POLITECHNIKA
BYDGOSKA
im. Jana i Jędrzeja Śniadeckich

RADA NAUKOWA DYSCYPLINY
INFORMATYKA TECHNICZNA I TELEKOMUNIKACJA

ROZPRAWA DOKTORSKA

mgr inż. Jan Edward Baumgart

Zastosowanie starożytnych strategii bitew w nowych algorytmach rojowych do rozwiązywania problemów optymalizacyjnych

Applying Ancient Battle Strategies to New Swarm Algorithms to Solve Optimization Problems

DZIEDZINA: nauki inżynieryjno-techniczne
DYSCYPLINA: informatyka techniczna i telekomunikacja

PROMOTOR
dr hab. inż. Jacek M. Czerniak, prof. Politechniki Bydgoskiej
Politechnika Bydgoska im. Jana i Jędrzeja Śniadeckich,
Wydział Telekomunikacji, Informatyki i Elektrotechniki

PROMOTOR POMOCNICZY
dr inż. Dawid Ewald, prof. Politechniki Bydgoskiej
Politechnika Bydgoska im. Jana i Jędrzeja Śniadeckich,
Wydział Telekomunikacji, Informatyki i Elektrotechniki

Bydgoszcz 2026

PODZIĘKOWANIA

Pragnę wyrazić serdeczne podziękowania mojemu promotorowi, Panu dr. hab. inż. Jackowi M. Czerniakowi, za wieloletnią opiekę naukową, inspirację oraz nieocenione wskazówki i zaufanie. Doskonale pamiętam, jak ponad dziesięć lat temu podszedłem do Pana Profesora po wykładzie ze „Wstępu do sztucznej inteligencji", poświęconym algorytmom mrówkowym – i jak się ostatecznie okazało, od tamtej chwili nie zająłem się już w życiu na poważnie niczym innym.

W założeniu Pan Profesor miał jedynie nadzorować powstawanie moich prac; w praktyce ukształtował całą moją naukową drogę. Wpływ wykroczył przy tym daleko poza sprawy badawcze – to między innymi dzięki Profesorowi poznałem moją żonę – przez co bez wątpienia zasługuje On na miano niekwestionowanego rekordzisty świata w kategorii „wpływ promotora na losy doktoranta". Dziękuję za wszystko.

Słowa wdzięczności kieruję również do promotora pomocniczego, Pana dr. inż. Dawida Ewalda, za cenne uwagi merytoryczne i zaangażowanie. Nasza współpraca dowiodła przy tym, że potrafi przetrwać najgorszą pogodę i najgorszą łódź. Dziękuję za każdą cenną uwagę, wspólnie przepłyniętą milę oraz za to, że ostatecznie nie wyrzuciłeś mnie za burtę pomimo licznych okazji.

Szczególne podziękowania kieruję do mojej żony, Marceliny, bez której ukończenie tej pracy nie byłoby możliwe. Gdy próbowałem łączyć pracę zawodową, obowiązki na uczelni i przygotowanie rozprawy, to Ona wzięła na swoje barki troskę o dom i dzieci, znajdując przy tym siłę, by o drugiej w nocy ze spokojem wysłuchiwać zapewnień, że „komputer będzie szumieć obliczeniami jeszcze tylko kilka dni". Za Twoje nieustające wsparcie, cierpliwość i wyrozumiałość jestem Ci głęboko wdzięczny.

Dziękuję również moim córkom – Lilii, która niezawodnie dbała o moją równowagę między pracą a życiem, wpadając z żądaniem „włącz mi piosenkę" w sam środek statystyki, oraz maleńkiej Amelce, której największym wkładem badawczym było to, że na ostatniej prostej pozwalała mi spokojnie pracować. Dziewczyny, jesteście moim największym sukcesem – przy Was każda dysertacja to tylko plik tekstu.

Słowa najgłębszej wdzięczności kieruję do moich Rodziców. Codziennie dawaliście z siebie wszystko, abym mógł się rozwijać i marzyć dalej niż horyzont rodzinnych pól. Inwestowaliście we mnie i rodzeństwo. Mamo, Tato – pewnie wyobrażaliście sobie nieco inny finał porannych ćwiczeń z instrumentem; mój doktorat jest w zasadzie również koncertem, z tą różnicą, że oklasków raczej nie będzie, a zamiast bisów co najwyżej lista poprawek. Każdy Wasz wysiłek zaprowadził mnie do tego momentu, a ta praca jest w równej mierze Wasza. Dziękuję.

Wszystkim, którzy przyczynili się do powstania niniejszej rozprawy, składam wyrazy szczerej wdzięczności.

4:84946581

METRYKA PRACY DYPLOMOWEJ

Dane ogólne

Nazwa Uczelni	Politechnika Bydgoska im. Jana i Jędrzeja Śniadeckich
Wydział	Telekomunikacji, Informatyki i Elektrotechniki
Dane autora	mgr inż. Jan Edward Baumgart
Dane promotora	dr hab. inż. Jacek M. Czerniak, prof. Politechniki Bydgoskiej
Dane promotora pomocniczego	dr inż. Dawid Ewald, prof. Politechniki Bydgoskiej

Dane dotyczące pracy dyplomowej

Język pracy	język polski [PL]
Tytuł pracy	Zastosowanie starożytnych strategii bitew w nowych algorytmach rojowych do rozwiązywania problemów optymalizacyjnych
Opis pracy	Rozprawa przedstawia algorytm metaheurystyczny Ancient Battlefield Optimizer (ABO) inspirowany starożytnymi strategiami bitew. ABO wykorzystuje sześć taktyk bitewnych (falanga, szyk ukośny, przebicie centrum, oskrzydlenie, okrążenie, zwiad) do eksploracji i eksploatacji przestrzeni rozwiązań. Opracowano też system Commander AI oparty na Q-learningu, który dobiera taktyki w trakcie optymalizacji. Pełny benchmark objął 42 metaheurystyki porównawcze oraz 4 warianty ABO na 33 funkcjach benchmarkowych w 5 wymiarach.
Typ pracy	doktorska
Streszczenie	Rozprawa opisuje Ancient Battlefield Optimizer (ABO) – algorytm rojowy inspirowany starożytnymi taktykami wojskowymi. Opracowano sześć taktyk bitewnych mapujących starożytne strategie na operatory optymalizacyjne: falangę i przebicie centrum (eksploatacja lokalna), oskrzydlenie i zwiad (eksploracja globalna) oraz szyk ukośny i okrążenie jako strategie hybrydowe wspierające dywersyfikację oraz unikanie minimów lokalnych. ABO zawiera system rozpoznania mapujący przestrzeń poszukiwań i mechanizm honorów wpływający na zachowanie agentów. W wariancie CommanderABO moduł dowodzenia oparty na Q-learningu dobiera taktyki w trakcie optymalizacji. Pełny benchmark objął 33 funkcje benchmarkowe w 5 wymiarach (2D–100D), łącznie 759 000 uruchomień w końcowym zbiorze rangowym (33 funkcje $\times$ 5 wymiarów $\times$ 100 przebiegów $\times$ 46 algorytmów). Test Friedmana ($\chi^2 = 3775{,}96$, $p < 0{,}001$) odrzucił hipotezę zerową. Cztery warianty ABO zajęły cztery pierwsze miejsca rankingu Friedmana (rangi 6,21–6,88 wobec 10,02 dla najlepszego algorytmu spoza rodziny, GWO); najlepszą średnią rangę w kryterium mediany osiągnął ręcznie strojony ABO-ManualPhases (6,21), a adaptacyjny CommanderABO uplasował się tuż za nim (6,77). W kryterium najlepszego wyniku (best-of-run) to CommanderABO uzyskał najlepszą średnią rangę w całym polu 46 algorytmów, co wskazuje na najwyższy potencjał jakości rozwiązań. W ujęciu rang minimalnych dla remisów CommanderABO znalazł się w top-3 na 81 ze 165 par funkcja–wymiar oraz w top-10 na 141 parach, przy umiarkowanym koszcie obliczeniowym (mediana 1,37 s na uruchomienie). Przewaga rodziny ABO była najsilniejsza dla $D \geq 10$; w 2D lepsze wyniki osiągały wybrane algorytmy klasyczne (MFO, ABC, GWO). Architektura ABO ma charakter modułowy i umożliwia dodawanie nowych strategii.
Słowa kluczowe	algorytmy rojowe, metaheurystyki, optymalizacja, uczenie ze wzmocnieniem, Q-learning, strategie bitew

DIPLOMA THESIS RECORD

General information

University name	Bydgoszcz University of Science and Technology
Faculty	Telecommunications, Computer Science and Electrical Engineering
Author's information	Jan Edward Baumgart, M.Sc. Eng.
Supervisor's information	Jacek M. Czerniak, D.Sc. Eng., Associate Professor
Secondary supervisor's information	Dawid Ewald, Ph.D. Eng., Associate Professor

Data regarding the diploma thesis

Language of thesis	Polish [PL]
Title of thesis	Applying Ancient Battle Strategies to New Swarm Algorithms to Solve Optimization Problems
Description	This doctoral dissertation presents a novel metaheuristic algorithm called Ancient Battlefield Optimizer (ABO) inspired by ancient battle strategies. The algorithm employs six battle tactics (phalanx, oblique order, center penetration, flanking, surrounding, scouting) for exploration and exploitation of the solution space. Additionally, a Commander AI system based on reinforcement learning (Q-learning) was developed to adaptively select tactics during the optimization process. Extensive comparative experiments were conducted with 42 comparison metaheuristics and 4 ABO variants on 33 benchmark functions across 5 dimensions.
Type of thesis	Doctoral
Abstract	This dissertation presents the Ancient Battlefield Optimizer (ABO) – a novel swarm algorithm inspired by ancient military tactics. Six battle tactics were developed, mapping ancient strategies to optimization operators: phalanx and center penetration (local exploitation), flanking and scouting (global exploration), and oblique order and surrounding as hybrid strategies supporting diversification and avoidance of local optima. ABO incorporates a reconnaissance system that maps the search space, and features an honor mechanism that influences agent behavior. In the CommanderABO variant, a command module based on Q-learning adaptively selects tactics during optimization. The full benchmark covered 33 benchmark functions across 5 dimensions (2D–100D), totaling 759,000 runs with 46 algorithms (42 comparison metaheuristics + 4 ABO variants). The Friedman test ($\chi^2 = 3775.96$, $p < 0.001$) rejected the null hypothesis. The four ABO variants occupied the top four positions of the Friedman ranking (ranks 6.21–6.88 vs. 10.02 for the best non-ABO algorithm, GWO); the hand-tuned ABO-ManualPhases achieved the best average rank under the median criterion (6.21), with the adaptive CommanderABO close behind (6.77). Under the best-of-run criterion, however, CommanderABO attained the best mean rank across the entire 46-algorithm field, indicating the highest solution-quality ceiling. Using minimum ranks for ties, CommanderABO ranked in the top 3 on 81 of 165 function–dimension pairs and in the top 10 on 141 pairs, at a moderate computational cost (median 1.37 s per run). The advantage of the ABO family was strongest for $D \geq 10$; in 2D, selected classical algorithms (MFO, ABC, GWO) performed better. The ABO architecture is modular and allows for adding new strategies.
Keywords	swarm algorithms, metaheuristics, optimization, reinforcement learning, Q-learning, battle strategies

SPIS TREŚCI

11:10560212

WYKAZ SKRÓTÓW I SYMBOLI

Skróty ogólne

ABO	Ancient Battlefield Optimizer – główny algorytm opracowany w ramach rozprawy
SSI	Strategic Swarm Intelligence – strategiczna inteligencja rojowa (kategoria taksonomiczna)
CEC	Congress on Evolutionary Computation – zbiór funkcji benchmarkowych
RL	Reinforcement Learning – uczenie ze wzmocnieniem
MDP	Markov Decision Process – proces decyzyjny Markowa
AOS	Adaptive Operator Selection – adaptacyjny dobór operatorów
NFL	No Free Lunch theorem – twierdzenie o braku darmowego obiadu (Wolpert & Macready, 1997)
FE	Function Evaluations – liczba wywołań funkcji celu (miara kosztu obliczeniowego)
CI	Confidence Interval – przedział ufności

Algorytmy porównawcze użyte w benchmarku (42 metaheurystyki)

Algorytmy rojowe (Swarm Intelligence):

ABC	Artificial Bee Colony – sztuczna kolonia pszczół (Karaboga, 2005)
ACOR	Ant Colony Optimization (dziedziny ciągłe) – optymalizacja kolonią mrówek (Socha & Dorigo, 2008)
ALO	Ant Lion Optimizer – optymalizator mrówkolwa (Mirjalili, 2015)
BA	Bat Algorithm – algorytm nietoperzy (Yang, 2010)
BeesA	Bees Algorithm – algorytm pszczeli (Pham i in., 2006)
BSA	Bird Swarm Algorithm – algorytm ptasiego stada (Meng i in., 2016)
CSA	Crow Search Algorithm – algorytm wron (Askarzadeh, 2016)
DO	Dragonfly Optimization – optymalizacja ważek (Mirjalili, 2016)
FA	Fireworks Algorithm – algorytm fajerwerków (Tan & Zhu, 2010)
FOA	Fruit-fly Optimization Algorithm – algorytm muszki owocowej (Pan, 2012)
GOA	Grasshopper Optimization Algorithm – algorytm szarańczy (Saremi i in., 2017)
GWO	Grey Wolf Optimizer – optymalizator szarych wilków (Mirjalili, 2014)
JA	Jaya Algorithm – algorytm Jaya (Rao, 2016)

Algorytmy ewolucyjne (Evolutionary):

CRO	Coral Reefs Optimization – optymalizacja raf koralowych (Salcedo-Sanz i in., 2014)
DE	Differential Evolution – ewolucja różnicowa (Storn & Price, 1997)
EP	Evolutionary Programming – programowanie ewolucyjne (Fogel, 1966)

ES	Evolution Strategies – strategie ewolucyjne (Rechenberg, 1965)
FPA	Flower Pollination Algorithm – zapylanie kwiatów (Yang, 2012)
GA	Genetic Algorithm – algorytm genetyczny (Holland, 1975)
MA	Memetic Algorithm – algorytm memetyczny (Moscato, 1989)

Algorytmy inspirowane fizyką (Physics-Inspired):

ASO	Atom Search Optimization – przeszukiwanie atomów (Zhao, 2019)
EFO	Electromagnetic Field Optimization – optymalizacja pola elektromagnetycznego (Abedinpourshotorban i in., 2016)
HGSO	Henry Gas Solubility Optimization – rozpuszczalność gazów (Hashim i in., 2019)
SA	Simulated Annealing – symulowane wyżarzanie (Kirkpatrick, 1983)
TWO	Tug of War Optimization – przeciąganie liny (Kaveh & Zolghadr, 2016)

Algorytmy inspirowane człowiekiem (Human-Based):

BRO	Battle Royale Optimization – optymalizacja battle royale (Rahkar Farshi, 2020)
BSO	Brain Storm Optimization – burza mózgów (Shi, 2011)
CA	Cultural Algorithm – algorytm kulturowy (Reynolds, 1994)
CHIO	Coronavirus Herd Immunity Optimization – odporność stadna (Al-Betar i in., 2021)
ICA	Imperialist Competitive Algorithm – rywalizacja imperiów (Atashpaz-Gargari, 2007)

Algorytmy bio-inspirowane (Bio-Inspired):

BBO	Biogeography-Based Optimization – biogeografia (Simon, 2008)
BFO	Bacterial Foraging Optimization – bakteryjny foraging (Passino, 2002)
EOA	Earthworm Optimization Algorithm – algorytm dżdżownic (Wang, 2018)
GCO	Germinal Center Optimization – optymalizacja ośrodków rozmnażania (Villaseñor i in., 2018)
IWO	Invasive Weed Optimization – chwasty inwazyjne (Mehrabian, 2006)
SBO	Satin Bowerbird Optimization – altannik atłasowy (Moosavi & Bardsiri, 2017)

Algorytmy matematyczne i inne:

CEM	Cross-Entropy Method – metoda entropii krzyżowej (Rubinstein, 1997)
HC	Hill Climbing – wspinaczka górska (klasyka AI, Newell & Simon, lata 60.)
HS	Harmony Search – poszukiwanie harmonii (Geem i in., 2001)

Algorytmy hybrydowe i pozostałe:

CSO	Cat Swarm Optimization – rój kotów (Chu, 2006)
GSKA	Gaining-Sharing Knowledge Algorithm – zdobywanie i dzielenie wiedzy (Mohamed, 2019)
MFO	Moth-Flame Optimization – ćmy i płomień (Mirjalili, 2015)

Symbole matematyczne

D	Wymiarowość przestrzeni rozwiązań
N	Liczba agentów (rozmiar populacji)
T	Maksymalna liczba iteracji
t	Aktualna iteracja
x_i	Pozycja i-tego agenta w przestrzeni rozwiązań
x^*	Najlepsze znalezione rozwiązanie (lider)
$f(x)$	Funkcja celu (fitness)
lb, ub	Dolna i górna granica przestrzeni przeszukiwań
α, β, γ	Parametry kontrolne algorytmu
$Q(s, a)$	Wartość Q dla stanu s i akcji a w Q-learningu
ϵ	Współczynnik eksploracji w strategii ϵ-greedy
χ^2	Statystyka chi-kwadrat w teście Friedmana
R_j	Średni ranking algorytmu j
d	Współczynnik wielkości efektu Cohena

Terminologia wojskowa i algorytmiczna ABO

Jednostki i role:

Heavy Infantry	Ciężka piechota – jednostki eksploatacyjne, silna presja lokalna
Light Infantry	Lekka piechota – jednostki eksploracyjne, szybki ruch
Archers	Łucznicy – jednostki o zasięgu, wspierają eksplorację
Cavalry	Kawaleria – jednostki mobilne, szybka eksploracja przestrzeni
Chariots	Rydwany – jednostki hybrydowe, balans eksploracji i eksploatacji
War Elephants	Słonie bojowe – jednostki o dużej sile, lokalna intensyfikacja
Hero (status)	Bohater – status jednostki o wysokim honorze
Runagate (status)	Dezerter – status jednostki o ujemnym honorze
Scout	Zwiadowca – jednostka zarządzana przez moduł rozpoznania

Taktyki:

Phalanx	Falanga – starożytna formacja piechoty, lokalna eksploatacja
Oblique Order	Szyk skośny – atak skoncentrowany na flance przeciwnika
Center Penetration	Przełamanie centrum – atak koncentryczny na środek formacji

Flanking	Oskrzydlenie – manewr obejścia z boku
Surrounding	Okrążenie – całkowite otoczenie przeciwnika
Scouting	Rozpoznanie – zbieranie informacji o przestrzeni

Moduły systemu:

Commander AI	Moduł dowodzenia – Q-learning do wyboru taktyk
Reconnaissance	Rozpoznanie – system zwiadowców i mapa inteligencji
Honor System	System honorów – mechanizm awansów/degradacji jednostek
Formation	Formacja – grupowanie jednostek dla koordynacji ruchu

Metryki:

Fitness	Wartość funkcji celu dla danego rozwiązania
Honor	Skumulowana miara wydajności jednostki
Intelligence Map	Mapa inteligencji – siatka z informacjami o przestrzeni
Hardness Index	Indeks trudności funkcji benchmarkowej

Terminy statystyczne

Friedman test	Test nieparametryczny dla porównania wielu grup
Wilcoxon test	Test nieparametryczny dla dwóch grup sparowanych
Nemenyi post-hoc	Test post-hoc po teście Friedmana
Holm-Bonferroni	Metoda korekty dla wielokrotnych porównań (FWER)
FWER	Family-Wise Error Rate – prawdopodobieństwo błędu typu I
FDR	False Discovery Rate – odsetek fałszywych odkryć
Cohen's d	Miara wielkości efektu dla dwóch grup
CD	Critical Difference – różnica krytyczna (test Nemenyiego)
CD diagram	Diagram różnic krytycznych (Critical Difference)

16:12255724

1. WSTĘP

„Lepsze jest wrogiem dobrego”
„Le mieux est l’ennemi du bien.” (fr.)
– Voltaire

Rozdział omawia podstawy optymalizacji i algorytmów rojowych (Sekcja 1.1), motywacje i cele pracy (Sekcja 1.2), hipotezy badawcze (Sekcja 1.3) oraz strukturę rozprawy (Sekcja 1.4).

1.1 WPROWADZENIE DO PROBLEMU OPTYMALIZACJI

Każda praca naukowa, zgodnie z przyjętymi standardami akademickimi, w pierwszej kolejności powinna wskazać i zarysować podstawy teoretyczne, na których będzie się opierać. To nie tylko konwencja, ale i konieczność – to podstawa, na której czytelnik zrozumie, w jaki nowy świat wchodzi, będący specyfiką problemu. Dlatego przed omówieniem zaawansowanych tematów algorytmów rojowych w optymalizacji należy wspomnieć o tym, co stanowi sam rdzeń dziedziny i czym jest optymalizacja.

Problem optymalizacji polega na znalezieniu najlepszego rozwiązania spośród wszystkich dopuszczalnych [87, 20]. Jakość rozwiązania jest określana przez funkcję celu, która wyraża preferencje w badaniach. Formalnie optymalizacja sprowadza się do minimalizacji lub maksymalizacji funkcji celu w przestrzeni rozwiązań ograniczonej warunkami granicznymi.

Wartość funkcji celu $f(\mathbf{x})$ dla danego rozwiązania $\mathbf{x}$ nazywana jest *wartością przystosowania* (ang. *fitness*). Im niższa wartość fitness (przy minimalizacji), tym lepsze rozwiązanie. Termin ten jest powszechnie stosowany w literaturze algorytmów metaheurystycznych i będzie używany w dalszej części pracy zamiennie z pojęciem wartości funkcji celu.

Ekstremum funkcji $f(x)$ to taka wartość x, która maksymalizuje lub minimalizuje funkcję celu, zależnie od typu problemu. Poszukuje się go w przestrzeni przeszukiwania – zbiorze wszystkich wartości x zgodnych z ograniczeniami – a proces ten jest podstawowym elementem każdej analizy optymalizacyjnej.

Ograniczenia mogą mieć różne formy; na przykład, mogą dotyczyć jedynie matematycznych warunków, jak dodatniość zmiennych, bądź też bardziej złożonych relacji, które opisują problem. Na przykład, w przypadku optymalizacji wymiarów pudełka niezbędne będzie uwzględnienie zarówno warunków geometrycznych, jak i funkcjonalnych: wymiary muszą być dodatnie, ale jednocześnie na tyle ograniczone, by pojemnik nadal spełniał swoją funkcję przechowywania i nie stał się skrzynią. Z kolei w procesach przemysłowych, takich jak spawanie dwóch belek stalowych, ograniczenia determinują, czy użyte elementy podczas łączenia pozostaną belką, czy może belka stanie się blachą – istotna jest tutaj nie tylko geometria, ale również własności materiałowe i technologiczne.

Specjalistyczne metody rozwiązywania problemów optymalizacyjnych można rozpatrywać w kontekście sposobu rozpatrywania ograniczeń. Najpopularniejszą metodą jest zastosowanie funkcji karnych, które sztucznie zwiększają wartość funkcji celu dla rozwiązań wprowadzających zmiany w narzuconych warunkach. Jest to jednak tylko jeden z wielu sposobów.

1.1.1 Cechy problemów optymalizacji

Rozwiązywanie problemów optymalizacyjnych wymaga znajomości ich właściwości. Każdy problem ma cechy, które wpływają na dobór metod. Najczęściej są to: ciągłość, liniowość, modalność i wypukłość.

Ciągłość: Charakter funkcji celu – czy jest ciągła, czy nieciągła – to jeden z głównych elementów problemu. W funkcjach ciągłych drobne zmiany argumentów powodują proporcjonalne zmiany wartości, co umożliwia stosowanie metod analitycznych, np. optymalizacji gradientowej. Funkcje nieciągłe mogą zmieniać wartości skokowo, co utrudnia optymalizację i wymaga metod numerycznych, często opartych na heurystykach.

Liniowość: Optymalizacja może dotyczyć zarówno problemów liniowych, jak i nieliniowych. W przypadkach liniowych zarówno funkcja celu, jak i wszystkie ograniczenia przyjmują postać liniową. W optymalizacji nieliniowej funkcja celu lub ograniczenia są nieliniowe, co znacznie zwiększa złożoność obliczeniową. Niektóre z problemów nieliniowych da się przekształcić w postać quasi-liniową, ale w wielu wypadkach konieczne stało się stosowanie algorytmów iteracyjnych.

Modalność: Modalność funkcji celu odnosi się do liczby ekstremów lokalnych i globalnych. W problemach unimodalnych istnieje tylko jedno globalne optimum, które jest zarazem jedynym optimum lokalnym – każde ulepszenie prowadzi wprost do najlepszego rozwiązania. Funkcje multimodalne mają wiele lokalnych minimów i maksimów, przez co klasyczne metody mogą utknąć w lokalnym optimum. W takich

przypadkach stosuje się techniki eksploracyjne: algorytmy rojowe, symulowane wyżarzanie [69] czy strategie oparte na mutacji i selekcji [57].

Intuicyjnie multimodalność można porównać do górzystego krajobrazu z wieloma dolinami o różnej głębokości. Agent (rozwiązanie) stojący w jednej dolinie nie widzi, że za grzbietem istnieje dolina głębsza – jest to problem *optimum lokalnego*. Algorytm musi zatem dysponować mechanizmem „ucieczki" z bieżącej doliny i zbadania odleglejszych regionów przestrzeni, zanim zawęzi poszukiwanie do najbardziej obiecującego obszaru.

