for expensive black-box optimization. *Journal of Computational Science*, 5(1):12–23, 2014.
- [185] Rommel G Regis and Christine A Shoemaker. Constrained global optimization of expensive black box functions using radial basis functions. *Journal of Global optimization*, 31(1):153–171, 2005.
- [186] Ahmad Rezaee Jordehi and Jasronita Jasni. Particle swarm optimisation for discrete optimisation problems: a review. *Artificial Intelligence Review*, 43(2):243–258, 2015.
- [187] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [188] Roman Rosipal and Leonard J Trejo. Kernel partial least squares regression in reproducing kernel hilbert space. Journal of machine learning research, 2(Dec):97– 123, 2001.
- [189] Francesca Rossi, Peter Van Beek, and Toby Walsh. Handbook of constraint programming. Elsevier, 2006.

- [190] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv* preprint arXiv:1609.04747, 2016.
- [191] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, pages 1–34, 1995.
- [192] Rohit Salgotra, Urvinder Singh, Sriparna Saha, and Atulya Nagar. New improved salshade-cnepsin algorithm with adaptive parameters. In 2019 IEEE Congress on Evolutionary Computation (CEC), pages 3150–3156. IEEE, 2019.
- [193] Karam M Sallam, Saber M Elsayed, Ripon K Chakrabortty, and Michael J Ryan. Improved multi-operator differential evolution algorithm for solving unconstrained problems. In 2020 IEEE Congress on Evolutionary Computation (CEC), pages 1–8. IEEE, 2020.
- [194] Ali H Sayed and Thomas Kailath. Recursive least-squares adaptive filters. The Digital Signal Processing Handbook, 21(1), 1998.
- [195] Gisbert Schneider, Johannes Schuchhardt, and Paul Wrede. Artificial neural networks and simulated molecular evolution are potential tools for sequence-oriented protein design. *Bioinformatics*, 10(6):635–645, 1994.
- [196] Mourad Sefrioui and Jacques Périaux. A hierarchical genetic algorithm using multiple models for optimization. In *International Conference on Parallel Problem Solving From Nature*, pages 879–888. Springer, 2000.
- [197] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings* of the IEEE, 104(1):148–175, 2015.
- [198] Songqing Shan and G Gary Wang. Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. *Structural and multidisciplinary optimization*, 41(2):219–241, 2010.
- [199] L Shi and K Rasheed. A survey of fitness approximation methods applied in evolutionary algorithms. In *Computational intelligence in expensive optimization problems*, pages 3–28. Springer, 2010.

- [200] Nazmul Siddique and Hojjat Adeli. Nature inspired computing: an overview and some future directions. *Cognitive computation*, 7(6):706–714, 2015.
- [201] Dan Simon. Evolutionary optimization algorithms. John Wiley & Sons, 2013.
- [202] Gyula Simon, Peter Volgyesi, Miklós Maróti, and Akos Ledeczi. Simulation-based optimization of communication protocols for large-scale wireless sensor networks. In *IEEE aerospace conference*, volume 3, pages 31339–31346, 2003.
- [203] Timothy W Simpson, Andrew J Booker, Dipankar Ghosh, Anthony A Giunta, Patrick N Koch, and R-J Yang. Approximation methods in multidisciplinary analysis and optimization: a panel discussion. *Structural and multidisciplinary optimization*, 27(5):302–313, 2004.
- [204] Saša Singer and John Nelder. Nelder-mead algorithm. Scholarpedia, 4(7):2928, 2009.
- [205] Urban Škvorc, Tome Eftimov, and Peter Korošec. Gecco black-box optimization competitions: progress from 2009 to 2018. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, pages 275–276, 2019.
- [206] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. Statistics and computing, 14(3):199–222, 2004.
- [207] András Sobester, Alexander Forrester, and Andy Keane. Engineering design via surrogate modelling: a practical guide. John Wiley & Sons, 2008.
- [208] Jaroslaw Sobieszczanski-Sobieski and Raphael T Haftka. Multidisciplinary aerospace design optimization: survey of recent developments. *Structural optimization*, 14(1):1–23, 1997.
- [209] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. arXiv preprint arXiv:0912.3995, 2009.
- [210] Vladimir Stanovov, Shakhnaz Akhmedova, and Eugene Semenkin. Nl-shade-rsp algorithm with adaptive archive and selective pressure for cec 2021 numerical optimization. In 2021 IEEE Congress on Evolutionary Computation (CEC), pages 809–816. IEEE, 2021.