Wypukłość/Wklęsłość: W teorii optymalizacji istotne znaczenie ma również wypukłość funkcji [20]. Wypukłość gwarantuje, że każde lokalne minimum jest zarazem globalnym – innymi słowy, nie ma „pułapek" w postaci gorszych dolin. Dla funkcji wypukłych wystarczają klasyczne metody gradientowe. Problem zaczyna się, gdy funkcja *nie* jest wypukła – wtedy algorytm może utknąć w optimum lokalnym i konieczne staje się zastosowanie metaheurystyk (takich jak ABO, będący przedmiotem niniejszej pracy). Formalnie funkcja $f : S \to \mathbb{R}$ jest wypukła, jeśli dla dowolnych $x, y \in S$ oraz dowolnego $\lambda \in [0,1]$ spełniona jest nierówność:

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y) \tag{1.1}$$

W praktyce oznacza to, że każda lokalna optymalizacja prowadzi do rozwiązania globalnego. Wypukłość gwarantuje istnienie jednego optymalnego rozwiązania i upraszcza wyszukiwanie. Problemy wypukłe są dobrze zbadane i można je rozwiązywać metodami analitycznymi. Problemy niewypukłe wymagają metod eksploracyjnych, a znalezienie rozwiązania może być czasochłonne i obarczone ryzykiem utknięcia w lokalnym optimum.

1.1.2 Funkcje testowe w optymalizacji

Do oceny skuteczności algorytmów optymalizacyjnych stosuje się zbiory funkcji testowych (ang. *benchmark functions*) [61], reprezentujących zróżnicowane trudności obliczeniowe. Funkcje te dzieli się na kilka kategorii w zależności od ich charakteru.

Funkcje **unimodalne**, takie jak Sphere czy Schwefel-2.22, posiadają jedno globalne optimum i służą przede wszystkim do oceny szybkości zbieżności algorytmu. Funkcje **multimodalne** – np. Rastrigin [91], Ackley [1], Griewank [51] czy Schwefel [96] – zawierają liczne lokalne ekstrema i pozwalają sprawdzić zdolność metody do unikania pułapek lokalnych optimów. Odrębną grupę stanowią funkcje **kompozytowe**, łączące cechy kilku problemów w jednym, co umożliwia testowanie odporności algorytmu na złożone struktury przestrzeni rozwiązań.

W literaturze coraz częściej stosuje się również zestawy testowe CEC (*Congress on Evolutionary Computation*), które łączą różne typy trudności i stanowią uznany standard porównawczy. W niniejszej pracy wykorzystano 33 funkcje benchmarkowe

z biblioteki `opfunu` [107], reprezentujące unimodalne, multimodalne i kompozytowe problemy optymalizacyjne. Wykorzystanie standaryzowanej biblioteki zapewnia powtarzalność wyników i eliminuje ryzyko błędów implementacyjnych w funkcjach testowych.

Szczegółowy opis funkcji testowych, ich definicje matematyczne oraz wizualizacje przedstawiono w Rozdziale 5.

1.1.3 Optymalizacja w nauce i przemyśle

Poza funkcjami testowymi algorytmy optymalizacyjne znajdują praktyczne zastosowanie w różnych dziedzinach nauki i przemysłu.

Optymalizacja konstrukcyjna – stosowana w projektowaniu obiektów takich jak mosty, budynki czy elementy pojazdów. Celem jest zazwyczaj minimalizacja masy lub kosztów przy jednoczesnym spełnieniu wymagań dotyczących wytrzymałości, sztywności i stabilności. Proces ten często wymaga kosztownych obliczeniowo symulacji, takich jak analiza metodą elementów skończonych.

Dostrajanie parametrów w systemach sterowania — algorytmy optymalizacyjne znajdują zastosowanie w regulacji systemów dynamicznych, np. przy doborze parametrów regulatorów PID. Celem jest osiągnięcie stabilnej, szybkiej i precyzyjnej odpowiedzi układu. Wiele współczesnych metod przewyższa tradycyjne techniki strojenia, zarówno pod względem dokładności, jak i skuteczności.

Optymalizacja portfela inwestycyjnego – stosowana w celu maksymalizacji oczekiwanego zwrotu przy jednoczesnej minimalizacji ryzyka. Modele optymalizacyjne pozwalają na sprawne zarządzanie aktywami przy uwzględnieniu ograniczeń inwestycyjnych.

Zarządzanie ryzykiem – obejmuje optymalizację strategii redukcji ryzyka. Algorytmy optymalizacyjne pozwalają na lepsze modelowanie ryzyka i skuteczniejsze strategie zabezpieczeń.

Planowanie tras transportowych – obejmuje klasyczne problemy, takie jak problem komiwojażera (TSP) oraz problem trasowania pojazdów (VRP). Algorytmy takie jak optymalizacja mrówkowa (ACO) [34] pozwalają na minimalizację kosztów transportu i czasu dostaw przy jednoczesnym spełnieniu ograniczeń, np. pojemności pojazdów czy okien czasowych dostaw.

Zarządzanie flotą – obejmuje optymalizację harmonogramów i przydziału pojazdów w celu maksymalizacji wykorzystania zasobów i minimalizacji kosztów operacyjnych.

Optymalizacja łańcucha dostaw – algorytmy optymalizacyjne są stosowane do zarządzania przepływem towarów i informacji w sieciach dostaw. Optymalizacja poziomów zapasów, tras transportowych i harmonogramów produkcji pozwala na redukcję kosztów operacyjnych oraz poprawę jakości usług.

1.1.4 Wprowadzenie do algorytmów rojowych

Każdy z opisanych powyżej problemów raczej można określić mianem problemu o dużej złożoności. Jedną z popularnych metod ich rozwiązania jest wykorzystanie właśnie algorytmów rojowych. Ich popularność wynika z elastyczności i skuteczności w rozwiązywaniu problemów o dużej złożoności. Algorytmy te, należące do dziedziny inteligencji roju (ang. *swarm intelligence*) [18, 68], opierają się na obserwacji zachowań zbiorowych w grupach organizmów – mrówek, pszczół, ptaków i innych występujących w naturze zjawisk emergencji.

Idea algorytmów rojowych polega na wykorzystaniu prostych interakcji między agentami (cząstkami roju) do osiągnięcia złożonego zachowania na poziomie globalnym. Każdy agent działa niezależnie, kierując się kilkoma prostymi zasadami. Na przykład w algorytmie mrówkowym (Ant Colony Optimization, ACO) sztuczne mrówki poruszają się po przestrzeni rozwiązań, pozostawiając ślady feromonowe. Intensywność feromonów informuje inne mrówki o opłacalności danego rozwiązania. Z czasem rój znajduje optymalną trasę przez wzmacnianie najlepszych ścieżek.

Algorytmy rojowe charakteryzują się następującymi cechami:

- **Samoorganizacja:** Pożądane zachowanie globalne wyłania się spontanicznie z lokalnych interakcji agentów.
- **Skalowalność i stabilność:** Algorytmy rojowe dobrze radzą sobie z dużymi i złożonymi problemami, a ich wydajność jest relatywnie stabilna nawet w przypadku awarii niektórych agentów.
- **Elastyczność:** Mogą być stosowane do rozwiązywania różnorodnych problemów optymalizacyjnych, od problemów ciągłych po kombinatoryczne, w różnych dziedzinach, takich jak opisane powyżej, bez konieczności redefiniowania kodu.

Do najpopularniejszych algorytmów rojowych należą ACO [33], PSO [67] oraz algorytm sztucznej kolonii pszczół (Artificial Bee Colony, ABC) [64]. Każdy z nich czerpie inspirację z innego zjawiska naturalnego i ma własne cechy i parametry.

Kluczowym wyzwaniem projektowym każdego algorytmu rojowego jest utrzymanie równowagi między *eksploracją* a *eksploatacją* [25]. Eksploracja oznacza zdolność algorytmu do przeszukania całej przestrzeni rozwiązań – poszukiwanie nieznanych, potencjalnie obiecujących regionów. Eksploatacja to zawężanie poszukiwań do otoczenia najlepszego dotychczasowego rozwiązania i jego udoskonalanie.

Zbyt intensywna eksploracja sprawia, że algorytm nigdy nie udoskonala znalezionych rozwiązań; zbyt intensywna eksploatacja prowadzi do utknięcia w optimum lokalnym. Sposób, w jaki algorytm balansuje te dwa przeciwstawne cele, jest jednym z głównych kryteriów oceny jego jakości i będzie przewijał się przez całą niniejszą pracę.

Aby lepiej opisać problemy optymalizacyjne i charakteryzować przestrzenie rozwiązań w kolejnych sekcjach rozdziału przedstawiono podstawowe terminy i pojęcia potrzebne do analizy algorytmów rojowych.

1.1.5 Terminologia i pojęcia podstawowe

Jak już wcześniej wspomniano, każdy problem optymalizacyjny opiera się na przestrzeni rozwiązań i funkcji celu. Ich właściwości wpływają na skuteczność algorytmów poszukujących optymalnego rozwiązania.

Przestrzeń rozwiązań

Przestrzeń rozwiązań S obejmuje wszystkie możliwe rozwiązania problemu. Może mieć różne topologie, co wpływa na strategię wyszukiwania. Formalnie traktuje się ją jako przestrzeń metryczną, czyli zbiór S z funkcją odległości $d : S \times S \to \mathbb{R}$. Aksjomatyczną definicję przestrzeni metrycznej wprowadził Maurice Fréchet w swojej rozprawie doktorskiej z 1906 roku [41]; współczesne sformułowanie czterech zasad można znaleźć m.in. w [85]:

- **Zasada tożsamości:** [41] $d(x, y) = 0$ wtedy i tylko wtedy, gdy $x = y$.
- **Zasada symetrii:** [41] $d(x, y) = d(y, x)$.
- **Zasada nierówności trójkąta:** [41] $d(x, z) \leq d(x, y) + d(y, z)$.
- **Zasada nieujemności:** [41] $d(x, y) \geq 0$.

Istotna jest też spójność przestrzeni rozwiązań tzn. stopień, w jakim rozwiązania są ze sobą powiązane. Jeśli istnieje ścieżka między dowolnymi dwoma punktami, przestrzeń jest spójna.

Wymiarowość przestrzeni D ma również duże znaczenie. Im więcej zmiennych opisuje rozwiązanie, tym trudniejsza eksploracja. To zjawisko nazywane jest przekleństwem wymiarowości.

Przekleństwo wymiarowości

Wzrost liczby wymiarów powoduje lawinowe zwiększanie objętości przestrzeni i gwałtowny wzrost możliwych kombinacji parametrów. Nawet pobieżne przeszukanie całej przestrzeni staje się obliczeniowo niewykonalne.

Jeśli liczba wymiarów rośnie, liczba możliwych punktów rośnie wykładniczo. Nawet przy zaawansowanych metodach próbkowania eksploracja szybko staje się

niewystarczająca a czasem nawet niemożliwa obliczeniowo. W wielowymiarowych przestrzeniach większość punktów jest od siebie daleko, więc duże zbiory danych nie dostarczają wystarczających informacji o funkcji celu. Dla ilustracji: jeśli w przestrzeni jednowymiarowej (1D) sprawdzonych zostanie 100 równomiernie rozłożonych punktów, to w przestrzeni 10-wymiarowej należałoby sprawdzić $100^{10} = 10^{20}$ punktów – ponad dziesięciokrotnie więcej niż szacowana liczba ziaren piasku na wszystkich plażach Ziemi (ok. $7{,}5 \times 10^{18}$) [16]. Dlatego metody przeszukujące przestrzeń systematycznie (np. przeszukiwanie siatkowe) są skazane na porażkę w przestrzeniach wielowymiarowych.

Algorytmy rojowe też nie są wolne od tego problemu. Jeśli przestrzeń jest zbyt rozległa, rój może się rozproszyć, co osłabia konwergencję. Znalezienie równowagi między eksploracją a eksploatacją staje się trudniejsze niż w problemach o niższej wymiarowości.

Lokalne i globalne optima

Jak już wspomniano wcześniej każdy problem optymalizacyjny zawiera punkty, które wydają się najlepszymi rozwiązaniami w danym otoczeniu – są to optima lokalne.

Lokalne optimum x^* to rozwiązanie, dla którego istnieje otoczenie $N(x^*)$, w którym $f(x^*) \leq f(x)$ dla wszystkich $x \in N(x^*)$. Jest to punkt, w którym dalsza poprawa w bliskim sąsiedztwie jest niemożliwa. Nie oznacza to jednak, że jest to najlepsze rozwiązanie w całej przestrzeni.

Tylko globalne optimum x^* spełnia warunek $f(x^*) \leq f(x)$ dla wszystkich $x \in S$. Odnalezienie globalnego optimum jest często utrudnione przez obecność wielu lokalnych ekstremów, które mogą zatrzymać algorytm w pułapce przed osiągnięciem globalnego optimum.

Twierdzenie o braku jednego uniwersalnego rozwiązania

Twierdzenie *No Free Lunch* (NFL), sformułowane przez Wolperta i Macready'ego [119], stanowi fundamentalne ograniczenie teorii optymalizacji. Mówi ono, że uśredniając po **wszystkich** możliwych funkcjach celu, żaden algorytm optymalizacyjny nie jest lepszy od żadnego innego – w tym od losowego przeszukiwania. Formalnie, dla dowolnych dwóch algorytmów A_1 i A_2:

$$\sum_{f} P(d_m \mid f, m, A_1) = \sum_{f} P(d_m \mid f, m, A_2) \tag{1.2}$$

gdzie d_m oznacza ciąg obserwacji uzyskanych po m ewaluacjach funkcji f. Innymi słowy, jeśli algorytm A_1 przewyższa A_2 na pewnej klasie problemów, to istnieje inna klasa, na której relacja ta się odwraca.

Konsekwencja praktyczna jest jednoznaczna: nie istnieje uniwersalnie najlepszy algorytm optymalizacyjny dla wszystkich problemów. Każda metaheurystyka – w tym algorytmy rojowe – czerpie swoją przewagę z założeń (jawnych lub

ukrytych) o strukturze rozwiązywanego problemu. Ma to bezpośrednie znaczenie dla interpretacji wyników porównawczych prezentowanych w niniejszej pracy: algorytm, który dominowałby we wszystkich rankingach niezależnie od klasy funkcji testowych, wskazywałby nie na rzeczywistą uniwersalność, lecz na niewystarczającą różnorodność zbioru benchmarkowego lub błąd metodologiczny w procedurze ewaluacji.

Ograniczenia i warunki brzegowe

Poza podstawowymi zasadami które omówiono wcześniej większość rzeczywistych problemów optymalizacyjnych jest obciążona ograniczeniami, które definiują dopuszczalne rozwiązania.

1. **Ograniczenia równości** mają postać $g_i(x) = 0$, gdzie $g_i(x)$ określa warunek, jaki musi spełnić rozwiązanie.

2. **Ograniczenia nierówności** występują w postaci $h_j(x) \leq 0$ lub $h_j(x) \geq 0$.

3. **Ograniczenia dynamiczne** zmieniają się w czasie – są typowe dla problemów, w których warunki otoczenia ewoluują.

Zarządzanie granicami w algorytmach optymalizacyjnych zapewnia, że generowane rozwiązania pozostają w obrębie przestrzeni wyszukiwania. Istnieje wiele podejść do rozwiązania tego problemu. Najczęściej stosowaną metodą jest rzutowanie niepoprawnych rozwiązań na granicę obszaru przeszukiwań. Alternatywnie można wykorzystać operatory ruchu, które już na etapie generowania nowych punktów respektują narzucone ograniczenia.

Inną strategią jest penalizacja naruszeń. W takim podejściu do funkcji celu dodaje się dodatkowy składnik kary – jego wartość zależy od skali naruszenia ograniczeń. W efekcie zmodyfikowana funkcja celu przyjmuje postać:

$$f'(x) = f(x) + P(x) \tag{1.3}$$

gdzie $P(x)$ to funkcja kary, często oparta na ważonej sumie naruszeń.

Nie zawsze jednak kary są najlepszym rozwiązaniem. Mechanizmy naprawcze, zamiast karać, próbują modyfikować niewykonalne rozwiązania tak, aby stały się dopuszczalne. Może to obejmować heurystyki specyficzne dla danego problemu lub metody optymalizacyjne mające na celu znalezienie najbliższego wykonalnego punktu.

1.2 MOTYWACJA I CEL PRACY

1.2.1 Inspiracja starożytnymi strategiami

We wstępie pracy omówiono główne zagadnienie, czym jest optymalizacja i algorytmy rojowe. Nie jest to jednak istotą pracy. Główne pytanie, na które praca stara się odpowiedzieć, brzmi: czy starożytne strategie wojenne mogą inspirować algorytmy optymalizacyjne?

Wydaje się, że tak.

Podejmowanie decyzji w warunkach wojny jest złożone i wymaga dostosowywania się do zmiennych okoliczności – podobnie jak optymalizacja. Analogia między tymi dziedzinami pozwala inaczej spojrzeć na projektowanie algorytmów. Wiele elementów strategii wojskowej ma odpowiedniki w problemach optymalizacyjnych:

- Eksploracja i eksploatacja – działania wojenne wymagają równowagi między rozpoznaniem nieznanego terytorium a wykorzystaniem zdobytych przewag. Podobnie algorytmy balansują między eksploracją nowych rozwiązań a eksploatacją znanych kierunków.

- Koordynacja jednostek – synchronizacja ruchów wojsk jest niezbędna dla operacji. Algorytmy rojowe działają podobnie: agenci podejmują decyzje na podstawie informacji od sąsiadów.

- Zbieranie informacji – wywiad wojskowy dostarcza danych o wrogu i terenie. W optymalizacji analogicznie zbierane są informacje o funkcji celu, które kierują wyszukiwaniem.

- Podejmowanie decyzji w warunkach niepewności – dowódca wojskowy często musi działać na podstawie niepełnych danych. Podobnie algorytmy radzą sobie z zamglonymi funkcjami celu przy ograniczonej informacji.

- Adaptacyjne mechanizmy – armie dostosowują się do zmieniających się warunków przez zmianę taktyki. W optymalizacji często zmieniane są parametry w zależności od przebiegu wyszukiwania.

- Strategiczne pozycjonowanie – przewaga nad przeciwnikiem wiąże się z zajęciem ważnych pozycji. W optymalizacji odpowiada to znajdowaniu obiecujących regionów przestrzeni rozwiązań.

Przykłady koncepcji strategicznych:

Podobieństwa do algorytmów rojowych widać też w opisach konkretnych koncepcji strategicznych:

Sztuka wojny Sun Tzu [110] formułuje zasady takie jak znajomość wroga oraz wybór właściwego czasu i miejsca do ataku. Zasady te można przełożyć na koncepcje optymalizacji. „Znajomość wroga" odpowiada zrozumieniu cech funkcji celu. „Wybór właściwego czasu i miejsca" odnosi się do strategii eksploracji i eksploatacji.

Taktyka rzymskiego legionu [50] opierała się na dyscyplinie, organizacji i adaptacji. Zdolność legionu do manewrowania w różnych formacjach i reagowania na zmieniające się warunki jest analogiczna do elastyczności algorytmów rojowych.

Dynamika greckiej falangi [24] opierała się na zbiorowej sile i spójności ściśle upakowanej piechoty. Nasuwa to analogię do współpracy między agentami w algorytmach rojowych. Siła falangi wynikała z jej zjednoczonego ruchu, podobnie jak zbiorowa inteligencja roju wyłania się z interakcji agentów.

Strategie jazdy mongolskiej [66] kładły zaś nacisk na szybkość, mobilność i zdecentralizowane dowodzenie. Stosowanie przez Mongołów pozorowanych odwrotów i taktyk okrążania może inspirować nowe strategie wyszukiwania w optymalizacji, takie jak celowe odchodzenie od obiecujących regionów w celu zbadania innych obszarów przed powrotem z bardziej świadomym podejściem.

Aby jednak przygotować algorytm rojowy należy zacząć od modelowania strategii bitewnych. Dobrym przykładem jest manewr oskrzydlający który można modelować jako skoordynowany ruch grup cząstek, z których jedna angażuje „wroga" w postaci aktualnie znanego najlepszego wyniku, podczas gdy druga przemieszcza się, aby zaatakować z korzystniejszej pozycji potencjalnie odkrywając lepszy punkt. Kolejnym krokiem byłoby zastosowanie podejścia wielostrategicznego, w których algorytm dynamicznie przełącza się między różnymi strategiami inspirowanymi wojskiem w oparciu o bieżący stan wyszukiwania. Wybór strategii pozwala algorytmowi dostosować się do różnych problemów i faz wyszukiwania.

Adaptacyjne dostrajanie również może być używane do precyzyjnego dostrajania parametrów inspirowanych wojskiem. Na przykład „agresywność" manewru oskrzydlającego można kontrolować za pomocą parametru, który jest dynamicznie dostosowywany na podstawie wskaźnika powodzenia manewru lub zależnie od typu agenta.

Systematyczna eksploracja starożytnej strategii wojskowej może dostarczyć nowych spostrzeżeń i narzędzi do rozwiązywania trudnych problemów optymalizacji. Podejście interdyscyplinarne, łączące analizę historyczną z rygorem matematycznym i eksperymentami obliczeniowymi, potencjalnie otwiera nowy obszar badań.

1.2.2 Uzasadnienie proponowanego podejścia

Aby dobrze uzasadnić nowe podejście należy zauważyć że pomimo sukcesów w wielu zastosowaniach, algorytmy rojowe mają wady. Można je podzielić na ograniczenia teoretyczne i wydajnościowe.

Ograniczenia teoretyczne

Przedwczesna konwergencja jest poważnym problemem, zwłaszcza w optymalizacji multimodalnej [37]. Algorytmy rojowe, szczególnie te polegające na pojedynczym globalnym najlepszym rozwiązaniu, mogą przedwcześnie zbiegać się do lokalnego optimum. Często wynika to z utraty różnorodności w roju, gdy wszystkie cząstki skupiają się wokół jednego suboptymalnego rozwiązania.

Wrażliwość parametrów to kolejny problem. Wydajność algorytmów rojowych zależy od wyboru parametrów [37, 34] (np. szybkości parowania feromonów w ACO). Znalezienie optymalnych ustawień wymaga eksperymentów i dostrajania dla konkretnego problemu. Brak możliwości zmiany podejścia bez modyfikacji parametrów utrudnia generalizację rozwiązania.

Przekleństwo wymiarowości wpływa na algorytmy rojowe tak samo jak na inne techniki optymalizacji. Ze wzrostem wymiarowości objętość przestrzeni rośnie wykładniczo, co utrudnia rojowi skuteczną eksplorację i prowadzi do wolniejszej konwergencji lub wręcz je uniemożliwia.

Równowaga eksploracji i eksploatacji jest wyzwaniem we wszystkich algorytmach optymalizacji. Algorytmy rojowe muszą balansować między eksploracją nowych obszarów a udoskonalaniem znalezionych rozwiązań. Brak równowagi prowadzi do przedwczesnej konwergencji (zbyt duża eksploatacja) lub nieefektywnego wyszukiwania (zbyt duża eksploracja).

Ograniczenia wydajnościowe

Prędkość konwergencji jest problemem, zwłaszcza w aplikacjach czasu rzeczywistego. Algorytmy rojowe mogą wymagać dużej liczby iteracji, aby znaleźć dobre rozwiązania w wielowymiarowych problemach.

Problemy ze skalowaniem pojawiają się przy bardzo dużych problemach z wieloma wymiarami. Utrzymywanie dużej populacji i wykonywanie wielu iteracji może stać się obliczeniowo niewykonalne.

1.2.3 Potencjalne korzyści z wdrożenia strategii bitewnej

Strategie wojskowe wypracowane przez stulecia mogą ulepszyć wyszukiwanie, podejmowanie decyzji i odporność algorytmów rojowych.

Usprawnione mechanizmy wyszukiwania

Strategie wojskowe często obejmują eksplorację wielokierunkową. Manewr oskrzydlający polega na atakowaniu z wielu kierunków jednocześnie. Można to przełożyć na optymalizację przez użycie wielu rojów do eksplorowania różnych regionów przestrzeni rozwiązań.

Kontrola formacji, widoczna w legionach rzymskich, może inspirować algorytmy dynamicznie dostosowujące topologię komunikacji i wzorce ruchu. Może to poprawić równowagę między eksploracją a eksploatacją i zapobiec przedwczesnej konwergencji.

Poprawione podejmowanie decyzji

Ocena pozycji strategicznej obejmuje ocenę terenu i siły wroga. W optymalizacji można włączyć dodatkowe informacje poza wartością funkcji celu – gradient, gęstość rozwiązań czy historię poszukiwań. Dowódcy wykorzystują wywiad o wrogu i terenie do podejmowania decyzji. Podobnie w optymalizacji można gromadzić i wykorzystywać informacje o strukturze funkcji celu, np. gradient czy rozkład wyników.

Zwiększona stabilność systemu

Strategie wojskowe są często projektowane tak, aby były odporne na adaptację środowiskową, działania wroga i nieprzewidziane okoliczności. Tę cechę również można przełożyć na algorytmy optymalizacji.

Poprawki tolerancji błędów można osiągnąć poprzez włączenie nadmiarowości i zdecentralizowanej kontroli, zainspirowanej organizacją jednostek wojskowych. Jeśli jeden podrój zawiedzie, inne powinny kontynuować poszukiwania.

Dynamiczna reorganizacja jest kolejnym elementem adaptacji wojskowej. Armia może zmienić swoją formację lub strukturę dowodzenia w odpowiedzi na zmieniające się warunki na polu bitwy. Można to wdrożyć w optymalizacji, umożliwiając rojowi dynamiczne dostosowywanie topologii komunikacji, ustawień parametrów.

1.2.4 Cele badawcze i oczekiwane wyniki

W tej sekcji omówione zostaną cele badawcze i przewidywane wyniki rozprawy, która koncentruje się na opisanym powyżej nowym podejściu do optymalizacji inspirowanym strategią wojskową. To podejście, nazwane „Strategic Swarm Intelligence" (SSI), ma rozwiązać ograniczenia istniejących algorytmów rojowych i

zapewnić bardziej wydajne podejście do problemów optymalizacji.
Uwaga terminologiczna: W niniejszej pracy zaproponowany termin **SSI** (Strategic Swarm Intelligence) odnosi się do ogólnych ram koncepcyjnych, natomiast **ABO** (Ancient Battlefield Optimizer) i **CommanderABO** to konkretne algorytmy implementujące to podejście. W kolejnych rozdziałach (4-7) używane są nazwy algorytmów (ABO, CommanderABO), podczas gdy w tym rozdziale wprowadzającym stosowana jest nazwa SSI głównie ze względów praktycznych - podczas opisywania założeń koncepcyjnych algorytm ABO oraz jego modyfikowany odpowiednik CommanderABO jeszcze nie istniały.

Główny cel

Głównym celem jest opracowanie, wdrożenie i ocena podejścia SSI. Obejmuje to zdefiniowanie podstaw teoretycznych, zaprojektowanie architektury i opisanie metodologii integracji koncepcji strategii wojskowej oraz ocena czy wdrożenie strategii starożytnego wojska wpływają pozytywnie na wyniki osiągane przez algorytmy rojowe.

Podstawowe elementy podejścia

Podstawy teoretyczne **SSI** opierają się na syntezie zasad inteligencji roju, teorii optymalizacji i koncepcji ze strategii wojskowej (wymienionych w poprzednich sekcjach).

Podstawowe ramy **SSI** sformalizują mapowanie między koncepcjami wojskowymi a komponentami algorytmicznymi. Obejmuje to zdefiniowanie:

1. **Reprezentacja stanu**: W jaki sposób rozwiązania (pozycje w przestrzeni poszukiwań) są reprezentowane.
2. **Operatory ruchu**: Równania, które regulują ruch agentów (cząstek) w przestrzeni poszukiwań. Zostaną one przygotowane tak, aby uwzględnić manewry strategiczne (np. flankowanie, pozorowane odwroty, skoncentrowane ataki).
3. **Topologia komunikacji**: Definiowanie sposobu, w jaki agenci wchodzą w interakcje i udostępniają informacje. SSI będzie badać dynamiczną i hierarchiczną topologię, wzorowaną na wojskowej strukturze dowodzenia.
4. **Mechanizm wyboru strategii**: Model dowódczy zbudowany do wybierania i dostosowywania różnych strategii wyszukiwania podczas procesu optymalizacji.

Architektura **SSI** opiera się na modułowości, adaptacyjności i kontroli hierarchicznej. Architektura pojedynczej optymalizacji będzie zatem składać się z wielu oddziałujących na siebie rojów lub podrojów, z których każdy potencjalnie będzie stosował strategie i różne parametry. Agent lub moduł „dowódcy” może nadzorować cały proces wyszukiwania, przydzielając zasoby i koordynując działania podrojów lecz nie wpływa bezpośrednio na działanie poszczególnych agentów a tylko na to jak podroje współpracują ze sobą i co priorytetyzują.

SSI w początkowej fazie projektu wprowadza sześć typów jednostek bojowych, z których każdy implementuje odrębną metodę przeszukiwania przestrzeni:

1. **Piechota ciężka** (*Heavy Infantry*): strategia przeszukiwania współrzędnościowego (eksploatacja lokalna), naśladująca metodyczne postępy hoplitów.
2. **Piechota lekka** (*Light Infantry*): strategia ewolucji różnicowej, inspirowana zwinnymi welitami i peltastami.
3. **Łucznicy** (*Archers*): strategia wielopunktowego próbkowania zdalnego – odpowiednik obszarowych salw łuczników.
4. **Kawaleria** (*Cavalry*): strategia lotów Lévy'ego z pamięcią kierunkową – szarże o rozkładzie odległości z ciężkim ogonem (eksploracja globalna).
5. **Rydwany** (*Chariots*): strategia pędu Cauchy'ego – ruch liniowy z bezwładnością.
6. **Słonie bojowe** (*War Elephants*): strategia przeskoków międzydolinowych – łagodne udoskonalanie lokalne przeplatane nagłymi skokami.

Formalne definicje wszystkich sześciu operatorów ruchu wraz z parametrami podano w Rozdziale 4.

Założenie **SSI** włącza systemy sterowania adaptacyjnego na wielu poziomach. Obejmuje to parametry (np. automatyczne dostosowywanie „agresywności" manewrów flankujących), dynamiczne dostosowywanie topologii (np. zmiana sieci komunikacyjnej między agentami) i mechanizmy wyboru strategii (np. wykorzystanie uczenia się przez wzmacnianie [104] w celu wybrania najlepszej strategii dla bieżącej sytuacji).

Podejmowanie decyzji strategicznych w SSI jest realizowane poprzez połączenie kontroli hierarchicznej i adaptacji opartej na informacjach. Moduł dowódcy podejmuje decyzje wysokiego szczebla, takie jak przydzielanie formacji do różnych podrojów lub wybierając ogólne aktualne strategie wyszukiwania. Poszczególni agenci podejmują decyzje niższego szczebla w oparciu o lokalne informacje i przypisane im taktyki.

Od strony metodologicznej powyżej przedstawiono sposób przenoszenia zasad starożytnego pola bitwy do projektowania algorytmów. Oczekuje się, że przełoży się to na lepszą zbieżność i jakość rozwiązań, zwłaszcza w problemach multimodalnych i wysokowymiarowych, przy ograniczonym koszcie specjalizacji.

1.3 HIPOTEZY I PYTANIA BADAWCZE

W tej sekcji sformułowano hipotezy badawcze będące podstawą rozwoju i oceny podejścia Strategic Swarm Intelligence (SSI).

1.3.1 Teza rozprawy

Na podstawie przeprowadzonej analizy ograniczeń istniejących algorytmów rojowych oraz potencjału strategii wojskowych jako źródła inspiracji, sformułowano następującą tezę rozprawy:

> **Teza:** *Systematyczne przeniesienie starożytnych strategii bitewnych – obejmujących heterogeniczne typy jednostek, zróżnicowane taktyki manewrowe oraz hierarchiczne dowodzenie – do architektury algorytmu rojowego, uzupełnione o adaptacyjny dobór taktyk za pomocą uczenia ze wzmocnieniem, pozwala uzyskać algorytm optymalizacji konkurencyjny wobec współczesnych metaheurystyk na standardowych funkcjach benchmarkowych w szerokim zakresie wymiarowości.*

Alternatywne sformułowanie w nomenklaturze hyper-heuristics i AOS. W odpowiedzi na zasadną krytykę nurtu metaheurystyk opartych na metaforach (Sörensen [101], Aranha i in. [2], Camacho-Villalón i in. [22], szczegółowo dyskutowanej w Sekcji 2.3.7), powyższą tezę można *równoważnie* przeformułować w nomenklaturze **hyper-heuristics** oraz **Adaptive Operator Selection (AOS)**:

> ***(Teza, sformułowanie alternatywne)**: Repozytorium sześciu heterogenicznych operatorów ruchu (kombinujących Lévy flights, coordinate descent, differential evolution, Cauchy momentum, multi-point sampling i basin hopping), zorganizowane w jednolitej populacyjnej architekturze z tablicowym meta-kontrolerem Q-learning operującym na zdyskretyzowanej, pięciowymiarowej reprezentacji stanu krajobrazu fitness, stanowi konkurencyjną instancję paradygmatu hyper-heuristics z adaptacyjną selekcją operatora (AOS), zwalidowaną empirycznie na 165 konfiguracjach funkcja×wymiar.*

Oba sformułowania opisują *ten sam* koncept – Algorytm ABO – na różnych poziomach abstrakcji: pierwsza wersja akcentuje *dydaktyczny i mnemoniczny* wymiar projektu (organizacja konceptualna inspirowana taktyką militarną), druga wersja akcentuje *formalny i algorytmiczny* wymiar (mapowanie na ustaloną nomenklaturę RL i hyper-heuristics z referencjami do Burke i in. [21], Fialho i in. [39], DaCosta i in. [26]). Dekompozycja wkładu (Sekcja 6.5.2) pokazuje, że zdecydowana większość obserwowanej przewagi ABO pochodzi z architektury heterogenicznej populacji : sama architektura (wariant ABO-ManualPhases) wyprzedza najlepszy algorytm spoza rodziny o $\Delta R = 3{,}81$ rangi, natomiast adaptacyjna selekcja operatora przez Q-learning wnosi wkład komplementarny, zależny od kryterium (neutralny dla mediany, dodatni dla najlepszego wyniku z przebiegu, ang. *best-of-run*). Empirycznie potwierdza to, że obie warstwy mają mierzalne, identyfikowalne źródła w ustalonej literaturze.

Z powyższej tezy wynikają dwa cele badawcze i odpowiadające im hipotezy:

C1: Opracowanie i ewaluacja algorytmu ABO opartego na strategiach bitewnych → Hipoteza H1 (skuteczność).

C2: Integracja uczenia ze wzmocnieniem do adaptacyjnego wyboru taktyk → Hipoteza H2 (adaptacyjność).

1.3.2 Hipoteza 1: Skuteczność algorytmów inspirowanych bitwą

Hipoteza ta zakłada, że włączenie zasad strategii wojskowej do algorytmów rojowych pozwoli osiągnąć satysfakcjonującą wydajność w rozwiązywaniu problemów optymalizacji.

H1: Algorytm **ABO** (Ancient Battlefield Optimizer) wraz z jego wariantem **CommanderABO** – stanowiące implementację ram koncepcyjnych SSI – obejmujący strategie inspirowane bitwą, takie jak flankowanie, pozorowany odwrót i skoncentrowany atak, będzie wykazywał lepszą wydajność w porównaniu ze standardowymi algorytmami optymalizacyjnymi w tym inteligencji roju i ich ustalonymi wariantami w zestawie problemów testowych dla optymalizacji wielowymiarowej.

Oczekuje się, że ABO/CommanderABO osiągnie:

1. Wyższą jakość rozwiązań (niższe wartości funkcji celu dla problemów minimalizacji).
2. Lepszą zdolność do ucieczki od lokalnych optimów.
3. Lepszy poziom generalizacji rozwiązania.

Walidacja **H1** będzie obejmować następującą metodologię eksperymentalną:

Metodologia testowania: Algorytm ABO oraz wariant CommanderABO zostaną zaimplementowane i przetestowane na zestawie problemów optymalizacji.

Dla każdego problemu i algorytmu zostanie wykonanych wiele niezależnych przebiegów.

Wybór testów porównawczych: Zestaw testów porównawczych będzie zawierał różnorodne problemy, takie jak:

1. Funkcje unimodalne (np. funkcja Sphere) do testowania podstawowej zbieżności.
2. Funkcje multimodalne (np. funkcje Rastrigina, Ackleya) do testowania zdolności do ucieczki od lokalnych optimów.
3. Funkcje wielowymiarowe do testowania skalowalności i oceny wpływu klątwy wielowymarowości na skuteczność algorytmów.

Analiza statystyczna: Testy statystyczne [32] zostaną wykorzystane do porównania wydajności ABO/CommanderABO z wydajnością standardowych algorytmów roju i innych metod optymalizacji, których wyniki zostały opublikowane w sposób umożliwiający reprodukcję oraz weryfikację implementacji.

Metryki wydajności: Zostaną użyte następujące metryki wydajności:

1. Najlepsze dopasowanie: Najlepsza średnia wartość funkcji obiektywnej znaleziona przez algorytm (fitness).
2. Współczynnik powodzenia: Procent przebiegów, które znajdują rozwiązanie w ramach określonej tolerancji globalnego optimum.

Kryteria sukcesu dla hipotezy będą obejmować wykazanie poprawy w wybranych wskaźnikach wydajności w porównaniu z algorytmami bazowymi.

Walidacja: Test Friedmana [42] na 33 funkcjach $\times$ 5 wymiarów z udziałem 46 algorytmów (42 porównawcze + 4 warianty ABO). Kryterium: pozycja w pierwszej piątce rankingu ogólnego (w świetle twierdzenia NFL nie oczekuje się dominacji na wszystkich klasach funkcji, lecz wysokiej pozycji zagregowanej na zróżnicowanym zbiorze).

1.3.3 Hipoteza 2: Adaptacyjny wybór strategii

Hipoteza ta zakłada, że automatyczny dobór taktyk za pomocą uczenia ze wzmocnieniem dorówna – *bez ręcznego harmonogramowania sekwencji taktyk i bez potrzeby jego ponawiania przy zmianie zestawu funkcji* – czołowym eksperckim harmonogramom fazowym, a jednocześnie podniesie najwyższą osiąganą jakość rozwiązań.

H2: Adaptacyjny dobór taktyk za pomocą Q-learningu (Commander AI), *bez ręcznego strojenia sekwencji taktyk*, osiągnie w kryterium mediany wynik konkurencyjny z najlepszym predefiniowanym harmonogramem fazowym, a w kryterium najlepszego rozwiązania (best-of-run) *istotnie statystycznie* przewyższy nieadaptacyjne (statyczne, Q-tabelowe) warianty doboru taktyk, podnosząc „sufit" jakości rozwiązań.

Oczekuje się, że algorytm:

1. w kryterium najlepszego rozwiązania (best-of-run) osiągnie istotnie wyższą rangę Friedmana niż statyczne warianty Q-tabelowe, zajmując czołową pozycję w całym polu porównawczym – co potwierdza podniesienie „sufitu" jakości rozwiązań przez adaptacyjną politykę;

2. w kryterium mediany osiągnie wynik konkurencyjny z strojonym harmonogramem fazowym *bez* ręcznego strojenia sekwencji taktyk, co dowodzi, że polityka uczona automatyzuje projektowanie harmonogramu bez utraty powtarzalnej jakości;

3. wykaże zdolność do **transferu i generalizacji**: polityka uczona na rozłącznym zbiorze treningowym generalizuje na zbiór benchmarkowy (test generalizacji, Sekcja 5.2.2) i adaptuje dobór taktyk online – dzięki czemu, w odróżnieniu od harmonogramu strojonego pod konkretny benchmark, zmiana zestawu funkcji nie wymaga ponownego ręcznego wyszukiwania taktyk; możliwa jest też akumulacja Q-tabeli między uruchomieniami.

Walidacja: Parowy test Wilcoxona [118] (dwustronny, $\alpha = 0{,}05$) porównujący CommanderABO z nieadaptacyjnymi wariantami ABO (ManualPhases, QTable4K, QTable10K) na 165 parach funkcja–wymiar, przeprowadzony *osobno* dla kryterium mediany i kryterium best-of-run, uzupełniony o analizę rozmiaru efektu (Cohen's r) [23].

1.4 STRUKTURA PRACY

Rozprawa składa się z siedmiu rozdziałów ułożonych tak, by przejść od podstaw teoretycznych, przez projektowanie i implementację, do weryfikacji eksperymentalnej.

Praca ma układ typowy dla badań w informatyce: *problem → teoria → projekt → implementacja → eksperyment → wnioski*. Kolejność rozdziałów odpowiada chronologii powstawania algorytmu Ancient Battlefield Optimizer – od inspiracji militarnej, przez formalizację matematyczną, po weryfikację eksperymentalną.

Zawartość kolejnych rozdziałów

Rozdział 2: Przegląd literatury omawia stan wiedzy w dziedzinie metod optymalizacji – od podejść klasycznych po współczesne. Opisano algorytmy ewolucyjne, strategie ewolucyjne, metody rojowe oraz techniki hybrydowe i adaptacyjne. Omówiono też podejścia interdyscyplinarne: algorytmy immunologiczne, systemy inspirowane zachowaniami zwierząt (algorytm nietoperzy, algorytm świetlików), metody oparte na zachowaniach społecznych, a także dotychczasowe próby adaptacji strategii wojskowych do optymalizacji. Na końcu rozdziału wskazano lukę badawczą – brak algorytmów łączących różnorodne taktyki bitewne w jednym spójnym systemie rojowym.

Rozdział 3: Teoretyczne podstawy starożytnych strategii bitewnych dostarcza analizy wybranych formacji i taktyk używanych w starożytności, tworząc most między domeną militarną a obliczeniową. Omówiono:

- *Falangę i formacje liniowe* – analizując historię od greckich początków (VIII-IV w. p.n.e.), przez modyfikacje macedońskie (sarissa, heterogenna falanga), aż po adaptacje rzymskie (manipuły, kohorty). Przedstawiono geometrię formacji, koordynację jednostek, dynamikę ruchu oraz rozkład sił.

- *Manewry oskrzydlające i obejścia* – omawiając taktyki flankowania, podwójnego obejścia, odcięcia linii zaopatrzenia oraz wykorzystania terenu. Przeanalizowano historyczne przykłady (Kannae, Leuctra, Gaugamela) i ich zasady strategiczne.

- *Taktyki koncentracji siły* – przedstawiając zasadę masy krytycznej, punktów przełamania, rezerw strategicznych oraz sekwencyjnych ataków. Omówiono podstawy matematyczne rozkładu sił i optymalizacji alokacji zasobów.

- *Strategie wywiadu i rozpoznania* – analizując rolę zwiadowców, zbierania informacji, mapowania terenu oraz adaptacji taktycznej. Przedstawiono analogie do eksploracji przestrzeni poszukiwań.

- *Zasady dowodzenia i koordynacji* – omawiając hierarchie dowodzenia, systemy sygnalizacji, autonomię podjednostek oraz elastyczność struktury dowodzenia.

Dla każdej strategii przedstawiono historię i ewolucję, podstawowe zasady działania, zalety i ograniczenia oraz – co najważniejsze – potencjał adaptacji do algorytmów optymalizacji poprzez formalne mapowanie koncepcji militarnych na elementy obliczeniowe.

Rozdział 4: Projektowanie modułowych algorytmów rojowych inspirowanych strategiami bitewnymi opisuje szczegółowo architekturę i implementację opracowanego algorytmu Ancient Battlefield Optimizer (ABO). Przedstawiono tu:

- *Architekturę modułową* opartą na autonomicznych jednostkach bitewnych różnych typów: ofensywnych (piechota lekka, kawaleria – eksploracja globalna), defensywnych (piechota ciężka – eksploatacja lokalna), adaptacyjnych (jednostki bohaterskie), wspomagających (łucznicy, rydwany) oraz przywódczych (Commander AI).

- *System operatorów optymalizacji* inspirowanych konkretnymi manewrami bitewnymi: falanga (eksploatacja lokalna z wysoką kohezją), szyk skośny (asymetryczne przeszukiwanie), przełamanie centrum (koncentracja na obiecujących regionach), oskrzydlenie (eksploracja peryferyjna), okrążenie (poszukiwanie wielokierunkowe) oraz rozpoznanie (ucieczka z optimów lokalnych).

- *Mechanizmy komunikacji i koordynacji* między jednostkami, w tym system dzielenia wiedzy, formacje przestrzenne (falanga, linia, kolumna, klin, łuk, okrąg) oraz protokoły synchronizacji.

- *System zwiadowczy* (Reconnaissance Intelligence) – siatka mapująca przestrzeń poszukiwań z poziomami zainteresowania i niebezpieczeństwa, wykorzystywana do kierowania eksploracją.

- *Hierarchiczny system dowodzenia* wykorzystujący uczenie przez wzmacnianie (Q-learning) w module Commander AI, który adaptacyjnie wybiera taktyki i formacje na podstawie bieżącego stanu optymalizacji. Opisano reprezentację stanów (kompozycja armii, wydajność, postęp), przestrzeń akcji (wybór taktyk i formacji), funkcję nagrody oraz strategię eksploracji-eksploatacji.

- *System honorów* motywujący jednostki i wpływający na ich zachowanie poprzez dynamiczną modyfikację parametrów.

Rozdział 5: Implementacja i środowisko eksperymentalne szczegółowo opisuje praktyczną realizację algorytmu ABO w języku Python. Przedstawiono:

- *Uzasadnienie wyborów technologicznych*: Python dla czytelności i ekosystemu bibliotek naukowych (NumPy dla obliczeń numerycznych, SciPy dla algorytmów optymalizacji), opfunu (100+ funkcji benchmarkowych), mealpy (algorytmy porównawcze), Matplotlib i Plotly (wizualizacja), Streamlit (interfejs).

- *Środowisko testowe* wykorzystujące ponad 100 funkcji benchmarkowych z biblioteki opfunu, sklasyfikowanych według modalności (unimodalne/wielomodalne), wymiaru oraz charakterystyk.

- *Metodykę eksperymentów*: plan testów obejmujący przestrzenie wymiarów 2D–100D (2D, 10D, 30D, 50D, 100D), wielokrotne uruchomienia dla wiarygodności statystycznej, kryteria zbieżności i warunki brzegowe.

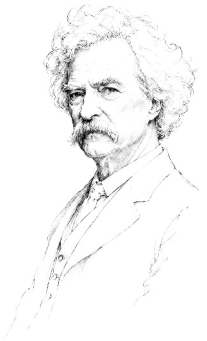
- *Metryki oceny*: dokładność rozwiązania (odległość od znanego optimum), wskaźnik sukcesu (procent przebiegów osiągających próg), czas obliczeniowy oraz wskaźniki specjalistyczne (trajektoria poszukiwań, różnorodność populacji).

- *Metody analizy statystycznej*: testy parametryczne i nieparametryczne, wielokrotne porównania z poprawką Bonferroniego oraz wykresy box-plot i krzywe konwergencji.

Rozdział 6: Wyniki eksperymentalne i dyskusja zawiera analizę wyników i weryfikację hipotez badawczych. Porównano warianty ABO z algorytmami metaheurystycznymi za pomocą rankingu Friedmana, testów post-hoc i studium porównawczego. Cztery warianty ABO zajęły 4 pierwsze pozycje ogólnego rankingu pełnego benchmarku: najlepszą medianę uzyskał ABO-ManualPhases (śr. rang 6,21), a CommanderABO uplasował się tuż za nim (6,77), osiągając zarazem najlepszą rangę w kryterium najlepszego wyniku (best-of-run) w całym polu 46 algorytmów. Omówiono też ograniczenia algorytmu.

Rozdział 7: Zakończenie podsumowuje badania i formułuje wnioski oraz w tym rozdziale również wskazano kierunki dalszych prac – m.in. rozszerzenie o kolejne strategie wojskowe, zastosowania w konkretnych problemach dziedzinowych i integrację z uczeniem głębokim.

2. STUDIUM LITERATURY



„Człowiek, który nie czyta,
nie ma żadnej przewagi nad tym,
który czytać nie umie.”
„The man who does not read has no advantage
over the man who cannot read.” (ang.)
– Mark Twain

W poprzednim rozdziale przedstawiono podstawy optymalizacji – przekleństwo wymiarowości, pułapki lokalnych minimów, ograniczenia. Przed sprawdzeniem, czy starożytne strategie bitewne mogą inspirować algorytmy optymalizacyjne, trzeba poznać istniejące metody – co potrafią, gdzie zawodzą i przede wszystkim czego w nich brakuje.

Rozdział ten zawiera przegląd literatury z dziedziny metod optymalizacyjnych. Przegląd rozpoczyna się od klasycznych metod gradientowych, po których przedstawiono algorytmy ewolucyjne, genetyczne, strategie ewolucyjne oraz symulowane wyżarzanie (Sekcja 2.1). Następnie omówiono algorytmy rojowe i metody inspirowane naturą (Sekcja 2.2), a dalej interdyscyplinarne podejścia w optymalizacji, w tym uczenie ze wzmocnieniem jako mechanizm adaptacyjnego sterowania oraz dotychczasowe próby adaptacji strategii wojskowych (Sekcja 2.3). Algorytmy te są ważne zarówno jako tło teoretyczne, jak i jako metody użyte w badaniu porównawczym. Na końcu wskazano lukę badawczą, którą ta praca ma wypełnić (Sekcja 2.4).

2.1 OD METOD KLASYCZNYCH DO EWOLUCYJNYCH

Jak omówiono w poprzednim rozdziale, klasyczne metody optymalizacji mają dobrze znane ograniczenia. Przede wszystkim obejmują one problemy obliczeniowe w przypadku wysokiej wymiarowości problemu i dużych zbiorów danych, które powodują trudności ze skalowalnością. Ograniczenia teoretyczne wynikają z założeń, jakie te metody stawiają. Jak już nadmieniono, klasyczne metody optymalizacji wymagają założeń o wypukłości (lub wklęsłości) oraz różniczkowalności funkcji celu. W rzeczywistych zastosowaniach warunki te często nie są spełnione, co może prowadzić do utknięcia algorytmu w optimach lokalnych. Te podstawowe

ograniczenia związane z wypukłością, różniczkowalnością i złożonością obliczeniową motywują badania nad nowymi paradygmatami: metaheurystykami [17, 126] i optymalizacją stochastyczną. Te podejścia, choć często pozbawione gwarancji optymalności, radzą sobie z bardziej złożonymi problemami.

2.1.1 Algorytmy gradientowe

Metody gradientowe są podstawą wielu algorytmów optymalizacji [87]. Opierają się na koncepcji pochodnej kierunkowej, która mierzy tempo zmian funkcji wzdłuż określonego kierunku. Gradient, wektor wskazujący kierunek najbardziej stromego wznoszenia, jest wartością ujemną kierunku najbardziej stromego spadku. Podstawową zasadą gradientu jest iteracyjna aktualizacja parametrów funkcji w kierunku ujemnego gradientu, minimalizując w ten sposób funkcję.

Podstawowy algorytm gradientu zstępującego zapisuje się jako:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t), \tag{2.1}$$

gdzie $\mathbf{x}_t$ jest wektorem parametrów w iteracji t, η jest współczynnikiem uczenia (rozmiarem kroku), a $\nabla f(\mathbf{x}_t)$ jest gradientem funkcji celu f w $\mathbf{x}_t$. Wybór współczynnika uczenia ma istotne znaczenie. Zbyt mały współczynnik prowadzi do powolnej zbieżności, zbyt duży powoduje oscylacje lub dywergencję. Strategie wyboru rozmiaru kroku obejmują stałe współczynniki uczenia, metody wyszukiwania liniowego i adaptacyjne metody współczynnika uczenia.

Właściwości zbieżności gradientu zależą od charakterystyki funkcji celu [20]. W przypadku funkcji wypukłych gradient zstępujący z odpowiednią szybkością uczenia gwarantuje zbieżność do minimum globalnego. W przypadku funkcji niewypukłych może zagwarantować zbieżność jedynie do minimum lokalnego.

Wymienione ograniczenia skłoniły badaczy do poszukiwania podejść niewymagających informacji o gradiencie. Algorytmy ewolucyjne [8, 36], choć nie gwarantują optymalności, radzą sobie z funkcjami niewypukłymi, nieróżniczkowalnymi i wielomodalnymi.