- [211] Jörg Stork, Agoston E Eiben, and Thomas Bartz-Beielstein. A new taxonomy of global optimization algorithms. *Natural Computing*, pages 1–24, 2020.
- [212] Rainer Storn and Kenneth Price. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.
- [213] Dirk Sudholt. Computational complexity of evolutionary algorithms, hybridizations, and swarm intelligence. 2008.
- [214] P.N Suganthan. Code of top methods. https://github.com/P-N-Suganthan/ 2021-SO-BCO/blob/main/Codes-of-top-methods%20(1).zip.
- [215] Ryoji Tanabe and Alex Fukunaga. Success-history based parameter adaptation for differential evolution. In 2013 IEEE congress on evolutionary computation, pages 71–78. IEEE, 2013.
- [216] Ryoji Tanabe and Alex Fukunaga. Tuning differential evolution for cheap, medium, and expensive computational budgets. In 2015 IEEE Congress on Evolutionary Computation (CEC), pages 2018–2025. IEEE, 2015.
- [217] Ryoji Tanabe and Alex S Fukunaga. Improving the search performance of shade using linear population size reduction. In 2014 IEEE congress on evolutionary computation (CEC), pages 1658–1665. IEEE, 2014.
- [218] Yuanfu Tang, Jianqiao Chen, and Junhong Wei. A surrogate-based particle swarm optimization algorithm for solving optimization problems with expensive black box functions. *Engineering optimization*, 45(5):557–576, 2013.
- [219] Jason Teo. Exploring dynamic self-adaptive populations in differential evolution. Soft Computing, 10(8):673–686, 2006.
- [220] Saurabh Tewari, Umakant Dhar Dwivedi, and Susham Biswas. Intelligent drilling of oil and gas wells using response surface methodology and artificial bee colony. *Sustainability*, 13(4):1664, 2021.
- [221] Mateusz Uliński, Adam Żychowski, Michał Okulewicz, Mateusz Zaborski, and Hubert Kordulewski. Generalized Self-adapting Particle Swarm Optimization Algorithm. In Lecture Notes in Computer Science (including subseries Lecture Notes)

in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 3242, pages 29–40. Springer, Cham, 2018.

- [222] Holger Ulmer, Felix Streichert, and Andreas Zell. Evolution strategies with controlled model assistance. In Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753), volume 2, pages 1569–1576. IEEE, 2004.
- [223] Konstantinos Varelas, Anne Auger, Dimo Brockhoff, Nikolaus Hansen, Ouassim Ait ElHara, Yann Semet, Rami Kassab, and Frédéric Barbaresco. A comparative study of large-scale variants of cma-es. In *International conference on parallel problem* solving from nature, pages 3–15. Springer, 2018.
- [224] Shanti Verma and Kalyani Patel. Importance of heuristic algorithms for ontology based search of product in mobile-commerce. In 2018 2nd International Conference on Inventive Systems and Control (ICISC), pages 328–333. IEEE, 2018.
- [225] Pradnya A Vikhar. Evolutionary algorithms: A critical review and its future prospects. In 2016 International conference on global trends in signal processing, information computing and communication (ICGTSPICC), pages 261–265. IEEE, 2016.
- [226] G Gary Wang and Songqing Shan. Review of metamodeling techniques in support of engineering design optimization. 2007.
- [227] Zi Wang, Bolei Zhou, and Stefanie Jegelka. Optimization as estimation with gaussian processes in bandit settings. In *Artificial Intelligence and Statistics*, pages 1022–1031. PMLR, 2016.
- [228] Zhenglei Wei, Changqiang Huang, Xiaofei Wang, Tong Han, and Yintong Li. Nuclear reaction optimization: A novel and powerful physics-based algorithm for global optimization. *IEEE Access*, 7:66084–66109, 2019.
- [229] Sanford Weisberg. Applied linear regression. John Wiley & Sons, 2013.
- [230] David H Wolpert, William G Macready, et al. No free lunch theorems for search. Technical report, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.
- [231] Yu Xue, Jiongming Jiang, Binping Zhao, and Tinghuai Ma. A self-adaptive artificial bee colony algorithm based on global best for global optimization. *Soft Computing*, 22(9):2935–2952, 2018.