2.1.2 Klasyczne metaheurystyki

Algorytmy ewolucyjne

Algorytmy genetyczne (GA) [57, 49] to klasa algorytmów ewolucyjnych inspirowanych procesem doboru naturalnego. Działają na populacji rozwiązań kandydackich, iteracyjnie stosując operatory genetyczne w celu ewolucji populacji w kierunku lepszych rozwiązań. Podstawowymi elementami składowymi GA są reprezentacja chromosomu, funkcja sprawności, mechanizmy selekcji i operatory genetyczne.

Reprezentacja chromosomu koduje rozwiązanie kandydackie jako ciąg (często binarny, ale możliwe są inne kodowania), analogicznie do chromosomu biologicznego. Funkcja sprawności ocenia jakość rozwiązania, analogicznie do sprawności

organizmu w jego środowisku. Funkcja ta musi być starannie zaprojektowana, aby odzwierciedlać pożądany cel optymalizacji. Mechanizmy selekcji określają, które osobniki w populacji są wybierane do reprodukcji, na podstawie ich sprawności. Do powszechnych metod selekcji należą selekcja koła ruletki [9] (gdzie prawdopodobieństwo selekcji jest proporcjonalne do sprawności), selekcja turniejowa (gdzie jednostki rywalizują w małych grupach) i selekcja oparta na rankingu.

Operatory genetyczne wprowadzają zmienność do populacji. Podstawowymi operatorami są krzyżowanie i mutacja. Krzyżowanie łączy materiał genetyczny z dwóch chromosomów rodzicielskich w celu uzyskania potomstwa, naśladując rozmnażanie płciowe. Powszechnymi operatorami krzyżowania są krzyżowanie jednopunktowe, dwupunktowe i jednorodne [38]. Mutacja wprowadza losowe zmiany do chromosomu, analogicznie do mutacji biologicznych. Tempa mutacji są zazwyczaj utrzymywane na niskim poziomie, aby uniknąć zakłócania dobrych rozwiązań, ale są niezbędne do utrzymania różnorodności i eksploracji nowych regionów przestrzeni poszukiwań.

Dynamika populacji w GA jest uwarunkowana przez kilka czynników [29, 117]. Presja selekcyjna, determinowana przez mechanizm selekcji, kieruje populację w stronę wyższej sprawności. Wysoka presja selekcyjna może prowadzić do przedwczesnej konwergencji, w której populacja traci różnorodność i zostaje uwięziona w lokalnym optimum. Dryf genetyczny, zjawisko stochastyczne wynikające ze skończonej wielkości populacji, może również prowadzić do utraty różnorodności, nawet przy braku presji selekcyjnej. Wzajemne oddziaływanie presji selekcyjnej i dryfu genetycznego determinuje wydajność GA, ale udowodnienie konwergencji dla ogólnych GA pozostaje trudnym problemem.

Strategie ewolucyjne

Strategie ewolucyjne (ES) [92, 97, 15] to kolejna klasa algorytmów ewolucyjnych, zaprojektowana głównie do ciągłej optymalizacji parametrów. W przeciwieństwie do algorytmów genetycznych, które często wykorzystują kodowanie binarne, ES bezpośrednio działają na wektorach o wartościach rzeczywistych.

Historycznie ES zaczynały się od prostych strategii, takich jak (1+1)-ES, która polega na tym, że jeden rodzic produkuje jedno potomstwo poprzez mutację, a lepszy z nich jest wybierany. (μ,λ)-ES wykorzystuje rodziców μ do generowania potomstwa λ, a najlepsze osobniki λ z potomstwa są wybierane w celu utworzenia następnego pokolenia. $(\mu + \lambda)$-ES wybiera najlepsze osobniki μ z połączonej puli rodziców i potomstwa.

Nowoczesne ES często wykorzystują samoadaptacyjną kontrolę parametrów [52]. Parametry kontrolujące operatora mutacji (np. siła mutacji) są zakodowane w osobnikach i podlegają ewolucji. Pozwala to algorytmowi dostosować strategię wyszukiwania do charakterystyki problemu.

Mutację w ES uzyskuje się zazwyczaj poprzez dodanie losowego wektora o rozkładzie normalnym do osobnika nadrzędnego. Rekombinacja, choć mniej podkreślana niż w GA, może być również stosowana, często obejmując uśrednianie lub pośrednią rekombinację parametrów nadrzędnych. Selekcja jest zazwyczaj deterministyczna, wybierając najlepsze osobniki na podstawie ich sprawności.

Programowanie genetyczne

Programowanie genetyczne (GP) [70] rozszerza zasady GA, aby rozwijać programy komputerowe lub inne wykonywalne struktury. W przeciwieństwie do GA, które działają na ciągach o stałej długości, GP zazwyczaj używa reprezentacji opartych na drzewach, gdzie każdy węzeł reprezentuje funkcję (zmienną lub stałą).

Zestaw funkcyjny definiuje zbiór funkcji, które mogą być używane w drzewach (np. operatory arytmetyczne, operatory logiczne, instrukcje warunkowe). Zestaw terminalny definiuje zbiór zmiennych i stałych, które mogą być używane jako dane wejściowe lub liście drzew.

Symulowane wyżarzanie

Symulowane wyżarzanie (SA) [69] to metaheurystyczny algorytm optymalizacji inspirowany procesem wyżarzania w metalurgii – w odróżnieniu od algorytmów ewolucyjnych, SA jest metodą trajektoryjną operującą na pojedynczym rozwiązaniu. Jest to probabilistyczna technika aproksymacji globalnego optimum danej funkcji. Algorytm zaczyna od początkowego rozwiązania i iteracyjnie bada okolice bieżącego rozwiązania, akceptując ruchy, które poprawiają funkcję celu, a czasami akceptując ruchy, które pogarszają funkcję celu z prawdopodobieństwem, które maleje z czasem.

Harmonogram chłodzenia ma zasadnicze znaczenie w SA. Definiuje on, jak temperatura spada wraz z upływającym czasem. Typowe harmonogramy obejmują chłodzenie geometryczne ($T_{k+1} = \alpha T_k$, gdzie α jest stałą mniejszą niż 1), chłodzenie liniowe i chłodzenie logarytmiczne. Zbyt szybkie wychładzanie może prowadzić do przedwczesnej konwergencji do lokalnego optimum, zbyt wolne jest nieefektywne obliczeniowo.

Teoretyczne dowody zbieżności dla SA, w pewnych warunkach (np. wystarczająco powolny harmonogram chłodzenia), gwarantują, że algorytm zbiegnie do globalnego optimum z prawdopodobieństwem 1, gdy liczba iteracji zbliża się do nieskończoności. Warunki te są jednak często niepraktyczne w rzeczywistych zastosowaniach.

2.2 ALGORYTMY ROJOWE I METODY INSPIROWANE NATURĄ

Obok algorytmów ewolucyjnych, drugim ważnym nurtem metaheurystyk jest inteligencja rojowa – algorytmy inspirowane zbiorowym zachowaniem prostych agentów (pszczół, mrówek, ptaków). W tej sekcji omówiono podstawowe i zaawansowane techniki rojowe oraz ich zastosowania.

2.2.1 Podstawowe algorytmy rojowe na przykładzie sztucznej kolonii pszczół

Spośród podstawowych algorytmów rojowych szczegółowo omówiono w tej sekcji jedynie sztuczną kolonię pszczół (ABC), traktując ją jako reprezentatywny przykład klasycznej inteligencji roju – ilustruje ona typowy podział populacji na role, mechanizmy eksploracji i eksploatacji oraz schemat dzielenia się informacją, które są wspólne dla całej rodziny tych metod.

Sztuczna kolonia pszczół (ABC) [64] to jeden z podstawowych algorytmów inteligencji roju zainspirowany zachowaniem żerowania kolonii pszczół miodnych. Jest on uznawany za jeden z fundamentów dziedziny, ponieważ dobrze oddaje podstawowe założenia algorytmów rojowych. Modeluje zachowanie trzech typów pszczół: pszczół robotnic, pszczół obserwujących i pszczół zwiadowczych.

Pszczoły robotnice są powiązane ze specyficznymi źródłami pożywienia (rozwiązaniami). Wykorzystują swoje źródło pożywienia i dzielą się informacjami o jego jakości (sprawności) z pszczołami obserwującymi. Pszczoły obserwujące czekają w ulu i wybierają źródło pożywienia do wykorzystania na podstawie informacji dostarczonych przez pszczoły robotnice. Pszczoły zwiadowcze są odpowiedzialne za losowe eksplorowanie nowych źródeł pożywienia.

W algorytmie ABC każde źródło pożywienia reprezentuje potencjalne rozwiązanie problemu optymalizacji. Liczba zatrudnionych pszczół jest równa liczbie źródeł pożywienia. Pozycja źródła pożywienia reprezentuje rozwiązanie, a ilość nektaru reprezentuje sprawność.

Pszczoły robotnice wykonują przeszukiwanie sąsiedztwa wokół swojego obecnego źródła pożywienia w celu znalezienia lepszego rozwiązania. Zwykle odbywa się to poprzez modyfikację jednego lub więcej wymiarów wektora rozwiązania przy użyciu losowego zaburzenia:

$$v_{ij} = x_{ij} + \varphi_{ij}(x_{ij} - x_{kj}) \tag{2.2}$$

gdzie v_{ij} jest nową wartością j-tego wymiaru i-tego źródła pożywienia, x_{ij} jest bieżącą wartością, x_{kj} jest wartością j-tego wymiaru losowo wybranego sąsiada k, a φ_{ij} jest liczbą losową z przedziału od -1 do 1.

Pszczoły obserwujące wybierają źródło pożywienia z prawdopodobieństwem proporcjonalnym do jego sprawności. Powszechnym mechanizmem selekcji jest selekcja koła ruletki. Po wybraniu źródła pożywienia pszczoła obserwująca wykonuje przeszukiwanie sąsiedztwa podobnie jak pszczoły zatrudnione.

Jeśli źródło pożywienia nie zostanie ulepszone przez wstępnie zdefiniowaną liczbę iteracji (parametr „limit"), jest uważane za porzucone, a odpowiednia zatrudniona pszczoła staje się pszczołą zwiadowczą. Pszczoła zwiadowcza losowo generuje nowe źródło pożywienia.

Eksploatację w ABC napędza przeszukiwanie sąsiedztwa, wykonywane przez pszczoły robotnice i obserwatorki. Różnorodność populacji utrzymują pszczoły zwiadowcze, które generują nowe rozwiązania. Schematy adaptacji parametrów, w których parametry kontrolujące wyszukiwanie w sąsiedztwie (np. zakres φ_{ij}) są dostosowywane, mogą poprawić wydajność.

2.2.2 Zaawansowane techniki rojowe

Rozwój algorytmów rojowych doprowadził do powstania metod hybrydowych, łączących elementy różnych paradygmatów.

Algorytmy hybrydowe

Algorytmy hybrydowego roju łączą mocne strony wielu technik optymalizacji, tworząc bardziej niezawodne i wydajne rozwiązania. Teoretyczne podstawy hybrydyzacji opierają się na twierdzeniu o braku darmowego obiadu (*No Free Lunch*, NFL) [119], które mówi, że żaden pojedynczy algorytm nie jest uniwersalnie lepszy dla wszystkich problemów optymalizacyjnych. Podejścia hybrydowe, łącząc różne algorytmy, wykorzystują uzupełniające się mocne strony składników, osiągając lepszą wydajność w szerszym zakresie problemów.

Hybrydyzację można osiągnąć za pomocą różnych mechanizmów integracji. Hybrydyzacja kooperacyjna obejmuje różne algorytmy współpracujące ze sobą, sekwencyjnie lub jednocześnie, w celu rozwiązania tego samego problemu. Na przykład jeden algorytm może być używany do początkowej eksploracji, a następnie inny do lokalnego udoskonalenia. Hybrydyzacja konkurencyjna obejmuje wiele algorytmów działających równolegle, przy czym najlepiej działający algorytm (lub jego rozwiązanie) jest wybierany na końcu.

Efekty synergistyczne są głównym celem hybrydyzacji. Kombinacje ACO [33]–ABC mogą wykorzystywać zdolność ACO do przyrostowego konstruowania rozwiązań, a następnie lokalne wyszukiwanie ABC. Integracja algorytmów inteligencji roju z lokalnymi metodami wyszukiwania (np. gradientem zstępującym lub symulowanym wyżarzaniem) może zwiększyć eksploatację. Po zidentyfikowaniu przez algorytm roju obiecujących regionów przestrzeni wyszukiwania można zastosować metodę lokalną w celu dopracowania rozwiązań w tych regionach. Taksonomia algorytmów hybrydowych może opierać się na kilku czynnikach, w tym na typach łączonych algorytmów, poziomie integracji (niskiego lub wysokiego poziomu) i strategii kontroli (stałej lub adaptacyjnej). Zasady projektowania algorytmów hybrydowych obejmują identyfikację uzupełniających się mocnych stron algorytmów

składowych, wybór odpowiednich mechanizmów integracji i ostrożne dostrajanie interakcji między komponentami.

Równowaga między eksploracją a eksploatacją jest jeszcze bardziej istotna w systemach hybrydowych. Strategia hybrydyzacji musi zapewnić, że algorytm skutecznie eksploruje przestrzeń wyszukiwania, a jednocześnie wykorzystuje obiecujące regiony. Mechanizmy interakcji między komponentami powinny umożliwiać wymianę informacji i skoordynowane wyszukiwanie. Dostrajanie parametrów hybrydowych jest często bardziej złożone niż dostrajanie pojedynczych algorytmów, ponieważ obejmuje parametry wielu komponentów i ich interakcji.

2.2.3 Efektywność i zastosowania

Analiza wydajności algorytmu roju

Ocena wydajności algorytmów roju wymaga wieloaspektowego podejścia. Podstawą są teoretyczne miary wydajności, które dostarczają wglądu w możliwości algorytmu, oraz wskaźniki eksperymentalne, które oceniają jego praktyczną wydajność.

Analiza teoretyczna koncentruje się na właściwościach, takich jak szybkość zbieżności, złożoność obliczeniowa i gwarancje jakości rozwiązania. Analiza szybkości zbieżności, często trudna dla algorytmów stochastycznych, takich jak algorytmy rojowe, ma na celu określenie, jak szybko algorytm zbliża się do optimum albo do rozwiązania suboptymalnego o akceptowalnej jakości. Może to obejmować analizę oczekiwanego spadku wartości funkcji celu na iterację lub prawdopodobieństwa znalezienia rozwiązania w ramach pewnej tolerancji od optimum. Gwarancje jakości rozwiązania, często ograniczone do określonych klas problemów lub uproszczonych modeli algorytmów, określają granice jakości rozwiązań znalezionych przez algorytm.

Eksperymentalne wskaźniki wydajności zapewniają praktyczną ocenę zachowania algorytmu. Czas wykonania, zwykle mierzony w sekundach, odzwierciedla wydajność obliczeniową algorytmu. Charakterystyki skalowalności opisują, w jaki sposób wydajność algorytmu zmienia się wraz ze wzrostem rozmiaru problemu (np. liczby zmiennych, liczby ograniczeń). Skalowalność jest często oceniana poprzez pomiar czasu wykonania lub jakości rozwiązania.

Struktury analizy statystycznej służą do porównywania wydajności różnych algorytmów. Testy parametryczne są używane, gdy dane spełniają pewne założenia (np. normalność, jednorodność wariancji). Testy nieparametryczne [32, 48] są używane, gdy te założenia nie są spełnione. Procedury wielokrotnych porównań są używane do kontrolowania współczynnika błędów podczas wykonywania wielokrotnych porównań.

Na wydajność algorytmu wpływa kilka czynników.

- Ustawienia parametrów, takie jak wielkość populacji, waga bezwładności lub szybkość parowania feromonów.

- Charakterystyki problemu, takie jak modalność (liczba lokalnych optimów), nieregularność krajobrazu (ang. ruggedness) i wymiarowość przestrzeni wyszukiwania. Problemy wielowymiarowe są trudne dla wielu algorytmów roju, często prowadząc do pogorszenia wyników.

Porównanie z metodami klasycznymi

Porównanie algorytmów roju z klasycznymi metodami optymalizacji (programowanie liniowe, gradient zstępujący) sprowadza się do kompromisu między gwarancjami teoretycznymi a odpornością praktyczną. Jak omówiono w Sekcji 2.1, metody klasyczne zapewniają silniejsze gwarancje zbieżności, lecz wymagają założeń (wypukłość, różniczkowalność), które w rzeczywistych problemach często nie zachodzą. Algorytmy roju [37, 68] tych gwarancji nie dają, ale radzą sobie z funkcjami niewypukłymi, wielomodalnymi i wielkoskalowymi. Wybór algorytmu zależy zatem od konkretnych cech problemu.

2.3 INTERDYSCYPLINARNE PODEJŚCIA W OPTYMALIZACJI

2.3.1 Algorytmy immunologiczne

Algorytmy immunologiczne (Artificial Immune System) są istotne z perspektywy niniejszej pracy, ponieważ demonstrują, jak złożone biologiczne systemy obronne – oparte na hierarchii, pamięci i adaptacji – mogą inspirować skuteczne metaheurystyki. Analogiczne mechanizmy występują w starożytnych armiach, co zostanie wykazane w Rozdziale 3.

Sztuczne systemy immunologiczne to systemy obliczeniowe inspirowane zasadami biologicznego układu odpornościowego. Wykorzystują niezwykłe zdolności układu odpornościowego do rozpoznawania wzorców, adaptacji i zapamiętywania, aby rozwiązywać złożone problemy w różnych dziedzinach.

Biologiczny układ odpornościowy chroni organizm przed patogenami (np. bakteriami, wirusami) poprzez złożoną sieć komórek, tkanek i narządów. Główne składniki obejmują komórki B (które produkują przeciwciała) i komórki T (które pomagają regulować odpowiedź immunologiczną i bezpośrednio zabijają zakażone komórki). Układ odpornościowy wykazuje cechy wykorzystywane w **AIS**:

- **Rozpoznawanie:** Zdolność rozróżniania między własnymi (własnymi komórkami organizmu) a obcymi (obcymi najeźdźcami).

- **Adaptacja:** Zdolność uczenia się i ulepszania swojej reakcji na patogeny w czasie.

- **Pamięć:** Zdolność do zapamiętywania poprzednich spotkań z patogenami, pozwalająca na szybszą i skuteczniejszą reakcję po ponownym narażeniu.

Algorytmy selekcji klonalnej (Clonal Selection Algorithm CSA) są główną klasą algorytmów zainspirowanych teorią selekcji klonalnej. Teoria ta wyjaśnia, w jaki sposób układ odpornościowy reaguje na antygeny (substancje obce) poprzez selektywne klonowanie i proliferację komórek B, które wytwarzają przeciwciała o wysokim powinowactwie (siła wiązania) do antygenu. Obliczeniowo rozwiązania kandydackie są reprezentowane jako przeciwciała, a funkcja obiektywna reprezentuje antygen. Algorytm iteracyjnie wybiera przeciwciała o wysokim powinowactwie, klonuje je (z mutacjami) i eliminuje przeciwciała o niskim powinowactwie.

2.3.2 Optymalizacja inspirowana ewolucją

Zasady ewolucyjne, wykraczające poza te uchwycone przez tradycyjne algorytmy genetyczne, zainspirowały szeroką gamę algorytmów optymalizacji. Algorytmy te wykorzystują koncepcje takie jak dobór naturalny, dynamika populacji i adaptacja w celu poszukiwania optymalnych rozwiązań.

Ewolucja różnicowa (DE) [102, 27] to algorytm oparty na populacji, który wykorzystuje unikalny schemat mutacji oparty na różnicach między osobnikami w populacji. Dla każdego osobnika (wektora docelowego) tworzony jest wektor mutanta poprzez dodanie ważonej różnicy między dwoma innymi losowo wybranymi osobnikami do trzeciego osobnika. Następnie wektor mutanta jest rekombinowany z wektorem docelowym, a lepszy z nich jest wybierany do następnego pokolenia. DE jest szczególnie skuteczny w przypadku problemów ciągłej optymalizacji.

Algorytmy szacowania dystrybucji (EDA) zastępują tradycyjne operatory krzyżowania i mutacji algorytmów genetycznych probabilistycznym modelem populacji. W każdym pokoleniu EDA szacuje rozkład prawdopodobieństwa obiecujących rozwiązań, a następnie pobiera nowe rozwiązania z tego rozkładu. To podejście może być bardziej wydajne niż tradycyjne algorytmy genetyczne, zwłaszcza w przypadku problemów ze złożonymi zależnościami między zmiennymi.

Algorytmy memetyczne (MA) [81] łączą algorytmy ewolucyjne z lokalnymi technikami wyszukiwania. Po zastosowaniu operatorów genetycznych (lub innych mechanizmów ewolucyjnych) procedura lokalnego wyszukiwania jest stosowana do każdego osobnika w celu poprawy jego sprawności. Łączy to globalne możliwości eksploracji algorytmów ewolucyjnych z lokalnymi możliwościami eksploatacji lokalnego wyszukiwania.

Mechanizmy doboru naturalnego, takie jak dobór proporcjonalny do sprawności, dobór turniejowy i dobór oparty na rangach, są podstawą algorytmów ewolucyjnych. Dynamika populacji, w tym czynniki takie jak wielkość populacji, presja selekcyjna i dryf genetyczny, silnie wpływają na wydajność algorytmu. Adaptacja jest osiągana poprzez iteracyjne stosowanie operatorów wariacji (mutacja, rekombinacja) i selekcji, co prowadzi do stopniowej poprawy sprawności populacji.

2.3.3 Systemy oparte na zachowaniu zwierząt

Zachowanie zwierząt, zwłaszcza zachowanie społeczne i inteligencja zbiorowa, zainspirowały wiele algorytmów optymalizacji. Algorytmy te naśladują procesy podejmowania decyzji, wzorce komunikacji i zasady samoorganizacji obserwowane w grupach zwierząt.

Algorytmy takie jak Cuckoo Search (CS) [128] są inspirowane pasożytniczym zachowaniem lęgowym kukułek, które podrzucają swoje jaja do gniazd innych ptaków. Każde gniazdo reprezentuje potencjalne rozwiązanie, a nowe jajo kukułki odpowiada nowemu rozwiązaniu kandydackiemu. Jeśli gospodarz wykryje obce jajo, porzuca gniazdo i buduje nowe w losowej lokalizacji. Eksploracja przestrzeni przeszukiwań opiera się na losowych skokach, które umożliwiają zarówno lokalne przeszukiwanie (krótkie skoki), jak i dalekie eksploracyjne przeniesienia, naśladując wzorce lotu ptaków w poszukiwaniu pożywienia.

Algorytmy nietoperzy (Bat Algorithms) [125] są inspirowane zachowaniem echolokacyjnym nietoperzy. Nietoperze emitują impulsy dźwiękowe i wykorzystują echa do nawigacji i lokalizowania ofiary. W algorytmie każdy nietoperz reprezentuje potencjalne rozwiązanie, a jego pozycja i prędkość są aktualizowane na podstawie jego własnego doświadczenia i najlepszego dotychczas znalezionego rozwiązania. Algorytm modeluje również głośność i częstotliwość emisji impulsów nietoperzy.

Algorytmy Firefly [123] są inspirowane wzorami migania świetlików, które są wykorzystywane do komunikacji i przyciągania partnerów. W algorytmie każdy świetlik reprezentuje potencjalne rozwiązanie, a jego atrakcyjność jest proporcjonalna do jego jasności (sprawności). Świetliki poruszają się w kierunku jaśniejszych świetlików, eksplorując przestrzeń poszukiwań.

Matematyczne modelowanie tych zachowań zwierząt obejmuje uproszczone reprezentacje procesów biologicznych. Zachowania zbiorowe, takie jak gromadzenie się, rojenie lub stado, są modelowane przy użyciu modeli opartych na agentach, w których każdy agent przestrzega prostych reguł opartych na swoim lokalnym środowisku i interakcjach z innymi agentami. Procesy podejmowania decyzji można modelować przy użyciu reguł probabilistycznych, logiki rozmytej lub sieci neuronowych. Wzory interakcji społecznych, takie jak komunikacja za pomocą feromonów, sygnałów wizualnych lub wokalizacji, są reprezentowane przez mechanizmy wymiany informacji między agentami.

Zachowanie emergentne [18, 93], w którym złożone globalne wzorce powstają z prostych lokalnych interakcji, charakteryzuje wiele algorytmów inspirowanych przez zwierzęta. Zasady samoorganizacji są stosowane w celu osiągnięcia odpornego i adaptacyjnego zachowania.

2.3.4 Inspiracje socjologiczne

Podejścia socjologiczne są szczególnie bliskie metaforze wojskowej – hierarchia społeczna, systemy motywacyjne i procesy podejmowania decyzji grupowych mają bezpośrednie odpowiedniki w organizacji starożytnych armii.

Algorytmy oparte na zachowaniach społecznych czerpią inspirację ze sposobów, w jakie ludzie wchodzą w interakcje, uczą się i rozwiązują problemy zbiorowo [90]. Algorytmy te wykorzystują zasady psychologii społecznej, socjologii i nauk kognitywnych do projektowania strategii optymalizacji.

Teoria poznawczo-społeczna, która akcentuje rolę obserwacji, naśladownictwa i samoskuteczności w uczeniu się, jest podstawą kilku algorytmów optymalizacji. Uczenie się społeczne, w którym jednostki uczą się na podstawie doświadczeń innych, jest modelowane poprzez włączanie informacji z najlepszych rozwiązań znalezionych przez inne osoby w populacji lub poprzez bardziej złożone mechanizmy, takie jak optymalizacja oparta na nauczaniu i uczeniu się. Ewolucja kulturowa, w której wiedza i zachowania są przekazywane i dostosowywane w czasie, inspiruje algorytmy utrzymujące „kulturę" dobrych rozwiązań lub strategii wyszukiwania. Inteligencja zbiorowa, czyli zdolność grup do rozwiązywania problemów skuteczniej niż jednostki, jest często osiągana poprzez zdecentralizowane podejmowanie decyzji i dzielenie się informacjami.

Modelowanie hierarchii społecznej, w której jednostki mają różne role i poziomy wpływu, można włączyć do algorytmów optymalizacji. Procesy podejmowania decyzji grupowych, takie jak tworzenie konsensusu i mechanizmy głosowania, można wykorzystać do łączenia rozwiązań lub preferencji wielu jednostek. Wpływy sieci społecznych, w których interakcje między jednostkami są ustrukturyzowane przez sieć społeczną, można modelować w celu kontrolowania przepływu informacji i wpływania na proces wyszukiwania.

2.3.5 Uczenie ze wzmocnieniem w optymalizacji metaheurystycznej

Uczenie ze wzmocnieniem (*Reinforcement Learning*, RL) stanowi trzeci – obok uczenia nadzorowanego i nienadzorowanego – filar uczenia maszynowego. W paradygmacie RL agent podejmuje sekwencyjne decyzje w środowisku, obserwując stany, wykonując akcje i otrzymując nagrody, a jego celem jest maksymalizacja skumulowanej nagrody w czasie [104]. Formalnie problem RL modeluje się jako **proces decyzyjny Markowa** (MDP): krotkę $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, gdzie $\mathcal{S}$ to zbiór stanów, $\mathcal{A}$ to zbiór akcji, $P(s'|s,a)$ to prawdopodobieństwo przejścia, $R(s,a)$ to funkcja nagrody, a $\gamma \in [0,1)$ to współczynnik dyskontowania.

Najpopularniejszym algorytmem RL jest **Q-learning** [114], metoda bezmodelowa (*model-free*), która uczy się funkcji wartości akcji $Q(s,a)$ bez znajomości modelu przejść środowiska. Reguła aktualizacji Q-learningu ma postać:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{2.3}$$

gdzie α to współczynnik uczenia, a wyrażenie w nawiasie kwadratowym to *błąd różnicy czasowej* (TD error). Watkins i Dayan [114] udowodnili, że przy wystarczającej eksploracji i malejącym współczynniku uczenia tablicowy Q-learning zbiega do optymalnej funkcji wartości Q^*.

W optymalizacji metaheurystycznej RL znalazł zastosowanie przede wszystkim w **adaptacyjnym wyborze operatorów** (*Adaptive Operator Selection*, AOS). Problem AOS polega na dynamicznym dobieraniu najskuteczniejszego operatora przeszukiwania spośród dostępnego repertuaru, w zależności od bieżącego stanu procesu optymalizacji. Thierens [106] zaproponował adaptacyjne mechanizmy doboru operatorów krzyżowania w algorytmach genetycznych. Li i in. [72] zastosowali Q-learning do adaptacyjnego doboru operatorów mutacji w ewolucji różnicowej (DE), dyskretyzując stan optymalizacji i ucząc politykę wyboru operatora na podstawie historii popraw fitnessu. Fialho i in. [39] porównali metody AOS oparte na wielorękim bandycie (*multi-armed bandit*) z metodami opartymi na RL, wykazując, że podejścia uwzględniające stan procesu (takie jak Q-learning) przewyższają metody bezstanowe na problemach o zmiennej strukturze.

Ważnym zagadnieniem jest wybór między **tablicowym Q-learningiem** a **głębokim RL** (DQN, PPO). Tablicowy Q-learning przechowuje jawną tabelę $|\mathcal{S}| \times |\mathcal{A}|$ wartości i gwarantuje zbieżność przy skończonej przestrzeni stanów, ale wymaga wystarczającego pokrycia par stan–akcja. Głęboki RL aproksymuje $Q(s,a)$ siecią neuronową, radzi sobie z ciągłymi lub dużymi przestrzeniami stanów, ale traci gwarancje zbieżności i wymaga znacznie więcej danych treningowych. Dla problemów o niewielkiej, zdyskretyzowanej przestrzeni stanów (rzędu 10^2–10^3) Q-learning tablicowy pozostaje metodą z wyboru ze względu na prostotę, interpretowalność i gwarancje teoretyczne [115, 104].

Warunki zbieżności i ich rola w praktyce. Klasyczne dowody zbieżności tablicowego Q-learningu [114] wymagają spełnienia warunków Robbinsa-Monro: $\sum_t \alpha_t = \infty$ oraz $\sum_t \alpha_t^2 < \infty$, co praktycznie oznacza malejący współczynnik uczenia, np. $\alpha_t = 1/t$. W implementacjach praktycznych, w tym w module Commander AI niniejszej pracy, stosuje się jednak *stały* (lub ograniczony) α – wówczas Q zbiega do ϵ-otoczenia Q^*, a nie do dokładnej wartości optymalnej. Strategie adaptacyjnej selekcji operatorów wpisują się w szerszy nurt *Adaptive Operator Selection* i *parameter control*, których przegląd przedstawili Da Costa i in. [26] oraz Karafotias i in. [65].

Zastosowanie RL w metaheurystykach niesie specyficzne wyzwania: *niestacjonarność środowiska* (stan optymalizacji zmienia się w trakcie procesu), *opóźnione*

nagrody (efekt zmiany taktyki może być widoczny dopiero po wielu iteracjach) oraz *ograniczona liczba epizodów treningowych* (każdy przebieg optymalizacji to jeden epizod). Strategie radzenia sobie z tymi wyzwaniami obejmują *uczenie transferowe* (ang. *transfer learning*; przenoszenie Q-tabeli między przebiegami), adaptacyjne współczynniki eksploracji (ε-greedy z malejącym ε) oraz dyskretyzację przestrzeni stanów do niewielkiej liczby poziomów.

W niniejszej pracy moduł Commander AI wykorzystuje tablicowy Q-learning do adaptacyjnego doboru taktyk (Rozdział 4, Sekcja 4.2.5), operując na zdyskretyzowanej przestrzeni 243 stanów i 6 akcji – podejście motywowane gwarancjami zbieżności i interpretowalnością polityki, przy jednoczesnym zachowaniu wystarczającego pokrycia stanów w ramach treningu na 10 funkcjach treningowych (zbiór rozłączny z 33 funkcjami testowymi, szczegóły w Rozdziale 5).

2.3.6 Nowe inspiracje militarne

Jak już wspomniano w poprzednim rozdziale, strategia wojskowa [66, 110] – nastawiona na osiąganie celów w zmiennym i niepewnym środowisku – daje się przełożyć na algorytmy optymalizacji. Wymaga to jednak zidentyfikowania analogii między operacjami wojskowymi a procesami poszukiwań.

Matematyczne modelowanie struktur dowodzenia wojskowego może być używane do tworzenia hierarchicznych algorytmów optymalizacji. Algorytm optymalizacji może naśladować hierarchię wojskową, przy czym wyższe poziomy koncentrują się na globalnej eksploracji, a niższe na lokalnej eksploatacji. Strategie alokacji zasobów, takie jak ustalanie składu roju - przydziału wojsk do obszarów krytycznych, mogą być modelowane jako adaptacyjne mechanizmy kontroli parametrów lub podejścia wielopopulacyjne, w których zasoby obliczeniowe są przydzielane do różnych strategii wyszukiwania. Procesy podejmowania decyzji taktycznych, takie jak wybór między różnymi opcjami ataku, mogą być modelowane jako adaptacyjny wybór operatora lub mechanizmy przełączania strategii.

Wdrożenie wojskowych zasad strategicznych obejmuje zdefiniowanie odpowiednich reprezentacji i operatorów. Mechanizmy kontroli terytorium, w których różni agenci lub subpopulacje kontrolują różne regiony przestrzeni wyszukiwania, mogą promować różnorodność i zapobiegać przedwczesnej konwergencji. Strategie rozmieszczania zasobów mogą opierać się na sprzężeniu zwrotnym dotyczącym wydajności, przydzielając więcej zasobów do obszarów, w których poczyniono postęp. Metody adaptacji taktycznej, w których strategia wyszukiwania jest dostosowywana na podstawie bieżącej sytuacji, można wdrożyć za pomocą adaptacyjnej kontroli parametrów lub przełączania strategii.

Taktyki bitewne jako modele obliczeniowe

Taktyki bitewne, które obejmują określone manewry i działania jednostek wojskowych na polu bitwy, należałoby przełożyć na modele obliczeniowe w celu optymalizacji. Wiąże się to z przedstawieniem elementów taktycznych jako algorytmicznych komponentów i operacji.

Klasyczne formacje bitewne, takie jak **falanga** (gęsta prostokątna formacja) lub **formacja liniowa**, można modelować jako różne sposoby organizacji agentów w przestrzeni poszukiwań. Na przykład formacja liniowa może być używana do systematycznej eksploracji określonego regionu, podczas gdy bardziej rozproszona formacja może być używana do szerszej eksploracji. Manewry ofensywne i defensywne, takie jak **ataki flankujące** lub **fortyfikacje obronne**, można modelować jako różne operatory poszukiwań lub mechanizmy obsługi ograniczeń. Koordynacja jednostek taktycznych, taka jak **skoordynowane ataki** lub **reakcje obronne**, może być implementowana przy użyciu mechanizmów komunikacji i współpracy agentów.

Matematyczna reprezentacja organizacji przestrzennej pola bitwy może obejmować zdefiniowanie siatki lub innej struktury przestrzennej w celu przedstawienia przestrzeni poszukiwań. Wzory ruchu jednostek można modelować jako dyskretne kroki lub ciągłe trajektorie, kierowane przez siły przyciągania/odpychania, ślady feromonów lub inne mechanizmy. Reguły zaangażowania, które określają wynik interakcji między przeciwnymi siłami, można modelować jako porównania sprawności lub inne mechanizmy selekcji.

Metryki oceny postępu bitwy, takie jak **zadane straty** lub **zdobyte terytorium**, mogą być analogiczne do wartości funkcji obiektywnych lub innych miar wydajności. Analogie warunków zwycięstwa, takie jak **zdobycie głównego celu** lub **wyeliminowanie przeciwnych sił**, można mapować na kryteria zakończenia dla algorytmu optymalizacji.

Istniejące algorytmy rojowe inspirowane polem walki

Choć algorytmy metaheurystyczne czerpią inspirację z najróżniejszych zjawisk przyrodniczych i społecznych – od zachowań ptaków i mrówek po prawa fizyki – tematyka działań wojennych i taktyki bitewnej pozostawała przez długi czas stosunkowo niewykorzystanym źródłem inspiracji w dziedzinie inteligencji rojowej. Za wczesnego prekursora można uznać **Imperialist Competitive Algorithm** (ICA), zaproponowany w 2007 roku przez Atashpaz-Gargariego i Lucasa [4], który modeluje rywalizację imperialistyczną między państwami – mocarstwa (imperialiści) przyciągają kolonie, a słabsze imperia upadają na rzecz silniejszych. Algorytm ten, choć nawiązuje do procesów geopolitycznych i militarnych, koncentruje się jednak na dynamice konkurencji między mocarstwami, a nie na bezpośrednim odwzorowaniu taktyk pola walki. *Demarkacja względem ABO*: ICA operuje na monolitycznej populacji rozwiązań grupowanych w imperia z jednym operatorem ruchu (asymilacja kolonii ku stolicy) i jednym mechanizmem rywalizacji (przejmowanie kolonii), nie

różnicując ról agentów ani nie adaptując strategii do stanu krajobrazu fitness. ABO wprowadza natomiast heterogeniczną populację 6 typów jednostek o odrębnych operatorach przeszukiwania, dynamiczny system honoru z promocjami między rolami oraz adaptacyjny moduł Q-learning wybierający taktykę globalną co τ iteracji – żaden z tych komponentów nie ma odpowiednika w ICA. Wspólne militarne źródło inspiracji nie implikuje zatem strukturalnego podobieństwa algorytmicznego (porównanie empirycznej przewagi: CommanderABO rang 6,77 vs. ICA rang 20,02 w pełnym rankingu Friedmana 46 algorytmów, por. Sekcja 6.3.2).

Pierwszym algorytmem rojowym bezpośrednio inspirowanym taktykami pola walki był **Enemy Surrounding Inspired Optimisation Algorithm** (ESIOA), przedstawiony w 2021 roku przez Baumgarta [10]. Algorytm ten czerpie z klasycznej taktyki okrążania przeciwnika, w której siły atakujące manewrują wokół pozycji wroga, stopniowo zawężając pierścień okrążenia. W kontekście optymalizacji agenci przeszukujący przestrzeń rozwiązań naśladują ten manewr, eksplorując otoczenie aktualnie najlepszego rozwiązania z różnych kierunków i stopniowo zbliżając się do optimum. Praca ta zapoczątkowała w ramach algorytmów metaheurystycznych nurt inspiracji taktykami pola walki, rozwijany w kolejnych publikacjach.

W kontynuacji tego nurtu Baumgart opublikował w 2023 roku dwie kolejne prace. Pierwsza z nich, **Numidian Swarm Riders** [11], proponuje nowe podejście do optymalizacji oparte na mądrości starożytnej kawalerii numidyjskiej. Numidyjscy jeźdźcy, znani ze swoich niezwykle zwinnych i nieprzewidywalnych manewrów na polu bitwy, stosowali taktykę polegającą na szybkich atakach, wycofaniach i ponownym natarciu z innego kierunku. W algorytmie tym agenci naśladują te zachowania, łącząc intensywną eksplorację z dynamicznymi zmianami kierunku przeszukiwania, co pozwala na skuteczne unikanie minimów lokalnych. Druga praca, zatytułowana *The Elephant in the Room: Swarm Algorithms Inspired by Warfare* [12], napisana wspólnie z Leonidem Rusanovem, stanowi szersze opracowanie problematyki algorytmów rojowych inspirowanych działaniami wojennymi, analizując potencjał taktyk bitewnych jako źródła inspiracji dla nowych metaheurystyk i systematyzując dotychczasowe podejścia w tej dziedzinie.

Równolegle do prac Baumgarta, Ayyarao i in. [7] zaproponowali **War Strategy Optimization** (WSO) – algorytm inspirowany starożytnymi strategiami wojennymi, w szczególności hinduskimi formacjami bojowymi zwanymi *Vyuha*. WSO modeluje strategiczne przemieszczanie się oddziałów wojskowych podczas bitwy, traktując każdego żołnierza jako agenta dynamicznie poruszającego się w kierunku wartości optymalnej. Algorytm implementuje dwie podstawowe strategie wojenne: strategię ataku (eksploracja) i strategię obrony (eksploatacja). Warto zaznaczyć, że WSO porusza również wątek **łączenia jednostek konnych i słoni bojowych** w ramach jednej formacji – motyw rozwijany równolegle i niezależnie w pracach autora [11, 12] – co świadczy o kształtowaniu się wspólnego repertuaru metafor militarnych w tej rodzinie algorytmów.

W 2024 roku Xu i in. [122] zaproponowali **Multiplayer Battle Game-Inspired Optimizer** (MBGO) – algorytm inspirowany nie tyle historycznymi bitwami, co współczesnymi wieloosobowymi grami typu *battle royale*. MBGO upraszcza mechaniki tych gier do dwóch faz: fazy ruchu, w której gracze przemieszczają się w kierunku bezpiecznych stref o wyższym potencjale przetrwania, oraz fazy walki, symulującej różnorodne strategie przyjmowane przez graczy w zależności od sytuacji bojowej. Choć inspiracja pochodzi ze świata gier komputerowych, a nie z rzeczywistych działań wojennych, MBGO wpisuje się w szerszą kategorię algorytmów wykorzystujących mechanizmy konfliktu i rywalizacji jako motory przeszukiwania.

Tabela 1. Zestawienie algorytmów inspirowanych polem walki – porównanie cech architektonicznych

Rok	Algorytm	Heter.	RL	Rekon.	Honor
2007	ICA	✓	–	–	⊖
2021	ESIOA	–	–	⊖	–
2022	WSO	✓	–	–	⊖
2023	NSR	⊖	–	⊖	–
2024	MBGO	⊖	–	⊖	–
2025	**ABO**	✓	✓	✓	✓

Legenda: ✓ – cecha obecna w pełnej, dedykowanej formie; ⊖ – cecha obecna częściowo lub w formie zalążkowej (np. ograniczony mechanizm, brak wyodrębnionego modułu); – – cecha nieobecna.

Heter. = heterogeniczne typy jednostek (różne role i równania ruchu); RL = uczenie ze wzmocnieniem jako mechanizm meta-kontroli (wybór taktyki/operatora online); Rekon. = wyodrębniony moduł rozpoznania przestrzeni (mapa, pamięć obszarów); Honor = persystentna miara reputacji agenta wpływająca na jego zachowanie.

Uwaga metodologiczna: dobór cech nie jest neutralny i odzwierciedla wymiary istotne dla architektury ABO. Porównanie ma charakter *architektoniczny*, nie wydajnościowy – obiektywną ocenę skuteczności zapewnia benchmark empiryczny przedstawiony w Rozdziale 6.

Tabela 1 zestawia chronologię algorytmów inspirowanych polem walki wraz z porównaniem kluczowych cech architektonicznych. Algorytmy te różnią się źródłem inspiracji (formacje starożytne, kawaleria, gry wojenne), ale łączy je wykorzystanie dynamiki konfliktu – z podziałem na eksplorację (zwiad, manewr) i eksploatację (atak, obrona) – jako mechanizmu napędzającego optymalizację. Rosnąca liczba publikacji wskazuje, że tematyka wojenna jest wciąż mało zbadanym, ale wartym uwagi kierunkiem w projektowaniu metaheurystyk.

2.3.7 Krytyka metaforyki metaheurystyk i pozycjonowanie pracy

Przegląd algorytmów inspirowanych przyrodą, pszczołami, wilkami, świetlikami czy bitwami nie byłby kompletny bez odniesienia do rosnącego nurtu krytycznego w społeczności badawczej. Krytyka ta jest nie tylko uprawniona, lecz

wręcz konieczna z perspektywy higieny metodologicznej dziedziny – a niniejsza praca świadomie się w tym nurcie sytuuje.

Pierwszą systematyczną krytykę przedstawił Weyland [116], wykazując, że Harmony Search Algorithm jest matematycznie tożsamy ze strategią ewolucyjną $(\mu + 1)$-ES, lecz został opublikowany jako „nowy" algorytm wyłącznie dzięki przebraniu w metaforę improwizacji muzycznej. Drugą falę zapoczątkował Sörensen [101] w głośnym artykule *Metaheuristics–the metaphor exposed*, w którym wskazał, że dziesiątki algorytmów rojowych (od ABC, przez Cuckoo Search, Firefly Algorithm, aż po Krill Herd i Water Cycle) wnoszą znikomy wkład formalny i różnią się jedynie ozdobnym narratywem. Trzecią, najbardziej radykalną falę reprezentują Camacho-Villalón, Dorigo i Stützle [22] oraz manifest Aranha i in. [2], którzy explicite domagają się *zaprzestania* publikowania algorytmów uzasadnianych wyłącznie metaforą, bez wykazania mechanizmu algorytmicznego niesprowadzalnego do istniejących metod.

Argumentacja krytyków sprowadza się do trzech głównych punktów:

1. **Nadmiarowość matematyczna** – wiele „nowych" algorytmów to reparametryzacje PSO, ES lub DE z kosmetycznymi zmianami wzorów aktualizacji (Camacho-Villalón i in. [22] pokazują to dla GWO, MFO, WOA, FA, BA, ALO).

2. **Brak ablacji** – autorzy nie izolują wkładu nowych komponentów, więc nie wiadomo, czy poprawa wynika z mechanizmu, czy z lepszego strojenia względem konkurencji.

3. **Asymetryczne porównania** – nowy algorytm jest strojony na zbiorze testowym, podczas gdy konkurenci pracują z domyślnymi parametrami z implementacji bibliotecznych (mealpy, EvoloPy, NiaPy).

Pozycjonowanie Ancient Battlefield Optimizer. Powyższa krytyka jest *trafna* i niniejsza praca świadomie się do niej ustosunkowuje, deklarując następujące stanowisko:

(P1) Metafora służy zapamiętywaniu, a nie definiowaniu algorytmu.
Wojskowe nazewnictwo (falanga, kawaleria, łucznicy, taktyki: falanga, szyk ukośny, oskrzydlenie) ma jedynie ułatwić czytelnikowi zrozumienie i zapamiętanie poszczególnych elementów algorytmu. Sam algorytm można *w pełni odtworzyć* na podstawie pseudokodu (Załącznik 7.4) oraz opisu formalnego z Rozdziału 4, bez żadnego odwoływania się do starożytności. Każdy typ jednostki odpowiada konkretnemu operatorowi matematycznemu (Tabela 88):

- **Ciężka piechota** (HeavyInfantry) – przeszukiwanie po współrzędnych z krokiem metody Neldera-Meada,

- **Lekka piechota** (LightInfantry) – operator mutacji DE/rand/1 z ewolucji różnicowej,
- **Łucznicy** (Archers) – próbkowanie wielopunktowe z odległości,
- **Kawaleria** (Cavalry) – lot Lévy'ego z utrzymaniem kierunku ruchu,
- **Rydwany** (Chariots) – ruch po trajektorii Cauchy'ego z bezwładnością,
- **Słonie bojowe** (WarElephants) – skoki Cauchy'ego między dolinami funkcji celu z kryterium akceptacji Metropolisa.

Innymi słowy: nazwy są wojskowe, ale za każdą z nich kryje się opracowana metoda numeryczna.

(P2) Ablacja jest wykonana i raportowana. Rozdział 6 prezentuje cztery warianty ABO: *ManualPhases* (heurystyczny scheduler bez Q-learningu), *QTable4K* i *QTable10K* (różne budżety treningu meta-kontrolera) oraz *CommanderABO* (pełna konfiguracja). Różnica rang między ManualPhases (6,21) a CommanderABO (6,77) izoluje wkład meta-uczenia od wkładu samej heterogenicznej populacji – zależny od kryterium.

(P3) Asymetria strojenia jest przyznana. Rozdział 7 (sekcja ograniczeń) explicite informuje, że ABO był strojony na rozłącznym zbiorze treningowym (10 funkcji), podczas gdy konkurenci pracują z parametrami rekomendowanymi w oryginalnych publikacjach (nie „fabrycznymi mealpy"). Skala obserwowanej przewagi (rzędu wielkości większa niż typowa różnica strojony/niestrojony [25]) pozostaje istotna nawet po skorygowaniu o tę asymetrię.

(P4) Demarkacja architektoniczna. Tabela 1 oraz dyskusja w Sekcji 2.3 dokumentują demarkację względem pięciu poprzedników (ICA, WSO, ESIOA, NSR, MBGO): choć pojedyncze cechy występują u niektórych poprzedników (np. pełna heterogeniczność ról w ICA i WSO, zalążkowy mechanizm rozpoznania w ESIOA/NSR/MBGO, miary reputacji w ICA i WSO), żaden z pięciu wcześniejszych algorytmów inspirowanych polem walki nie łączy *wszystkich czterech* komponentów – w szczególności RL-owa meta-kontrola wyboru taktyki pozostaje cechą wyłączną ABO. Wspólne militarne źródło inspiracji nie implikuje strukturalnego podobieństwa algorytmicznego.

Walidacja na 165 konfiguracjach funkcja×wymiar z 46 algorytmami konkurencyjnymi i 16 500 powtórzeniami spełnia kryteria reprodukowalności zalecane przez Sörensena [101] i standardy IEEE CEC.

2.4 ZIDENTYFIKOWANA LUKA BADAWCZA

Z przeglądu literatury wynika, że zdecydowana większość metaheurystyk czerpie z metafor biologicznych, ewolucyjnych, immunologicznych lub fizycznych. Tabela 2 zestawia najczęściej przywoływane algorytmy referencyjne.

Tabela 2. Najczęściej przywoływane algorytmy referencyjne.

Skrót	**Pełna nazwa**	**Źródło**
ABC	Artificial Bee Colony	[64]
GWO	Grey Wolf Optimizer	[83]
WOA	Whale Optimization Algorithm	[82]
GA	Genetic Algorithm	[57]
DE	Differential Evolution	[102]
ES	Evolution Strategies	[97]
SA	Simulated Annealing	[69]

Metafora wojskowa była dotąd wykorzystana w zaledwie kilku pracach, z czego duża część z nich stanowiła element badań, który pozwolił na stworzenie tej pracy.

Żaden z tych algorytmów jednak nie łączy: (1) heterogenicznych typów jednostek o różnych rolach (eksploracja, eksploatacja, wsparcie), (2) systemu dowodzenia opartego na uczeniu ze wzmocnieniem, (3) mechanizmu rozpoznania i mapowania przestrzeni poszukiwań, (4) systemu honoru modyfikującego parametry jednostek. Brak algorytmu integrującego te cztery elementy to luka, którą wypełnia opracowany w tej rozprawie Ancient Battlefield Optimizer (ABO).

3. TEORETYCZNE PODSTAWY STAROŻYTNYCH STRATEGII BITEWNYCH

„Taktyka to robienie tego, co możesz, z tym, co masz."
„Tactics means doing what you can with what you have." (ang.)
– Saul Alinsky

Jak wskazano w Rozdziale 2, strategia wojskowa pozostaje niewykorzystanym źródłem inspiracji dla algorytmów optymalizacyjnych.

Rozdział ten tworzy pomost między historią wojskowości a informatyką. Przeanalizowano wybrane formacje i taktyki używane w starożytności, identyfikując uniwersalne zasady, które można przełożyć na operatory algorytmów optymalizacyjnych w taki sposób aby przedstawić sposób odniesienia się do historii w ujęciu algorytmicznym. Od geometrycznej precyzji greckiej falangi, przez błyskawiczne manewry kawalerii mongolskiej, po systemy rozpoznania i dowodzenia armii rzymskich [66, 95].

3.1 METODY ADAPTACJI STRATEGII BITEWNYCH

Przekształcenie strategii wojskowych w algorytmy optymalizacji wymaga systematycznego procesu abstrakcji i mapowania.