- [232] Takahiro Yamaguchi and Youhei Akimoto. Benchmarking the novel cma-es restart strategy using the search history on the bbob noiseless testbed. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, pages 1780–1787, 2017.
- [233] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.
- [234] Xin-She Yang. Nature-inspired optimization algorithms. Academic Press, 2020.
- [235] Xin-She Yang and Suash Deb. Engineering optimisation by cuckoo search. International Journal of Mathematical Modelling and Numerical Optimisation, 1(4):330– 343, 2010.
- [236] Xin Yao. An overview of evolutionary computation. Chinese Journal of Advanced Software Research, 3:12–29, 1996.
- [237] J Yazdi and SAA Salehi Neyshabouri. A simulation-based optimization model for flood management on a watershed scale. Water Resources Management, 26(15):4569–4586, 2012.
- [238] Milad Yousefi, Moslem Yousefi, and Flavio S Fogliatto. Simulation-based optimization methods applied in hospital emergency departments: A systematic review. *Simulation*, 96(10):791–806, 2020.
- [239] Tjalling J Ypma. Historical development of the newton-raphson method. SIAM review, 37(4):531–551, 1995.
- [240] Haibo Yu, Ying Tan, Chaoli Sun, and Jianchao Zeng. A generation-based optimal restart strategy for surrogate-assisted social learning particle swarm optimization. *Knowledge-Based Systems*, 163:14–25, 2019.
- [241] Mateusz Zaborski and Jacek Mańdziuk. Improving Ishade by means of a prescreening mechanism. In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '22, page 884–892, New York, NY, USA, 2022. Association for Computing Machinery.
- [242] Mateusz Zaborski and Jacek Mańdziuk. LQ-R-SHADE: R-SHADE with quadratic surrogate model. 2022. Accepted at International Conference on Artificial Intelligence and Soft Computing (ICAISC'22), Zakopane, Poland.

- [243] Mateusz Zaborski and Jacek Mańdziuk. Surrogate-assisted lshade algorithm utilizing recursive least squares filter. In International Conference on Parallel Problem Solving from Nature, pages 146–159. Springer, 2022.
- [244] Mateusz Zaborski, M Okulewicz, and Jacek Mańdziuk. Generalized self-adapting particle swarm optimization algorithm with model-based optimization enhancements. In 2nd PP-RAI Conference (PPRAI-19), pages 380–383, 2019.
- [245] Mateusz Zaborski, Michał Okulewicz, and Jacek Mańdziuk. Analysis of statistical model-based optimization enhancements in generalized self-adapting particle swarm optimization framework. *Foundations of Computing and Decision Sciences*, 45(3):233–254, 2020.
- [246] Mateusz Zaborski, Marcin Woźniak, and Jacek Mańdziuk. Multidimensional red fox meta-heuristic for complex optimization. Applied Soft Computing, 131:109774, 2022.
- [247] Tao Zeng, Hui Wang, Wenjun Wang, Tingyu Ye, and Luqi Zhang. Surrogate-assisted artificial bee colony algorithm. In International Conference on Bio-Inspired Computing: Theories and Applications, pages 262–271. Springer, 2021.
- [248] Dawei Zhan and Huanlai Xing. Expected improvement for expensive optimization: a review. *Journal of Global Optimization*, 78(3):507–544, 2020.
- [249] Zhi-Hui Zhan, Jun Zhang, Yun Li, and Henry Shu-Hung Chung. Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part* B (Cybernetics), 39(6):1362–1381, 2009.
- [250] Jie Zhang, Souma Chowdhury, and Achille Messac. An adaptive hybrid surrogate model. Structural and Multidisciplinary Optimization, 46(2):223–238, 2012.
- [251] Jingqiao Zhang and Arthur C Sanderson. Jade: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, 13(5):945–958, 2009.
- [252] Zongzhao Zhou, Yew Soon Ong, Prasanth B Nair, Andy J Keane, and Kai Yew Lum. Combining global and local surrogate models to accelerate evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(1):66–76, 2006.

Spis rysunków

2.1	Wysokopoziomowa klasyfikacja wybranych problemów optymalizacji. Ko-	
	lorem niebieskim zaznaczono właściwości stanowiące przedmiot zaintereso-	
	wania rozprawy.	27
2.2	Ilustracja optymalizacji czarnoskrzynkowej.	28
2.3	Przykład wykresu porównawczego, wygenerowanego przez COCO, dla	
	funkcji f_{10} w 20 wymiarach w domyślnej konfiguracji. Źródło: [74]	36
2.4	Przykład wykresu porównawczego, wygenerowanego przez COCO, dla	
	funkcji f_{10} w 20 wymiarach w scenariuszu kosztownym. Źródło: [74]	37
2.5	Podział optymalizacji ze względu dostępny budżet (liczbę ewalu acji f.c.). $% f(x)=\int f(x) dx$	40
3.1	Wysokopoziomowa klasyfikacja wybranych metod rozwiązujących proble- my optymalizacji ciągłej z jednym celem. Na szaro zaznaczono metody znaj- dują zastosowanie w optymalizacji spełniającej paradygmat czarnoskrzyn- kowości. Na niebiesko zaznaczono algorytmy z rodziny DE oraz algorytmy z rodziny CMA-ES.	43
4.1	Gradacja sposobów pozyskania informacji o wartości funkcji celu w danym punkcie. Kolorem szarym zaznaczono sposoby właściwe dla algorytmów	
	populacyjnych wykorzystujące metamodele.	61
4.2	Sposoby wykorzystania metamodeli.	63
4.3	Przykład perceptronu wielowarstwowego z jedną warstwą ukrytą. \ldots .	69
6.1	Porównywanie sposobu konstruowania zbiorów uczących w metamodelu	
	kwadratowym oraz metamodelu wielomianowym. $\hfill\hf$	106
6.2	Wyniki (na zbiorze testowym COCO) M-GAPSO w wariantach: DE, PD oraz PDLP dla wszystkich 24 funkcji dla $D \in \{2, 3, 5, 10, 20, 40\}$, w budżecie	
	optymalizacji $10^6 \cdot D$ ewaluacji f.c.	108