1. **Mapowanie koncepcyjne:** Pierwszym krokiem jest zidentyfikowanie zasad strategii wojskowej lub taktyki i mapowanie ich na odpowiadające im elementy optymalizacji. Odpowiednio:
    - *Jednostka wojskowa → Agent/rozwiązanie wyszukiwania*
    - *Pole bitwy/teren → Przestrzeń wyszukiwania/Przestrzeń funkcji celu*
    - *Siła na pozycji wroga → Wartość/ograniczenia funkcji celu*
    - *Ruch/manewr → Operator wyszukiwania (np. mutacja, krok w spadku gradientowym)*
    - *Atak/obrona → Eksploatacja/eksploracja*

- *Struktura dowodzenia → Hierarchia kontroli algorytmu*
- *Zbieranie informacji wywiadowczych → Próbkowanie/ocena funkcji celu*
- *Warunek zwycięstwa → Kryterium zakończenia*

3.2 ANALIZA WYBRANYCH ELEMENTÓW STRATEGII BITEWNYCH

3.2.1 Typy jednostek – agentów

Proponowany algorytm zakłada kilka typów jednostek, zorganizowanych w różnych formacjach i stosujących odmienne taktyki. Mobilność często przesądzała o wyniku starożytnych bitew [66] – szybki ruch pozwalał zgromadzić siły w decydującym punkcie, oskrzydlić wroga lub przeprowadzić rozpoznanie. Prędkość zależała od typu jednostki (piechota, kawaleria, rydwany), terenu i wsparcia logistycznego. Poniżej przedstawiono poszczególne typy jednostek wraz z ich historycznym znaczeniem i przełożeniem algorytmicznym. (Szczegółowe sformalizowanie algorytmiczne zamieszczono w Rozdziale 4.)

3.2.2 Kawaleria – siła uderzeniowa

Kawaleria była siłą uderzeniową starożytnych armii, zdolną do przełamywania linii wroga lub wykorzystywania przełamań [66, 45].

Fizyka szarż kawalerii

- **Pęd:** Pęd (p) szarżującej jednostki kawalerii jest iloczynem jej masy (m) i prędkości (v): $p = mv$. Cięższa, szybsza jednostka ma większy pęd [45].
- **Siła uderzenia:** Siła uderzenia (F) jest związana ze zmianą pędu (Δp) w czasie uderzenia (Δt): $F = \Delta p/\Delta t$. Krótszy czas uderzenia (np. z powodu sztywnej formacji) skutkuje większą siłą.

Zasady taktyczne

- **Czas:** Szarża kawalerii musiała być odpowiednio wymierzona w czasie, aby zmaksymalizować jej wpływ. Szarża zbyt wczesna mogła narazić kawalerię na ogień wroga, podczas gdy szarża zbyt późna mogła zmarnować okazję do wykorzystania przełomu.
- **Wektory podejścia:** Kąt podejścia miał duże znaczenie. Frontalny atak na przygotowaną linię wroga bywał samobójczy. Skuteczniejsze były ataki flankujące lub szarże na tyły.
- **Wybór celu:** Kawaleria zazwyczaj atakowała słabsze punkty w linii wroga, takie jak łucznicy, tyralierowie lub już zaangażowane jednostki piechoty.

Opcje formacji

- **Formacja klina:** Formacja trójkątna zaprojektowana w celu przebicia linii wroga.
- **Formacja liniowa:** Szeroka, płytka formacja używana do ataków zamiatających lub do pokrycia większego obszaru.
- **Taktyka scytyjska/partyjska:** Wykorzystanie pozorowanych odwrotów i strzelanie podczas odwrotu.

Kawaleria mongolska

Osobny przypadek stanowi kawaleria mongolska pod wodzą Czyngischana [66] i jego następców, która wyniosła mobilność konną na poziom nieosiągalny dla innych armii starożytności i średniowiecza. Mongolscy jeźdźcy, uzbrojeni w potężne łuki kompozytowe o zasięgu przekraczającym 300 metrów, łączyli siłę ognia z niespotykaną szybkością przemieszczania – armia mogła pokonywać 100 km dziennie, trzykrotnie więcej niż europejska kawaleria. Główną taktyką był pozorowany odwrót (*mangudai*): oddział symulował ucieczkę, wciągając wroga w pościg i rozciągając jego formację, po czym główne siły uderzały z flanek lub tyłów na rozproszonego przeciwnika. Ten manewr jest szczególnie ważny z perspektywy optymalizacji – odpowiada mechanizmowi ucieczki z minimów lokalnych, w którym tymczasowe pogorszenie rozwiązania (pozorny odwrót) prowadzi do eksploracji nowych regionów przestrzeni i znalezienia lepszego optimum globalnego.

3.2.3 Piechota ciężka – fundament armii

Piechota ciężka to trzon niemal każdej starożytnej armii. Greccy hoplici [24], rzymscy legioniści [50], macedońscy falangici – wszyscy ciężko uzbrojeni, zdolni do utrzymania pozycji i metodycznego niszczenia przeciwnika.

Charakterystyka taktyczna

- **Niska mobilność, wysoka siła rażenia:** Ciężkie uzbrojenie (hełm, pancerz, tarcza, włócznia lub miecz) ograniczało prędkość marszu, ale dawało przewagę w bezpośrednim starciu. Rzymski legionista niósł około 30 kg ekwipunku co ograniczało jego mobilność. [50].
- **Eksploatacja terenu:** Piechota ciężka najlepiej radziła sobie na pozycjach już zajętych – broniąc wzgórz, mostów, przesmyków. Nie szukała nowych pozycji, lecz wykorzystywała te, które już zajmowała.
- **Pamięć formacyjna:** Żołnierze potrafili odtwarzać wyuczone formacje nawet po ich rozbiciu. Spartańscy hoplici ćwiczyli manewry od dzieciństwa [24] i potrafili wrócić do optymalnej konfiguracji po chaosie walki.

Przełożenie na algorytm

Piechota ciężka przekłada się na agentów skoncentrowanych na eksploatacji – powolnych, ale dokładnych przeszukiwaczy otoczenia najlepszych znanych rozwiązań. Wysoka wartość parametru pamięci odzwierciedla zdolność legionistów do zapamiętywania i odtwarzania skutecznych formacji.

3.2.4 Piechota lekka – mobilność i rozpoznanie

Piechota lekka – *psiloi* w Grecji, *velites* w Rzymie, *peltaści* w Tracji [24, 50] – uzupełniała ciężkie formacje. Lekko uzbrojeni żołnierze operowali w luźnym szyku, nękając wroga z dystansu przed głównym starciem.

Rola taktyczna

- **Elastyczność:** Piechota lekka szybko zmieniała pozycje, wycofywała się przed natarciem i atakowała ponownie z innego kierunku. Ta mobilność sprawdzała się tam, gdzie trzeba było zarówno szukać, jak i wykorzystywać szanse.
- **Osłona i rozpoznanie:** Osłaniała główne siły, a jednocześnie zbierała informacje o pozycjach wroga. Rzymscy *velites* rozpoczynali bitwę od salwy oszczepów, testując siłę formacji przeciwnika.
- **Adaptacyjność terenowa:** Tyralierzy mogli skutecznie operować w lasach, górach i na bagnach – tam, gdzie ciężka piechota była bezradna.

Przełożenie na algorytm

W algorytmie piechota lekka to agenci o zrównoważonych parametrach eksploracji i eksploatacji. Umiarkowana prędkość, średnia pamięć i równomierny rozkład zdolności czynią z nich uniwersalny element populacji, zdolny do pracy zarówno w regionach obiecujących, jak i niezbadanych.

3.2.5 Łucznicy – siła na dystans

Łucznicy byli obecni w armiach od Egiptu i Asyrii po angielskich długołuczników późnego średniowiecza [95]. Rażenie wroga z dystansu zmieniało dynamikę pola bitwy – pozwalało osłabić przeciwnika zanim doszło do bezpośredniego starcia.

Zasady taktyczne

- **Pokrycie obszarowe:** Łucznicy prowadzili ostrzał na obszar, nie na konkretne cele. Salwa strzał wystrzelona pod kątem 45 stopni pokrywała prostokątny obszar terenu, przeszukując go niejako losowo. Zasada „ostrzału obszarowego” ma swój odpowiednik w eksploracji przestrzeni poszukiwań.

- **Dystans bezpieczeństwa:** Łucznicy trzymali się z dala od wroga. Efektywny zasięg łuku kompozytowego wynosił 150–300 metrów, co pozwalało na obserwację i ostrzał bez narażania się na kontratak.
- **Wsparcie ogniowe:** Łucznicy osłabiali wroga przed szarżą kawalerii lub natarciem piechoty. Ich zadaniem było przygotowanie terenu, nie samodzielne rozstrzyganie bitwy.

Przełożenie na algorytm

Łucznicy w algorytmie to agenci o wysokim współczynniku eksploracji – przeszukują szerokie obszary przestrzeni rozwiązań, identyfikując obiecujące regiony do późniejszej eksploatacji przez cięższe jednostki. Niska eksploatacja i wysoki zasięg wpływu odzwierciedlają historyczną rolę łuczników jako sił rozpoznawczych.

3.2.6 Rydwany bojowe – szybkość i uderzenie

Rydwany wojenne to jedne z najwcześniejszych pojazdów bojowych w historii. Egipcjanie, Hetyci, Asyryjczycy i Chińczycy łączyli w nich mobilność kawalerii z siłą ognia łuczników. Bitwa pod Kadesz (ok. 1274 p.n.e.) między Ramzesem II a Hetytami jest jednym z najlepiej udokumentowanych przykładów użycia rydwanów na masową skalę [44].

Charakterystyka bojowa

- **Platforma mobilna:** Rydwan dawał stabilną platformę dla łucznika lub włócznika. Dwa konie zaprzężone do lekkiego wozu mogły osiągnąć prędkość do 35 km/h na otwartym terenie.
- **Ograniczenia terenowe:** Rydwany wymagały równego, twardego terenu. Na polach uprawnych, w lasach czy w górach były bezużyteczne.

Przełożenie na algorytm

W algorytmie rydwany to agenci o umiarkowanej prędkości i zrównoważonym profilu eksploracji/eksploatacji. Ich efektywność zależy od charakterystyki przestrzeni poszukiwań – podobnie jak rydwany historyczne dominowały na otwartych równinach, ale zawodziły w trudnym terenie.

3.2.7 Słonie bojowe – siła przełamania

Słonie bojowe to najbardziej spektakularna broń starożytności. Armie indyjskie, kartagińskie, hellenistyczne i ptolemejskie używały ich jako żywych czołgów – jednostek zdolnych do przełamania każdej formacji pieszej. Hannibal przeprowadził 37 słoni przez Alpy w 218 r. p.n.e. [50], a Pyrrus z Epiru rozbił nimi rzymskie legiony pod Herakleą (280 p.n.e.) [95].

Rola taktyczna

- **Przełamanie formacji:** Słoń bojowy o masie 4–6 ton, biegnący z prędkością 25 km/h, generował pęd zdolny do rozerwania nawet najgęstszej falangi. Żaden piechur nie mógł fizycznie powstrzymać takiego uderzenia.
- **Rozpraszanie przestrzeni:** Słonie wymuszały na przeciwniku rozluźnienie formacji – żołnierze instynktownie ustępowali z drogi, tworząc luki dla następujących za słoniami jednostek.
- **Wpływ na otoczenie:** Obecność słoni zmieniała zachowanie okolicznych jednostek – wrogich (panika, ucieczka) i sojuszniczych (wzrost morale). Zasięg tego wpływu był większy niż u jakiejkolwiek innej jednostki.

Przełożenie na algorytm

W algorytmie słonie to agenci o najwyższym współczynniku eksploracji i wpływu – przeszukują nowe obszary przestrzeni rozwiązań i zmieniają zachowanie okolicznych agentów. Wysoki parametr wpływu odzwierciedla historyczną zdolność słoni do oddziaływania na wszystkie jednostki w otoczeniu. Niska prędkość przy wysokiej sile przełamania oznacza powolne, lecz głębokie przeszukiwanie.

3.3 FORMACJE

Oprócz typu jednostki, o skuteczności decydowała formacja – sposób, w jaki jednostki utrzymują strukturę podczas bitwy. W proponowanym algorytmie zaimplementowano sześć typów formacji, z których każdy ma historyczne uzasadnienie.

Formacja Falangi

Gęsta, prostokątna masa ciężko uzbrojonych żołnierzy z włóczniami lub pikami [24, 53]. Grecka falanga hoplitów (8–12 szeregów, odstępy poniżej metra) maksymalizowała front ataku przy zachowaniu głębokości absorbującej straty. Filip II Macedoński rozwinął ją w falangę macedońską (16 szeregów, *sarissy* o długości do 18 stóp) [95, 98], poświęcając indywidualną manewrowość na rzecz przytłaczającego zasięgu. Falanga dominowała pod Maratonem (490 p.n.e.) [71] i Cheroneą (338 p.n.e.) [95], lecz jej sztywność ujawniła się pod Kynoskefalaj (197 p.n.e.), gdzie nierówny teren rozbił spójność formacji, a rzymskie manipuły wykorzystały luki [50]. W algorytmie falanga to formacja o najwyższej zwartości, koncentrująca agentów w minimalnym obszarze wokół najlepszego rozwiązania – odpowiada czystej eksploatacji.

Formacja klinowa

Formacja trójkątna, stosowana przede wszystkim przez kawalerię. Macedońska kawaleria towarzyszy (*hetairoi*) pod wodzą Aleksandra Wielkiego regularnie

używała klina do przebijania formacji wroga [3]. Wąski czubek klina koncentrował siłę uderzenia w jednym punkcie, a rozszerzające się boki zabezpieczały flanki. Pod Hydaspes (326 p.n.e.) Aleksander zastosował klin kawalerii przeciw słoniom Porusa – uderzenie w wąskim punkcie rozdzieliło linię słoni, uniemożliwiając ich koordynację. W algorytmie klin to formacja o wysokiej zwartości, koncentrująca agentów w kierunku najlepszego znanego rozwiązania.

Formacja liniowa

Najprostsza formacja – szereg jednostek rozstawionych na jednej linii. Stosowana przez lekką piechotę i łuczników, maksymalizowała front ataku kosztem głębokości. Rzymski *triplex acies* (trzy linie) [50] rozwijał tę koncepcję – trzy kolejne linie piechoty dawały głębię strategiczną i możliwość rotacji zmęczonych jednostek. W algorytmie formacja liniowa oznacza umiarkowane rozproszenie agentów i równomierny przegląd regionu przestrzeni poszukiwań.

Formacja łuku

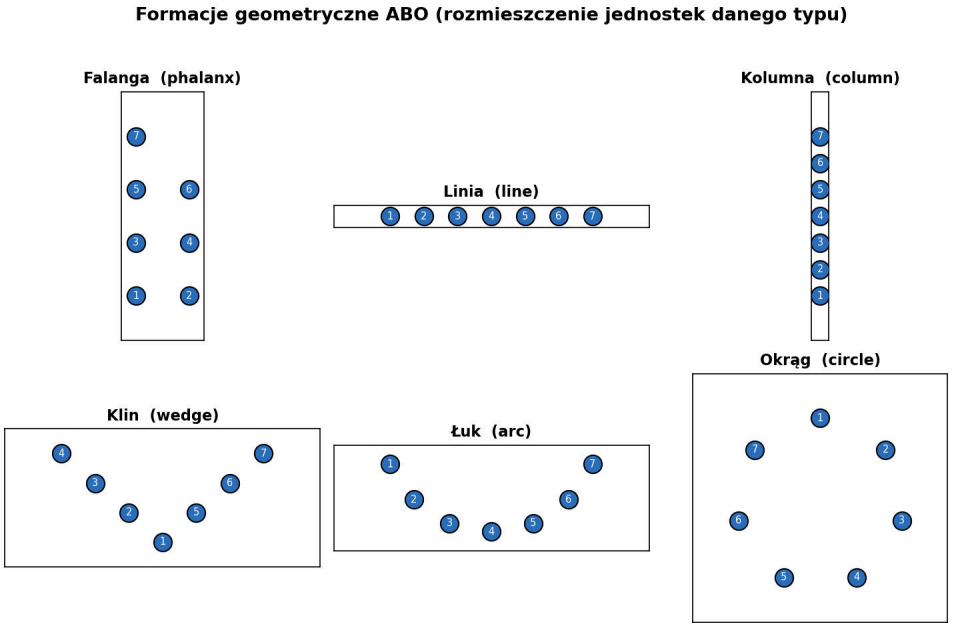
Wklęsła lub wypukła linia, stosowana przez łuczników do maksymalizacji pokrycia ogniowego. Łucznicy perscy i asyryjscy ustawiali się w łuku wypukłym, kierując strzały na centralną masę wroga. Pod Carrhae (53 p.n.e.) [43] partyjscy konni łucznicy utworzyli mobilne łuki wokół legionów Krassusa, prowadząc nieprzerwany ostrzał ze wszystkich kierunków – klasyczny przykład łuku jako formacji dystansowej. W algorytmie ta formacja rozkłada agentów wzdłuż krzywej, co daje szeroki, ale ukierunkowany przegląd przestrzeni.

Formacja kolumny

Głęboka, wąska formacja stosowana podczas marszów i przy forsowaniu wąskich przejść. Kolumna dawała maksymalną ochronę podczas przemieszczania się przez wrogie terytorium. Słonie bojowe często maszerowały w kolumnie, gdyż ich masa czyniła ją praktycznie nie do zatrzymania na wąskim froncie. W algorytmie formacja kolumny oznacza wysoką zwartość agentów, skoncentrowanych w wąskim paśmie przestrzeni poszukiwań.

Formacja okrężna

Formacja obronna, w której żołnierze tworzyli krąg z bronią skierowaną na zewnątrz w przypadku defensywy lub wewnątrz po zamknięciu oblężenia w przypadku ofensywy. Rzymskie legiony używały *orbis* jako ostatecznej formacji obronnej – w Lesie Teutoburskim (9 n.e.) [30] legiony Warusa próbowały uformować *orbis* podczas zasadzki germańskiej, aby przetrwać atak ze wszystkich stron. W algorytmie formacja okrężna oznacza równomierne rozmieszczenie agentów wokół centralnego punktu, co sprzyja lokalnemu przeszukiwaniu otoczenia.



Rys. 1. Sześć formacji geometrycznych ABO – rozmieszczenie jednostek danego typu względem środka formacji (wizualizacja wygenerowana bezpośrednio z implementacji `core/formations.py`). Każdemu typowi jednostek przypisano domyślną formację: ciężka piechota – falanga, lekka piechota oraz rydwany – linia, łucznicy – łuk, kawaleria – klin, słonie bojowe – kolumna.

3.4 TAKTYKI BITEWNE

Poza formacjami, starożytni dowódcy stosowali złożone taktyki – skoordynowane plany działania obejmujące wiele formacji i typów jednostek jednocześnie [95, 45]. W algorytmie zaimplementowano sześć taktyk, z których każda ma historyczne korzenie.

Falanga

Opisana szczegółowo w poprzedniej sekcji formacja falangi polegała na powolnym, zdyscyplinowanym natarciu gęstej formacji piechoty ciężkiej. W algorytmie jest to również strategia eksploatacyjna – koncentracja agentów przy najlepszym rozwiązaniu przy ograniczonej eksploracji. Parametry eksploracji maleją, a eksploatacja rośnie proporcjonalnie do postępu optymalizacji.

Szyk ukośny

Taktykę opracował tebański generał Epaminondas i zastosował ją pod Leuktrą (371 p.n.e.) [53]. Polega na asymetrycznym rozmieszczeniu sił – koncentracji przeważających sił na jednym skrzydle przy wstrzymaniu pozostałych. Epaminondas

zgromadził 50 szeregów na lewym skrzydle (zamiast typowych 8–12), miażdżąc elitarnych Spartan na prawym skrzydle, zanim słabsze centrum i prawe skrzydło zdążyły zostać zaangażowane.

Fryderyk Wielki udoskonalił tę taktykę w XVIII wieku, czyniąc ją podstawą pruskiej sztuki wojennej. Zasada jest prosta: zamiast rozpraszać siły równomiernie, koncentruje się je w punkcie decydującym.

W algorytmie szyk ukośny oznacza koncentrację zasobów obliczeniowych w jednym obiecującym regionie przestrzeni, przy minimalnej obecności w pozostałych obszarach. W implementacji algorytmicznej (Rozdział 4) asymetrię tę realizuje podział populacji na trzy grupy: silne skrzydło (agresywna eksploatacja), centrum (utrzymanie pozycji) i słabe skrzydło (eksploracja peryferyjna).

Przebicie centrum

Skoncentrowane uderzenie w centrum formacji wroga w celu przełamania jej na dwoje. Aleksander Wielki zastosował tę taktykę pod Gaugamelą [3, 43] – jego kawaleria towarzyszy uderzyła w lukę powstałą w centrum perskim, docierając bezpośrednio do Dariusza III i zmuszając go do ucieczki.

Istota tej taktyki to identyfikacja słabego punktu i skoncentrowanie w nim maksymalnej siły. W algorytmie przekłada się to na głębokie przeszukiwanie wąskiego regionu – agenci koncentrują się w jednym punkcie przestrzeni, maksymalizując eksploatację kosztem pokrycia.

Oskrzydlenie

Atak na boki formacji wroga, gdzie obrona jest najsłabsza. Kawaleria była najczęściej wykorzystywana do manewrów oskrzydlających ze względu na mobilność. Aleksander Wielki systematycznie oskrzydlał prawym skrzydłem kawalerii, podczas gdy falanga wiązała centrum wroga [3].

W algorytmie oskrzydlenie to strategia eksploracyjna: szybkie jednostki (kawaleria) poszukują rozwiązań w obszarach odległych od bieżącego optimum, podczas gdy wolniejsze jednostki eksploatują znane regiony.

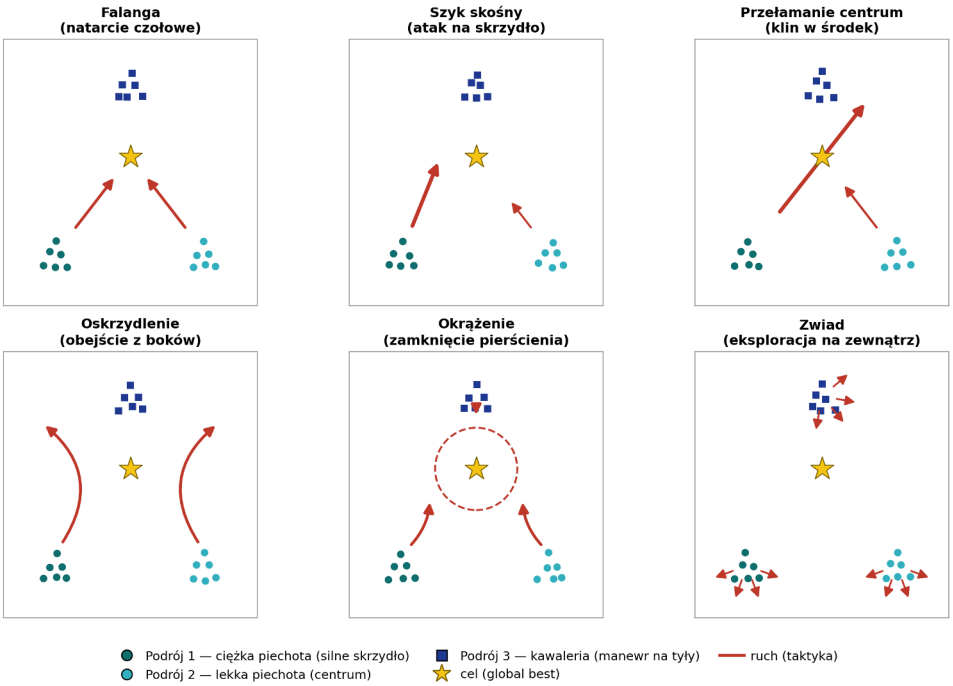
Okrążenie

Taktyka okrążenia polega na atakowaniu wroga ze wszystkich stron jednocześnie, odcinając mu drogi odwrotu i wymuszając kapitulację lub zniszczenie. Klasycznym przykładem jest podwójne okrążenie pod Kannami (216 p.n.e.) [50, 88], gdzie Hannibal celowo cofnął swoje centrum, wpuścił legiony rzymskie w pułapkę, a następnie zamknął okrążenie kawalerią na flankach i tyłach – niszcząc armię znacznie liczniejszą od własnej.

Zasadą okrążenia jest równomierne rozłożenie sił wokół celu, w przeciwieństwie do oskrzydlenia (atak na jedną flankę) czy przebicia centrum (koncentracja w jednym punkcie). Wymaga to dobrej koordynacji między wszystkimi skrzydłami

i odpowiedniej proporcji sił: siły muszą być wystarczająco liczne, by zamknąć pierścień, ale wystarczająco zdyscyplinowane, by utrzymać spójność.

W algorytmie taktyka okrążenia to strategia zrównoważona – agenci rozmieszczają się równomiernie wokół najlepszego znanego rozwiązania, zapewniając pokrycie ze wszystkich kierunków. Łączy cechy eksploracji (przeszukiwanie z różnych stron) i eksploatacji (koncentracja wokół obiecującego regionu).



Rys. 2. Sześć taktyk bitewnych ABO – kierunek manewru jednostek względem najlepszego znanego rozwiązania (★): natarcie czołowe (falanga), atak na skrzydło (szyk skośny), klin w środek (przebicie centrum), obejście z boków (oskrzydlenie), zamknięcie pierścienia (okrążenie) oraz eksploracja na zewnątrz (zwiad).

3.5 MECHANIZMY DOWODZENIA I KOORDYNACJI

Poza jednostkami, formacjami i taktykami, starożytne armie opierały się na trzech dodatkowych mechanizmach, które przełożono na struktury algorytmiczne.

3.5.1 System honoru i morale

Morale decydowało o wyniku starożytnych bitew. Większość bitew nie kończyła się fizycznym zniszczeniem armii, lecz załamaniem i ucieczką jednej ze stron. Historyk Ardant du Picq (XIX w.) oszacował, że w starożytnych bitwach straty

zadawane były głównie podczas pościgu za uciekającym wrogiem, nie podczas samej walki [30].