6.3	Wyniki (na zbiorze testowym COCO) M-GAPSO w wariantach: DE, PD oraz PDLP dla wszystkich 24 funkcji w podziale na kategorie funkcji dla $D = 10$, w budżecie optymalizacji $10^6 \cdot D$ ewaluacji f.c
6.4	Wyniki (na zbiorze testowym COCO) M-GAPSO w wariancie PDLP oraz PDLPk dla wszystkich 24 funkcji (wykres lewy) oraz funkcji separowalnych $f_1 - f_5$ (wykres prawy) dla $D = 10$ w budżecie optymalizacji $10^6 \cdot D$ ewaluacji f.c
6.5	Wyniki (na zbiorze testowym COCO) SHADE-LM w zestawieniu z wynika- mi bazowego R-SHADE dla wszystkich 24 funkcji w podziale na kategorie funkcji dla $D \in \{2, 5, 10, 20\}$ w budżecie optymalizacji $10^6 \cdot D$ ewaluacji f.c. 115
7.1	Wyniki (na zbiorze testowym COCO) LQ-R-SHADE (ozn. jako LQ-R-SH) oraz init-R-SHADE (ozn. jako init-R-SH) w zestawieniu z wynikami R-SHADE (ozn. jako R-SH) dla wszystkich 24 funkcji dla $D \in \{2,3,5,10,20,40\}$, w budżecie optymalizacji $10^3 \cdot D$ ewaluacji f.c
7.2	Wizualizacje 3D funkcji $(D = 2)$ ze zbioru testowego CEC2021: f_1 (a) , f_2 (b), f_3 (c) oraz f_8 (d). Rysunki pochodzą z pracy [152]
7.3	Uśredniony błąd psLSHADE (niebieska linia) oraz L-SHADE (czerwona linia) dla funkcji f_1 , f_2 , f_3 oraz f_8 ($D = 20$) w budżecie optymalizacji $10^3 \cdot D$. Na osi x zaznaczono liczbę ewaluacji f.c., a na osi y średni błąd ϵ . 131
7.4	Uśredniona hiperobjętość psLSHADE (niebieska linia) oraz L-SHADE (czerwona linia) dla funkcji f_1 , f_2 , f_3 oraz f_8 ($D = 20$) w budżecie optyma- lizacji $10^3 \cdot D$. Na osi x zaznaczono liczbę ewaluacji f.c., a na osi y średnią hiperobjętość
7.5	Uśredniona precyzja metamodelu w psLSHADE dla funkcji f_1 , f_2 , f_3 oraz f_8 ($D = 20$), w budżecie optymalizacji $10^3 \cdot D$. Na osi x zaznaczono liczbę ewaluacji f.c. jaka została wykonana do iteracji g włącznie, a na osi y średnią precyzję
7.6	Uśredniona wartość R^2 (kolor czerwony) oraz τ -Kendalla (kolor niebieski) dla funkcji f_1 , f_2 , f_3 oraz f_8 ($D = 20$) w budżecie optymalizacji $10^3 \cdot D$. Na osi x zaznaczono liczbę ewaluacji f.c., jaka została wykonana do iteracji g włącznie, a na osi y średnie wartości R^2 oraz τ -Kendalla
Spis tabel

2.1	Budżety optymalizacji (rozumiane jako dostępna liczba ewaluacji f.c.) przy- jęte w popularnych zbiorach testowych (Z.T.) w podziale na liczbę wymia- rów problemu (D). W przypadku gdy benchmark nie wykorzystuje pro- blemów o danej liczebnie wymiarów wartość budżetu oznaczono jako Ø.	38
51	Porównanie czasu [s] estymacji parametrów metamodeli (Mm.) ze względu	
5.1	na rozmiar $K \in \{50, 100, 500, 1000, 2000, 4000\}$ zbioru uczącego \mathcal{D} . Liczba wymiarów problemu jest stała i wynosi $D = 20$. W porównaniu uwzględ- niono metamodel Kriginga, metamodel wykorzystujący RBF, kwadratową	
	RW (ozn. jako k. RW) oraz kwadratową RW z interakcjami (ozn. jako k.	
	RW + in.).	87
5.2	Porównanie czasu [s] estymacji parametrów metamodeli (Mm.) ze względu na liczbę wymiarów problemu $D \in \{2, 5, 10, 20, 30, 40, 60\}$. Rozmiar zbioru uczącego \mathcal{D} jest stały i wynosi $K = 2000$. W porównaniu uwzględniono metamodel Kriginga, metamodel wykorzystujący RBF, kwadratową RW (ozn. jako k. RW) oraz kwadratową RW z interakcjami (ozn. jako k. RW)	
	+ in.)	88
5.3	Opis metamodeli stosowanych w mechanizmie inicjalizacji metamodelem. Estymowane współczynniki przy każdej ze zmiennych zostały pominięte	
	dla zwiększenia czytelności.	92
6.1	Parametry M-GAPSO.	107
6.2	Opis kaskady metamodeli stosowanych w SHADE-LM. Wykorzystywane metamodele to: kwadratowa RW (ozn. jako kw. RW) oraz kwadratowa RW z interakcjami (ozn. jako kw. RW + in.). Estymowane współczynniki przy	
	każdej ze zmiennych zostały pominięte dla zwiększenia czytelności.	113

6.3	Parametry SHADE-LM	114
6.4	Empiryczna złożoność obliczeniowa M-GAPSO w wariantach DE, PDa oraz PDLPa, mierzona zgodnie z procedurą CEC2021 (rozdział 2.4.3). T_0 jest czasem działania testowego programu, T_1 jest czasem $2 \cdot 10^5$ ewaluacji funk- cji f_{18} z CEC2017. T_2 jest średnim czasem działania algorytmu obejmujące- go $2 \cdot 10^5$ ewaluacji f.c. $T_3 = (T_2 - T_1)/T_0$ jest finalną empiryczną złożonością obliczeniową.	116
7.1	Opis kaskady metamodeli stosowanych w preselekcji w LQ-R-SHADE. Wy- korzystywane metamodele to: liniowa RW (ozn. jako lin. RW), kwadratowa RW (ozn. jako kw. RW) oraz kwadratowa RW z interakcjami (ozn. jako kw. RW + in.). Estymowane współczynniki przy każdej ze zmiennych zostały	
	pominięte dla zwiększenia czytelności.	122
7.2	Parametry R-SHADE i LQ-R-SHADE. $r(\cdot)$ zwraca argument zaokrąglony do najbliższej liczby całkowitej.	123
7.3	Opis transformacji oraz finalnej postaci metamodelu stosowanych w psLSHADE. Estymowane współczynniki przy każdej ze zmiennych zostały	
	pominięte dla zwiększenia czytelności.	127
7.4	Parametry L-SHADE i psLSHADE.	128
7.5	Wyniki (na zbiorze testowym CEC2021) ps LSHADE dla różnych wartości mnożnika osobników ($N_s \in \{2, 5, 10, 20\}$) w budżecie optymalizacji $10^3 \cdot D$	
	ewaluacji f.c.	128
7.6	Wyniki (na zbiorze testowym CEC2021) psLSHADE ($N_s = 5$) w zestawie- niu z wynikami L-SHADE oraz MadDE w budżetach optymalizacji $10^2 \cdot D$,	
	$10^3 \cdot D$ oraz $10^4 \cdot D$ ewaluacji f.c.	129
7.7	Opis metamodelu kwadratowej RW z interakcjami (ozn. jako kw. RW + in.) używanego w rmmLSHADE. Estymowane współczynniki przy każdej	190
	ze zmiennych zostały pominięte dla zwiększenia czytelności.	139
7.8	Parametry L-SHADE i rmmLSHADE	140
7.9	Wyniki (na zbiorze testowym CEC2021) rmmLSHADE z $\lambda \in$ {0.98,0.99,1.0} w zestawieniu z wynikami L-SHADE oraz psLSHA- DE w budżecie optymalizacji 10 ³ · D. $\lambda = \emptyset$ oznacza wariant bez adaptacji	
	parametrów metamodelu za pomocą filtra RLS	140