Starożytne armie stosowały rozbudowane systemy motywacyjne:

- **System nagród i awansów:** Rzymska armia stosowała formalne odznaczenia bojowe [50] – *corona civica* (wieniec obywatelski) za uratowanie życia współobywatela, *corona muralis* za pierwsze wdarcie się na mury wroga. Wyróżnieni żołnierze otrzymywali wyższy żołd, lepszą pozycję w szyku i autorytet wśród towarzyszy.

- **System kar i degradacji:** Tchórzostwo karano degradacją, publicznym upokorzeniem, a w skrajnych przypadkach *decimatio* – losowym straceniem co dziesiątego żołnierza w zbuntowanym oddziale.

- **Rola bohaterów:** Wybitni wojownicy walczący w pierwszym szeregu – podnosili morale całej armii swoją obecnością. Żołnierze orientowali się na bohaterów, naśladując ich zachowanie. Archetypowym przykładem jest Achilles – choć postać literacka, wiernie oddaje rzeczywisty fenomen wpływu herosa na spójność i determinację oddziału.

W algorytmie ABO system honorów odzwierciedla te mechanizmy: jednostki o najlepszych wynikach otrzymują status bohatera (zwiększony wpływ na otoczenie), a jednostki o słabych wynikach (kara za dezerterstwo) mogą zostać „zdegradowane" i zastąpione nowym agentem w nowej pozycji. Parametr zanikania honoru odpowiada procesowi, w którym dawne zasługi tracą na znaczeniu wobec bieżących wyników.

3.5.2 Rozpoznanie i wywiad wojskowy

Starożytne armie przeznaczały duże zasoby na systemy rozpoznania. Perski system „królewskiej drogi" i konnych posłańców pozwalał na przekazywanie informacji na odległość ponad 2500 km w ciągu tygodnia [56]. Rzymianie utrzymywali sieć *speculatores* (szpiegów) i *exploratores* (zwiadowców), którzy systematycznie zbierali informacje o terenie i wrogu.

Główne elementy starożytnego wywiadu wojskowego:

- **Dyskretyzacja terenu:** Dowódcy dzielili pole bitwy na sektory, przydzielając odpowiedzialność za obserwację każdego sektora konkretnym jednostkom. Ta dyskretyzacja jest bezpośrednią inspiracją dla siatkowej struktury modułu rozpoznania w ABO.

- **Estymacja sił wroga:** Na podstawie śladów marszowych, ognisk obozowych i relacji zwiadowców dowódcy szacowali liczebność i rozmieszczenie wroga. W algorytmie odpowiada to estymacji gradientów funkcji celu.

- **Identyfikacja szans:** Zwiadowcy raportowali nie tylko o zagrożeniach, ale też o szansach – niezabezpieczonych brodach, opuszczonych pozycjach, źródłach zaopatrzenia. W algorytmie przekłada się to na identyfikację obiecujących regionów przestrzeni poszukiwań.

W algorytmie taktyka zwiadu to faza eksploracyjna, w której wszystkie jednostki tymczasowo zwiększają swój zasięg przeszukiwania, zbierając informacje o topografii przestrzeni rozwiązań. System rozpoznania (*Reconnaissance Intelligence*) jest bezpośrednim przełożeniem historycznych praktyk zwiadowczych na mapowanie przestrzeni poszukiwań.

3.5.3 Hierarchia dowodzenia i adaptacyjne podejmowanie decyzji

Dowodzenie było najważniejszym elementem starożytnej sztuki wojennej. Wielcy dowódcy – Hannibal, Aleksander, Scypion Afrykański Starszy, Juliusz Cezar [43] – wyróżniali się zdolnością do adaptacyjnego podejmowania decyzji: odczytywania przebiegu bitwy i dostosowywania taktyki do zmieniających się warunków.

Hannibal pod Kannami (216 p.n.e.) [88] pokazał tę zdolność w sposób mistrzowski: obserwując natarcie rzymskiego centrum, zdecydował o kontrolowanym wycofaniu własnego centrum, jednoczesnym zamknięciu flanek i wysłaniu kawalerii na tyły – adaptował plan do rozwoju sytuacji w czasie rzeczywistym.

Zasady starożytnego dowodzenia:

- **Obserwacja stanu bitwy:** Dowódca nieustannie oceniał sytuację – postępy własnych sił, reakcje wroga, warunki terenowe. W module Commander AI ta ciągła obserwacja przekłada się na enkodowanie stanu.

- **Wybór taktyki:** Na podstawie obserwacji dowódca wybierał taktykę z dostępnego repertuaru. Doświadczony dowódca uczył się z wcześniejszych bitew, które taktyki sprawdzają się w jakich warunkach. Ta nauka z doświadczenia jest bezpośrednią inspiracją dla mechanizmu Q-learning w Commander AI.

- **Delegacja i autonomia:** Dowódcy niższego szczebla mieli autonomię w podejmowaniu decyzji taktycznych. Rzymski system centurionów zapewniał, że nawet po utracie naczelnego wodza armia mogła kontynuować walkę w sposób zdecentralizowany.

W algorytmie ABO moduł Commander AI formalizuje te zasady: obserwuje stan optymalizacji (postęp zbieżności ku optimum, trend poprawy), wybiera taktykę z repertuaru i uczy się na podstawie wyników swoich decyzji, gromadząc doświadczenie w tablicy Q-wartości.

3.5.4 Konsolidacja zdobytego terenu

Sukces taktyczny – przełamanie linii wroga, zajęcie wzgórza, zwycięstwo w starciu czołowym – nie kończył operacji wojskowej. W rzymskiej doktrynie wojskowej każda zdobycz wymagała natychmiastowej *consolidatio castrorum*: systematycznego umocnienia pozycji, zanim przeciwnik zdąży na nią kontratakować lub zanim zmierzch zakończy dzień bitewny. Termin pochodzi od łacińskiego *consolidare* – „uczynić twardym, scementować, ustabilizować” [111] – i obejmował kilka równoczesnych działań prowadzonych przez wyspecjalizowane oddziały (*fabri* – inżynierowie wojskowi, *antesignani* – żołnierze wybrani spośród legionu do zadań specjalnych).

Po zajęciu pozycji legion natychmiast wytyczał rzut obozu (*castra*) w standardowym schemacie prostokątnym z czterema bramami zgodnymi z kierunkami świata. *Fabri* kopali rów (*fossa*) głęboki na 1,2–2,7 metra i sypali z wybranego gruntu wał (*agger*) zwieńczony palisadą z kołków przenoszonych w marszu (*sudes* lub *pila muralia*) [50]. Każdy żołnierz nosił po dwa takie kołki, co pozwalało wznieść obóz dla pełnego legionu w ciągu 3–4 godzin. Polibiusz odnotował, że Rzymianie poświęcali nie mniej uwagi codziennemu umacnianiu pozycji niż samej walce – to ta dyscyplina pozwoliła im przetrwać klęski jak Trazymen czy Kanny i ostatecznie wygrać każdą prowadzoną wojnę [88].

Równocześnie z fortyfikacją prowadzono *exploratio per cardines* – systematyczne sondowanie terenu we wszystkich czterech kierunkach (*cardines*: północ, południe, wschód, zachód) przez lekkie oddziały (*velites*, później *exploratores*), wysyłane parami w odległości kilkuset kroków od obozu. Celem nie była już ofensywa, lecz precyzyjne ustalenie granic bezpiecznej strefy, identyfikacja niezauważonych wcześniej zagrożeń (resztki wroga w lasach, źródła wody pod kontrolą lokalnych plemion) oraz domknięcie wszystkich ognisk oporu w bezpośrednim sąsiedztwie. Tam, gdzie zwiad raportował poprawę pozycji (np. lepszy punkt obserwacyjny o kilkadziesiąt kroków dalej), kohorta przesuwała się i powtarzała procedurę – aż do osiągnięcia stabilnej granicy, której dalsze rozszerzanie nie przynosiło już korzyści.

Doktryna ta przetrwała upadek Rzymu i wróciła do europejskiej sztuki wojennej w epoce napoleońskiej jako *exploitation phase*: faza wykorzystania zwycięstwa, w której zwycięska armia nie pościga wroga w sposób chaotyczny, lecz metodycznie konsoliduje zdobyte pozycje, sondując każdy kierunek pod kątem dalszego marszu i wycofując się tam, gdzie napotka opór [43]. W obu doktrynach – starożytnej i nowożytnej – konsolidacja jest fazą *precyzyjną i lokalną*, prowadzoną wokół już osiągniętego punktu, ostrożnie testującą każdy kierunek, a nie ofensywą obejmującą całe pole operacji.

W algorytmie ABO ta zasada przekłada się na manewr konsolidacji opisany szczegółowo w Dodatku 7.4 (Algorytm 19): wyspecjalizowany operator uruchamiany dwukrotnie w trakcie kampanii (przy stagnacji głównych sił oraz na końcu ostatniej iteracji) sonduje teren wokół najlepszej znalezionej pozycji $\mathbf{x}^*$ kierunek

po kierunku, ekspanduje krok tam, gdzie wykryje poprawę, kontraktuje tam, gdzie napotka opór, i zatrzymuje się, gdy granica dalszego marszu zostanie osiągnięta. Bezpośrednim odpowiednikiem fortyfikacji *agger* jest precyzyjne dopięcie resztkowej różnicy między „okolicą minimum" a samym minimum – mechanizm ten najsilniej działa na funkcjach unimodalnych o ostrym dziobie minimum (np. ChungReynolds, Cigar, NeedleEye), gdzie sama armia ABO znajduje właściwy basen atrakcji, ale nie schodzi do precyzji maszynowej bez wsparcia inżynieryjnego.

Dotychczas przedstawiono poszczególne elementy starożytnej sztuki wojennej w izolacji – typy jednostek, formacje, taktyki, systemy honoru, rozpoznania i dowodzenia. Skuteczność armii zależała jednak nie od jakości poszczególnych komponentów, lecz od ich integracji w spójną strategię. Kolejna sekcja opisuje, jak te elementy łączą się w całość.

3.6 STRATEGIE – INTEGRACJA ELEMENTÓW

Opisane wcześniej typy jednostek, formacje, taktyki i mechanizmy koordynacji trzeba jeszcze zintegrować w spójną strategię. Strategia określa sposób rozmieszczenia podrojów na polu bitwy oraz dobór formacji i taktyk w zależności od etapu przeszukiwania i charakterystyki przestrzeni rozwiązań.

W historycznych bitwach strategia była nadrzędnym planem działania, uwzględniającym zarówno dyspozycję sił własnych, jak i przewidywane ruchy przeciwnika. W proponowanym algorytmie strategia pełni tę samą funkcję – decyduje o globalnym rozkładzie zasobów obliczeniowych.

Przyjęte strategie inspirowane są historycznymi wzorcami dowodzenia: konserwatywne podejścia z równomiernym rozłożeniem sił, strategie koncentracji uderzenia w newralgicznych punktach, adaptacyjne schematy reagujące na bieżące wyniki przeszukiwania. Szczegółowy opis poszczególnych strategii wraz z ich mapowaniem na mechanizmy algorytmiczne przedstawiono w kolejnych rozdziałach; poniżej omówiono podstawowe zasady, które pomogły w implementacji, oraz ich odniesienia historyczne.

Starożytne bitwy wymagały skoordynowanego działania różnych rodzajów wojsk – piechoty, kawalerii, łuczników, jednostek oblężniczych. Każda z tych formacji miała odmienną mobilność, siłę rażenia i odporność, co wymagało precyzyjnej synchronizacji zarówno w czasie, jak i w przestrzeni. Skuteczność operacji połączonych sił zależała nie tyle od jakości poszczególnych jednostek, ile od umiejętności ich wzajemnego wspierania.

W warunkach ograniczonych możliwości komunikacyjnych starożytni dowódcy wypracowali szereg mechanizmów umożliwiających koordynację działań na polu bitwy:

- **Wstępnie zaplanowane sygnały:** Przed bitwą ustalano zestaw sygnałów dźwiękowych (rogi, bębny, trąby) i wizualnych (sztandary, pochodnie, dym),

które inicjowały określone manewry. System pozwalał na jednoczesne przekazanie rozkazów wielu jednostkom, ale wymagał szczegółowego przygotowania i ograniczał reakcję na nieprzewidziane okoliczności.

- **Obserwacja wizualna:** Dowódcy nieustannie obserwowali pole bitwy, śledząc ruchy jednostek sojuszniczych i dostosowując własne działania do sytuacji. Skuteczność tego mechanizmu malała ze wzrostem skali starcia i przy ograniczonej widoczności.

- **Orientacja na jednostki elitarne:** Żołnierze w ferworze walki instynktownie kierowali wzrok ku najbardziej doświadczonym wojownikom. Częściowo wynikało to z pragnienia przetrwania – podążanie za skuteczniejszymi jednostkami zwiększało szanse na sukces – ale pełniło też funkcję spontanicznego mechanizmu koordynacyjnego, pozwalając na organizację działań nawet po załamaniu formalnej struktury dowodzenia.

- **Hierarchiczna struktura dowodzenia:** Armie dzielono na mniejsze pododdziały z wyznaczonymi dowódcami, co pozwalało na delegację odpowiedzialności i szybsze podejmowanie lokalnych decyzji bez angażowania naczelnego wodza.

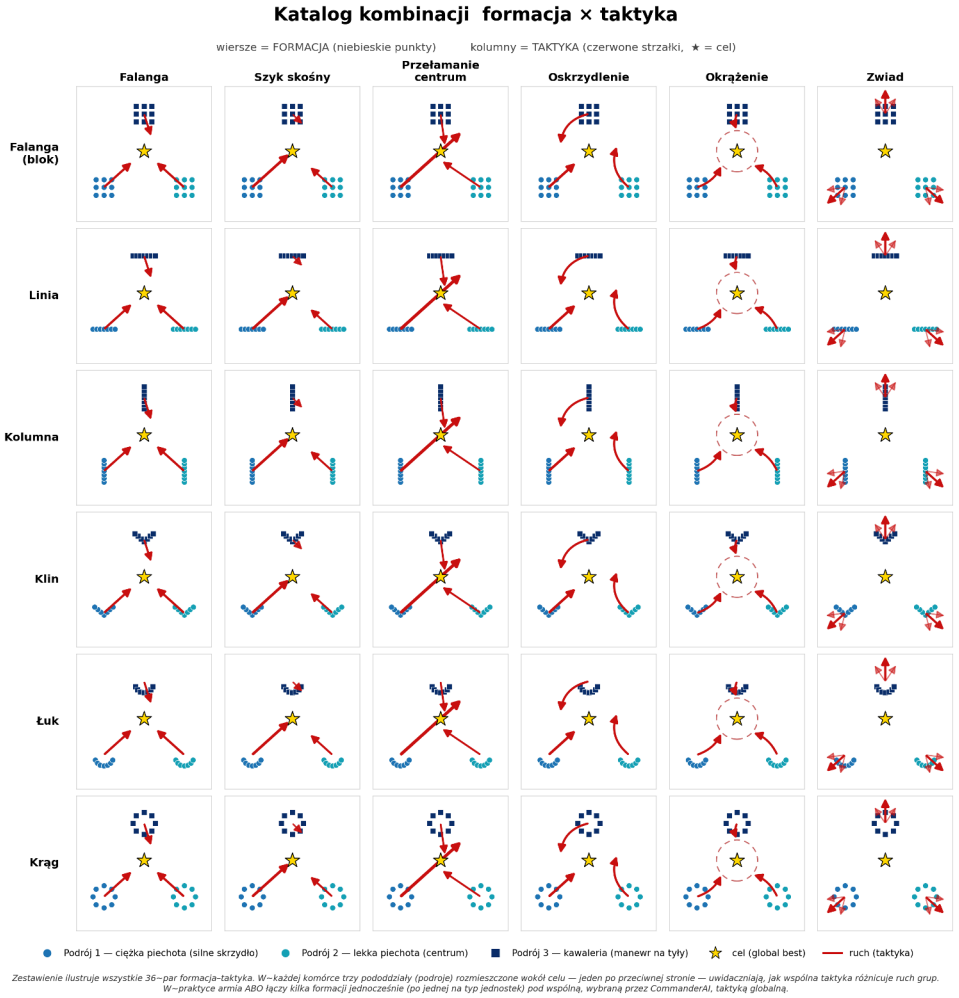
Ograniczone środki komunikacji na starożytnym polu walki – sygnały akustyczne, wskazówki wizualne, posłańcy piesi i konni – narzucały ograniczenia na złożoność manewrów. Paradoksalnie te ograniczenia wymuszały rozwiązania zdecentralizowane i adaptacyjne, które – jak pokazano w dalszej części pracy – są inspiracją dla mechanizmów koordynacji w proponowanym algorytmie. Przedstawiona poniżej taktyka ilustruje sposób takiej adaptacji.

3.6.1 Od taktyki do algorytmu – mapowanie pojęć

Przed przejściem do formalnego opisu algorytmu warto podsumować, w jaki sposób opisane powyżej elementy historyczne przekładają się na mechanizmy obliczeniowe. Poszczególnym typom jednostek przypisano odrębne operatory przeszukiwania: przeszukiwanie współrzędnościowe (piechota ciężka), mutację ewolucji różnicowej (piechota lekka), wielopunktowe próbkowanie (łucznicy), lot Lévy'ego (kawaleria), momentum Cauchy'ego (rydwany) oraz *basin hopping* (słonie bojowe). Pełne odwzorowanie jednostek na operatory wraz ze szczegółami formalnymi i parametrami przedstawia Rozdział 4 (Sekcja 4.2.1).

Dzięki temu zróżnicowaniu populacja jako całość równoważy eksplorację i eksploatację, co jest kluczowym wyzwaniem w optymalizacji metaheurystycznej.

Opisane typy jednostek, formacje, taktyki i mechanizmy koordynacji stanowią fundament historyczny algorytmu ABO. W kolejnym rozdziale przekształcono je w formalne rozwiązanie algorytmiczne.



Rys. 3. Katalog kombinacji *formacja* (wiersze; niebieskie punkty) × *taktyka* (kolumny; czerwone strzałki, ★ = cel). Zestawienie ilustruje wszystkie 36 par formacja–taktyka. W każdej komórce trzy pododdziały (podroje) rozmieszczono wokół celu – jeden po przeciwnej stronie – co uwidacznia, jak wspólna taktyka różnicuje ruch poszczególnych grup jednostek. W praktyce armia ABO łączy kilka formacji jednocześnie (po jednej na typ jednostek) pod wspólną, wybraną przez Commander AI, taktyką globalną.

4. PROJEKTOWANIE NOWYCH ALGORYTMÓW ROJOWYCH INSPIROWANYCH STRATEGIAMI BITEWNYMI

„Wojna kocha zmienność."
– Sun Tzu, Sztuka Wojny.

Poprzedni rozdział poprowadził przez pola bitew starożytności i pozwolił zidentyfikować uniwersalne zasady: koncentrację sił, koordynację jednostek, adaptację do terenu. Teraz nadszedł czas, by przetransponować te koncepcje z historii na algorytmy.

Ten rozdział przedstawia architekturę Ancient Battlefield Optimizer (ABO) – algorytmu rojowego [127, 37], w którym każda jednostka bitewna staje się agentem przeszukującym przestrzeń rozwiązań, każda taktyka przekształca się w operator modyfikujący pozycje agentów, a dowódca armii ewoluuje w moduł sterujący [110]. Rozdział rozpoczyna się od modułowej architektury jednostek (Sekcja 4.1), następnie przedstawia operatory optymalizacji inspirowane manewrami bitewnymi (Sekcja 4.2), a kończy się na mechanizmach integracji tworzących emergentną inteligencję zbiorową (Sekcja 4.3).

Tabela 3. Status implementacji komponentów algorytmu ABO

Komponent	**Opis**
6 typów jednostek	HeavyInfantry, LightInfantry, Archers, Cavalry, Chariots, WarElephants
6 taktyk bitewnych	Phalanx, Oblique, CenterPenetration, Flanking, Surrounding, Scouting
Formacje jednostek	6 typów formacji z parametrami zwartości i dyscypliny
System rozpoznania	Mapowanie przestrzeni, estymacja gradientów, poziomy zainteresowania
System honorów	Dynamiczny awans/karanie dezercji
Commander AI (Q-learning)	Adaptacyjna selekcja taktyk z uczeniem ze wzmocnieniem

4.1 ARCHITEKTURA MODUŁOWYCH JEDNOSTEK BITEWNYCH

Algorytm Ancient Battlefield Optimizer (ABO) opiera się na modułowej architekturze inspirowanej organizacją starożytnych armii [95, 24], w której zgodnie z podstawowymi założeniami algorytmów rojowych, autonomiczne jednostki o zróżnicowanych rolach współpracują, by osiągnąć wspólny cel [120, 40]. Każda jednostka reprezentuje rozwiązanie (agenta) w przestrzeni poszukiwań, a jej zachowanie zależy od typu jednostki, przypisanych parametrów oraz bieżącego stanu optymalizacji. Historyczne podstawy tej architektury – typy jednostek, ich role taktyczne, formacje oraz przykłady zastosowań z konkretnych bitew starożytnych – przedstawiono w Rozdziale 3.

4.1.1 Podstawy architektury modułowej

Koncepcja modułowości w algorytmach

Modułowość w ABO realizuje inżynierską zasadę *rozdzielenia odpowiedzialności* (*separation of concerns*) [46, 78]: w przeciwieństwie do monolitycznych algorytmów rojowych [18, 68], w których wszystkie agenty zachowują się identycznie, ABO wprowadza heterogeniczność inspirowaną strukturą wojskową. Opiera się ona na czterech zasadach: **enkapsulacji** (każdy typ jednostki ukrywa własne strategie ruchu za wspólnym interfejsem `Unit`), **polimorfizmie** (wszystkie typy dziedziczą z klasy bazowej `Unit`), **kompozycji nad dziedziczeniem** (zachowania składane z niezależnych modułów: ruch, ocena, honor, formacje, pamięć pbest) oraz **luźnym powiązaniu** (komunikacja przez zdefiniowane interfejsy: dzielenie najlepszego rozwiązania, wpływ bohaterów, rekomendacje zwiadu).

Matematycznie populację algorytmu można przedstawić jako:

$$\mathcal{P} = \{u_1, u_2, \ldots, u_N\} = \bigcup_{t \in \mathcal{T}} \mathcal{P}_t \tag{4.1}$$

gdzie $\mathcal{T}$ jest zbiorem 6 typów jednostek (HeavyInfantry, LightInfantry, Cavalry, Archers, Chariots, WarElephants), a $\mathcal{P}_t$ jest podzbiorem jednostek typu t.
Status *hero* lub *runagate* przyznawany jest dynamicznie na podstawie systemu honorów (jednostki o wysokim honorze $H \geq 6{,}0$ stają się bohaterami, a te o niskim honorze $H \leq -10{,}0$ zostają zdegradowane do statusu zbiega z pola walki). Asymetria tych progów – kara za słabe wyniki ($-10{,}0$) jest silniejsza niż nagroda za dobre ($+6{,}0$) – nawiązuje do zjawiska awersji do straty opisanego w psychologii behawioralnej [63]. Konkretne wartości progów dobrano empirycznie na zbiorze walidacyjnym (Sekcja 5).

W benchmarku eksperymentalnym (Rozdział 6) przyjęto populację $N = 40$ jednostek o następującym składzie: piechota ciężka – 8 (20%), piechota lekka – 8 (20%), łucznicy – 7 (17,5%), kawaleria – 8 (20%), rydwany – 5 (12,5%), słonie

bojowe – 4 (10%). Proporcje te odzwierciedlają historyczne składy armii, w których piechota stanowiła trzon (40% łącznie), kawaleria zapewniała mobilność, a jednostki specjalne (rydwany, słonie) pełniły rolę niszową.

Projektowanie interfejsów między modułami

Interfejs bazowy `Unit` definiuje zasady, które muszą spełniać wszystkie typy jednostek. Główne metody i atrybuty interfejsu obejmują:

Atrybuty stanu jednostki:

- $\mathbf{x} \in \mathbb{R}^d$ – pozycja w d-wymiarowej przestrzeni poszukiwań
- $f(\mathbf{x}) \in \mathbb{R}$ – wartość funkcji celu (fitness)
- $\mathbf{v} \in \mathbb{R}^d$ – wektor prędkości
- $\mathbf{x}_{\text{best}} \in \mathbb{R}^d$ – osobiste najlepsze rozwiązanie
- $H \in \mathbb{R}$ – wartość honoru (może być ujemna)
- status $\in$ {active, hero, runagate} – stan aktywności; *runagate* (zbieg) oznacza jednostkę, której honor spadł poniżej progu $-10{,}0$ – taka jednostka jest pomijana w aktualizacji pozycji, analogicznie do dezertera opuszczającego pole bitwy

Parametry behawioralne:

- $s \in \mathbb{R}^+$ – prędkość bazowa (speed)
- $\epsilon \in [0, \infty)$ – skłonność do eksploracji (exploration)
- $\xi \in [0, \infty)$ – skłonność do eksploatacji (exploitation)
- $\mu \in \mathbb{R}^+$ – siła pamięci (memory)
- $\iota \in \mathbb{R}^+$ – wpływ na inne jednostki (influence)

Parametry formacyjne:

- $\mathcal{F} \in$ {phalanx, line, column, wedge, arc, circle} – typ formacji
- $\tau \in [0{,}1]$ – zwartość formacji (tightness)
- $\delta \in [0{,}1]$ – dyscyplina formacyjna (discipline)

Główne metody interfejsu:

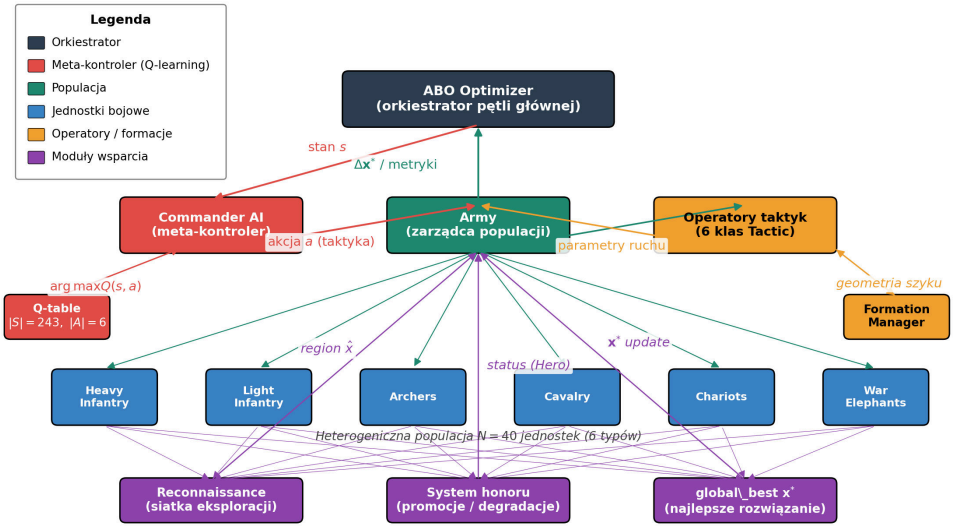
```
def update_position(new_position: np.ndarray):
    """Aktualizuje pozycję jednostki (z obcięciem do granic)."""

def update_fitness(objective_func: Callable) -> bool:
    """Ewaluuje funkcję celu; zwraca True jeśli poprawiono pbest."""

def update_honor(improved_global: bool, honor_decay: float):
    """Aktualizuje wartość honoru na podstawie wydajności."""
```

Ruch jednostki jest delegowany do obiektu `movement_strategy` przypisanego do danego typu (wzorzec Strategii – Tabela 88), a zarządzanie formacjami realizowane jest przez zewnętrzny `FormationManager`.

Interfejs komunikacyjny między modułami opiera się na **bezpośrednich wywołaniach metod** z armią (`Army`) jako centralnym koordynatorem propagacji zmian globalnego najlepszego rozwiązania, natomiast bieżąca zmiana taktyk i formacji realizowana jest przez **wzorzec Strategii** [46].

Zasady luźnego powiązania komponentów

Luźne powiązanie realizują trzy mechanizmy. **(1)** Klasa `Army` pełni rolę *fasady* [46] ukrywającej zarządzanie heterogeniczną populacją – optymalizator komunikuje się wyłącznie z nią, bez bezpośredniego dostępu do jednostek, co pozwala zmieniać ich implementację bez wpływu na logikę optymalizatora. **(2)** Zamiast modyfikować parametry jednostek wprost, optymalizator aplikuje obiekt `Tactic` (metoda `apply(units, global_best, iteration, max_iterations)`), dzięki czemu taktyki można komponować z góry lub wymieniać w czasie pracy bez zmian w kodzie jednostek. **(3)** Komponenty komunikują się przez bezpośrednie wywołania metod, a `Army` koordynuje aktualizację globalnego optimum (`update_global_best()`), propagację wpływu bohaterów (`share_knowledge()`), zwiad (`reconnaissance.update_with_position()`) oraz reorganizację formacji (`formation_manager.organize_units()`).



Rys. 4. Zintegrowany diagram architektury CommanderABO. Centralna oś pionowa (Optimizer → Army → 6 typów jednostek) odpowiada za pętlę optymalizacji. Po lewej: Commander AI z tablicą Q (243 stany × 6 akcji). Po prawej: warstwa taktyczna (6 klas `Tactic`) wraz z FormationManagerem zarządzającym geometrią szyku. Na dole: moduły wsparcia dostarczające informacji o stanie krajobrazu (Rekonesans, system honoru, global best). Strzałki przerywane na dole oznaczają zbiorczy wkład jednostek do modułów wsparcia.

4.1.2 Struktura i funkcje modułów

Architektura wewnętrzna modułów

Rysunek 4 przedstawia zintegrowany widok architektury algorytmu CommanderABO, pokazujący zależności między orkiestratorem pętli, meta-kontrolerem opartym na Q-learningu, populacją jednostek bojowych oraz modułami wsparcia (Rekonesans, System honoru, global best). Strzałki opisują przepływ informacji: stan krajobrazu trafia z Army do Commander AI, akcja (wybrana taktyka) wraca z powrotem, a operatory taktyk modyfikują parametry ruchu wspólne dla wszystkich jednostek danego typu.

Architektura ABO składa się z pięciu głównych modułów, z których każdy realizuje konkretne aspekty metafory bitewnej:

Moduł 1: Jednostki (abo_optimizer/core/units.py)

Odpowiedzialny za definicje wszystkich typów jednostek bitewnych oraz ich bazową mechanikę ruchu i oceny. Moduł implementuje 6 specjalizowanych klas dziedziczących z `Unit`:

Tabela 4. Parametry behawioralne poszczególnych typów jednostek (6 klas bazowych)

Typ	Speed	Expl	Expt	Mem	Infl	Domyślna Formacja
HeavyInfantry	0,7	0,6	1,5	2,3	1,2	phalanx
LightInfantry	1,2	1,0	1,0	2,0	1,0	line
Archers	1,0	1,5	0,7	1,9	0,8	arc
Cavalry	1,8	1,6	0,9	1,7	1,1	wedge
Chariots	1,6	1,2	1,2	1,9	1,3	line
WarElephants	0,9	1,8	1,4	1,6	1,8	column

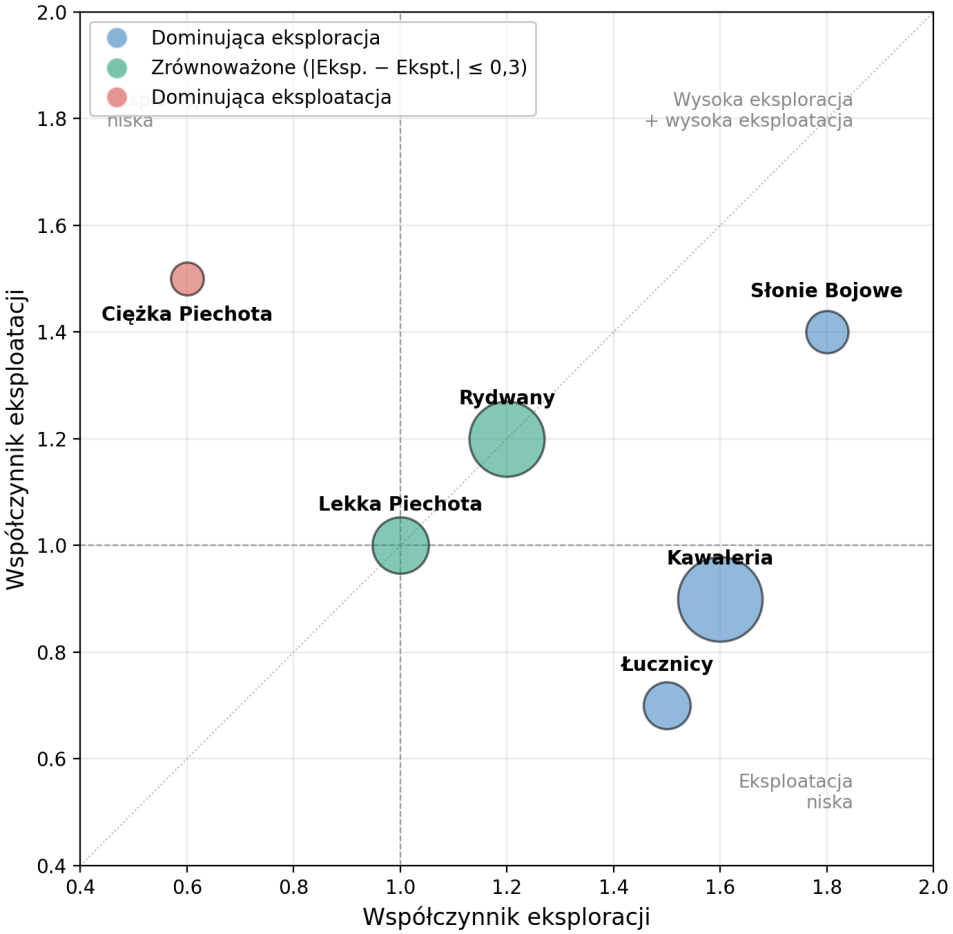
Jak już wspomniano wcześniej, system honorów (hero status) oraz zwiad (scouting) nie są oddzielnymi typami jednostek, lecz mechanizmami stosowanymi do istniejących jednostek. Każda jednostka może uzyskać status bohatera poprzez wysoką wydajność (zarządzane przez moduł Army), a zwiadowcy są częścią systemu ReconnaissanceIntelligence i również mogą być różnymi typami jednostek.

Tabela 4 zestawia pięć parametrów behawioralnych wszystkich typów jednostek, a Rysunek 5 pokazuje ich pozycjonowanie w dwuwymiarowej przestrzeni eksploracja–eksploatacja – kompromis ten stanowi podstawowy wymiar projektowania metaheurystyk [36, 126]. Klasyfikacja na trzy kategorie opiera się na progu |Eksploracja – Eksploatacja| > 0,3 – typy spełniające ten warunek przypisywane są do kategorii dominującej (eksploracyjnej lub eksploatacyjnej), pozostałe traktowane są jako zrównoważone. Dobór parametrów behawioralnych oparto na analizie historycznych charakterystyk jednostek (por. Sekcja 3.2) – na przykład niska prędkość ciężkiej piechoty (0,7) odzwierciedla ograniczoną mobilność hoplitów obciążonych ponad 30 kg ekwipunku, a wysoka prędkość kawalerii (1,8) – jej zdolność do szybkich rajdów na skrzydłach. Dwa uzupełniające ujęcia tej samej parametryzacji – hierarchię ról oraz pełne profile radarowe – zamieszczono w Dodatku 7.4 (Rysunki 46 i 47).

Moduł 2: Taktyki (abo_optimizer/core/tactics.py)

Implementuje 6 głównych taktyk bitewnych jako niezależne klasy strategii:

- `PhalanxTactic` – zwarty, skoordynowany atak (wzrost eksploatacji)
- `ObliqueTactic` – szyk ukośny z asymetrią sił
- `CenterPenetrationTactic` – przełamanie centrum z koncentracją siły
- `FlankingTactic` – oskrzydlenie z szybkimi jednostkami
- `SurroundingTactic` – okrążenie z wszechstronnym atakiem
- `ScoutingTactic` – agresywna eksploracja z głęboką integracją z systemem rekonesansu. Formacje w trybie Scouting stosują maksymalne rozproszenie jednostek w celu pokrycia jak największego obszaru przestrzeni poszukiwań.



Rys. 5. Pozycjonowanie sześciu typów jednostek ABO w dwuwymiarowej przestrzeni współczynnik eksploracji × współczynnik eksploatacji. Rozmiar punktu jest proporcjonalny do kwadratu prędkości jednostki, kolor oznacza kategorię dominującą (niebieski – eksploracja, zielony – zrównoważone, czerwony – eksploatacja). Linie przerywane wyznaczają wartość referencyjną 1,0 dla obu osi.

Moduł 3: Formacje (abo_optimizer/core/formations.py)

Zarządza geometrycznym rozmieszczeniem jednostek w przestrzeni poszukiwań.

`FormationManager` implementuje 6 typów formacji, poniżej przedstawiono dwie z nich (pozostałe znajdują się w Dodatku):

$$\mathbf{x}_i^{\text{form}} = \mathbf{c} + \mathbf{o}_i(\mathcal{F}, \tau, N) \tag{4.2}$$

gdzie $\mathbf{c}$ jest centrum formacji, $\mathbf{o}_i$ jest przesunięciem jednostki i zależnym od typu formacji $\mathcal{F}$, zwartości τ oraz liczby jednostek N.

Dla formacji **phalanx** (prostokątna siatka):

$$\text{cols} = \max(1, \lfloor \sqrt{N} \rfloor) \tag{4.3}$$

$$\text{rows} = \lceil N/\text{cols} \rceil \tag{4.4}$$

$$o_{i,x} = (i \bmod \text{cols} - (\text{cols} - 1)/2) \times \sigma \tag{4.5}$$

$$o_{i,y} = (\lfloor i/\text{cols} \rfloor - (\text{rows} - 1)/2) \times \sigma \tag{4.6}$$

gdzie $\sigma = 0{,}2 \times (2{,}0 - \tau)$ jest współczynnikiem skali.

Dla formacji **wedge** (klin):

$$o_0 = [0, -\sigma \times 0{,}5] \quad \text{(lider)} \tag{4.7}$$

$$o_{i,\text{left}} = [-i \times \sigma \times 0{,}8, -\sigma \times 0{,}5 + i \times \sigma \times 0{,}7] \tag{4.8}$$

$$o_{i,\text{right}} = [i \times \sigma \times 0{,}8, -\sigma \times 0{,}5 + i \times \sigma \times 0{,}7] \tag{4.9}$$

Elastyczne przyciąganie formacyjne:

Każda jednostka ma przypisaną pozycję docelową w formacji i podlega elastycznemu przyciąganiu w jej kierunku – im większe odchylenie od wyznaczonego miejsca, tym silniejsze przyciąganie z powrotem do szyku. Siła tego przyciągania zależy od statusu jednostki: jednostki o niskim honorze przyciągane są mocniej, dzięki czemu trzymają się formacji ściślej, natomiast bohaterowie ($H \geq 6{,}0$) mają siłę przyciągania zmniejszoną o 30%, co daje im więcej swobody i pozwala na szerszą eksplorację otoczenia.

Formalnie, niech $\mathbf{t}_i$ oznacza pozycję docelową jednostki i w formacji, a $\mathbf{d}_i = \mathbf{t}_i - \mathbf{x}_i$ wektor przesunięcia. Próg aktywacji wzmocnionego przyciągania skaluje się z zasięgiem przeszukiwania i postępem optymalizacji:

$$\tau_i = r_i \cdot 0{,}15 \cdot \left(1{,}5 - 0{,}5 \cdot \frac{t}{T}\right) \tag{4.10}$$

gdzie r_i jest zasięgiem przeszukiwania jednostki i (zależnym od typu), t bieżącą iteracją, a T maksymalną liczbą iteracji.

Bazowa siła przyciągania wynika z dyscypliny formacyjnej δ_i (domyślnie 0,7):

$$\phi_i^{(\text{base})} = \begin{cases} 0{,}7 \cdot \delta_i \cdot 0{,}4 & \text{jeśli } H_i \geq 6{,}0 \text{ (bohater)} \\ \delta_i \cdot 0{,}4 & \text{w przeciwnym razie} \end{cases} \tag{4.11}$$

Gdy odległość $\|\mathbf{d}_i\|$ przekracza próg τ_i, siła rośnie proporcjonalnie (model elastycznej taśmy):

$$\phi_i = \begin{cases} \min\left(0{,}8,\ \phi_i^{(\text{base})} \cdot \frac{\|\mathbf{d}_i\|}{\tau_i}\right) & \text{jeśli } \|\mathbf{d}_i\| > \tau_i \\ \phi_i^{(\text{base})} & \text{w przeciwnym razie} \end{cases} \tag{4.12}$$

Aktualizacja pozycji:

$$\mathbf{x}_i \leftarrow \text{clip}(\mathbf{x}_i + \phi_i \cdot \mathbf{d}_i,\ \mathbf{lb},\ \mathbf{ub}) \tag{4.13}$$

Taktyka może nadpisać wartość δ_i parametrem `formation_pull` (np. taktyka *Flanking*: ciężka piechota 0,8, kawaleria 0,3). Pełny pseudokod: Algorytm 18 w Dodatku A.

Moduł 4: Rekonesans (abo_optimizer/core/reconnaissance.py)

Implementuje system wywiadowczy mapujący przestrzeń poszukiwań jako siatkę komórek o rozdzielczości r (domyślnie 10 komórek na wymiar). Każda komórka $(g_1, g_2, \ldots, g_d)$ przechowuje statystyki odwiedzin, fitness, ostatnie pozycje i lokalny gradient (szczegółowa struktura danych – Sekcja 4.1, Moduł 4).

Intuicyjnie system rekonesansu działa jak mapa harcerska: przestrzeń poszukiwań jest podzielona na kwadraty (komórki siatki), a w każdym z nich algorytm zapisuje, czy region okazał się interesujący (znaleziono w nim dobre rozwiązanie) czy niebezpieczny (wielokrotnie odwiedzany bez poprawy fitness). Jednostki korzystają z tej mapy analogicznie do zwiadowców – kierują się tam, gdzie mapa wskazuje obiecujące, lecz jeszcze niedostatecznie zbadane rejony.

Na poziomie koncepcyjnym ocenę regionów można ująć w trzech metrykach: poziomie zainteresowania (I_g), poziomie niebezpieczeństwa (D_g) oraz statusie eksploracji (E_g). Stanowią one *model interpretacyjny* – ich pełną postać podano poniżej, a różnice względem implementacji referencyjnej omówiono na końcu opisu modułu.

Poziom zainteresowania I_g regionu g wyraża atrakcyjność regionu na podstawie najlepszego znalezionego w nim fitness:

$$I_g = \frac{f_{\text{global\_worst}} - f_{\text{best}}(g)}{f_{\text{global\_worst}} - f_{\text{global\_best}} + \varepsilon} \tag{4.14}$$

gdzie $f_{\text{best}}(g)$ jest najlepszą wartością fitness zaobserwowaną w komórce g, $f_{\text{global\_best}}$ i $f_{\text{global\_worst}}$ to odpowiednio najlepsza i najgorsza wartość fitness w całej populacji, a $\varepsilon > 0$ zapobiega dzieleniu przez zero. Dla minimalizacji najlepsze regiony uzyskują wartości bliskie 1, a regiony słabsze wartości bliższe 0.

Poziom niebezpieczeństwa D_g regionu g wyraża ryzyko utknięcia w optimum lokalnym na podstawie gęstości odwiedzin i braku poprawy:

$$D_g = 1 - \exp\left(-\lambda \cdot \frac{n_{\text{visits}}(g)}{n_{\text{total}}} \cdot (1 - \Delta_g)\right) \tag{4.15}$$

gdzie $n_{\text{visits}}(g)$ to liczba odwiedzin komórki g, n_{total} to łączna liczba odwiedzin wszystkich komórek, $\Delta_g \in [0, 1]$ to znormalizowana poprawa fitness w ostatnich odwiedzinach, a $\lambda > 0$ to parametr skali (domyślnie $\lambda = 5$). Region o wysokim D_g jest często odwiedzany, lecz nie przynosi poprawy – wskazuje na potencjalne optimum lokalne.

Status eksploracji regionu g określa, czy region wymaga dalszego zbadania:

$$E_g = \begin{cases} \texttt{unexplored} & \text{jeśli } n_{\text{visits}}(g) = 0 \\ \texttt{promising} & \text{jeśli } I_g > \tau_I \text{ i } D_g < \tau_D \\ \texttt{exhausted} & \text{jeśli } D_g \geq \tau_D \\ \texttt{explored} & \text{w pozostałych przypadkach} \end{cases} \tag{4.16}$$

gdzie $\tau_I = 0{,}3$ i $\tau_D = 0{,}7$ to domyślne progi klasyfikacji regionów.

Relacja do implementacji. Wzory (4.14)–(4.16) stanowią model interpretacyjny ułatwiający zrozumienie zasady działania. Implementacja referencyjna nie wyznacza wprost wartości D_g ani dyskretnych statusów `unexplored/promising/exhausted/explored`. Zamiast tego dla każdego regionu obliczany jest złożony wskaźnik atrakcyjności

$$\text{score}(g) = 0{,}6 \cdot \text{fitness\_score}(g) + 0{,}3 \cdot \text{exploration\_score}(g) + 0{,}1 \cdot \text{gradient\_score}(g), \tag{4.17}$$

z karą za niedawne odwiedziny (`recent_visits`); na jego podstawie utrzymywana jest lista do 10 najlepszych regionów (`promising_regions`). Rekomendacja kierunkowa dla jednostek jest następnie składana z czterech komponentów – lokalnego gradientu, regionu elitarnego (wokół globalnego optimum), regionu obiecującego oraz składnika eksploracyjnego – ważonych parametrem eksploracja–eksploatacja (szczegóły w Sekcji 4.2.6).

Moduł 5: AI Komander (abo_optimizer/ai/commander_ai.py)

Realizuje hierarchiczny system dowodzenia wykorzystujący uczenie ze wzmocnieniem (Q-learning) [114, 104]. Commander AI obserwuje stan armii i adaptacyjnie wybiera taktyki oraz szablony formacji w celu maksymalizacji poprawy procesu optymalizacji.

Przestrzeń stanów $\mathcal{S}$ jest pięciowymiarowa:

$$s = (\text{progress}, \text{stagnation}, \text{momentum}, \text{diversity}, \text{hero\_ratio}) \tag{4.18}$$

Każdy wymiar jest dyskretyzowany do 3 poziomów, dając łącznie $|\mathcal{S}| = 3^5 = 243$ stanów. Szczegóły dyskretyzacji stanów podano w Sekcji 4.2.5.

Przestrzeń akcji $\mathcal{A}$:

$$a = \text{tactic\_index} \in \{0, 1, \ldots, 5\} \tag{4.19}$$

z 6 taktykami (Phalanx, Oblique Order, Center Penetration, Flanking, Surrounding, Scouting), dając $|\mathcal{A}| = 6$ akcji. Formacje są dobierane automatycznie na podstawie wybranej taktyki poprzez mapowanie `tactic_formation_map`.

Specyfikacja interfejsów komunikacyjnych

Komunikacja odbywa się trzema kanałami. **Army↔Optimizer**: optymalizator ustawia taktykę i szablon formacji (`set_tactic`, `set_formation_template`) oraz wyzwala aktualizację jednostek (`update_units`), a odczytuje globalne najlepsze rozwiązanie i statystyki populacji. **Unit↔Army**: ruch jest delegowany do `movement_strategy.move(...)`, a jednostka udostępnia pozycję, fitness, honor oraz status bohatera/zbiega. **CommanderAI↔Optimizer**: optymalizator pobiera stan (`get_state`), wybiera akcję (`select_action`) i aktualizuje tablicę Q (`update_q_value`), a komendant zwraca indeks taktyki i szablon formacji.

Mechanizmy zarządzania stanem

Stan systemu jest rozproszony między komponenty: Army utrzymuje stan globalny ($\mathbf{x}^*$, f^*, bieżącą iterację, aktywną taktykę); jednostka – stan lokalny ($\mathbf{x}_i$,$\mathbf{v}_i$, pbest, honor H_i, licznik stagnacji c_i); rekonesans – zbiór komórek siatki $\mathcal{G}$ ze statystykami; Commander AI – tablicę Q, liczniki odwiedzin, tempo eksploracji ϵ oraz bufor pamięci $\mathcal{M}$. Synchronizacja iteracji w implementacji referencyjnej przebiega w kolejności: (1) aplikacja bieżącej taktyki do parametrów ruchu, (2) obserwacja stanu przez Commander AI i ewentualna zmiana taktyki/formacji w punkcie decyzyjnym, (3) aktualizacja celów formacyjnych i propagacja wpływu bohaterów, (4) ruch jednostek i aktualizacja rekonesansu, (5) aktualizacja globalnego $\mathbf{x}^*$ oraz honorów, (6) aktualizacja Q tylko w trybie treningowym.

4.1.3 Autonomiczność i skalowalność

Projektowanie jednostek autonomicznych

Autonomiczność jednostek w ABO opiera się na trzech mechanizmach:

1. Niezależność decyzyjna:

Każda jednostka podejmuje decyzje o swoim ruchu na podstawie lokalnych informacji i globalnego kontekstu, bez centralnej koordynacji na poziomie pojedynczych jednostek:

$$\mathbf{v}_i^{t+1} = f_{\text{velocity}}(\mathbf{x}_i^t, \mathbf{x}_i^{\text{pbest}}, \mathbf{x}^*, \mathbf{r}_i, \mathcal{H}_i, \mathbf{f}_i, \theta_i) \tag{4.20}$$

gdzie:

- $\mathbf{r}_i$ – rekomendacja zwiadowcza dla jednostki i
- $\mathcal{H}_i$ – zbiór bohaterów w zasięgu wpływu jednostki i
- $\mathbf{f}_i$ – pozycja docelowa wynikająca z formacji
- θ_i – parametry specyficzne dla typu jednostki i

2. Emergentne zachowania kolektywne:

Mimo że jednostki działają autonomicznie, ich interakcje prowadzą do emergentnych wzorców zbiorowych [18, 13]:

- *Skupianie się* wokół obiecujących regionów (poprzez wpływ $\mathbf{x}^*$), analogicznie do zachowań stadnych [93, 112]
- *Rozpraszanie się* przy stagnacji (poprzez zwiększenie ϵ)
- *Formowanie hierarchii* (bohaterowie → zwykłe jednostki → zbiegowie)
- *Koordynacja przestrzenna* (poprzez formacje)

Mechanizmy samoorganizacji

Samoorganizacja w ABO zachodzi na trzech poziomach:

Poziom 1: Organizacja bohatera (mikroskala)

System honoru naturalnie segreguje populację bez centralnej kontroli:

$$H_i^{t+1} = \gamma_H H_i^t + \underbrace{10 \cdot \mathbb{I}[\text{improved\_global}]}_{\text{nagroda za poprawę } f^*} - \underbrace{\mathbb{I}[\neg\text{improved\_global}] \cdot \big(0{,}2 + 0{,}1 \cdot \min(5, c_i)\big)}_{\text{kara za brak poprawy } f^* \text{ + stagnacja}} \tag{4.21}$$

gdzie $\gamma_H = 0{,}95$ to współczynnik zanikania honoru (nie mylić z $\gamma = 0{,}9$ – współczynnikiem dyskontowania Q-learningu, Sekcja 4.3.3), $\mathbb{I}[\cdot]$ to funkcja wskazująca. Wartość $\gamma_H = 0{,}95$ jest ustawiona w klasie `Army` (pole `honor_decay`) i przekazywana do metody `update_honor()` każdej jednostki w każdej iteracji.

Emergentna hierarchia:

$$H_i \geq 6{,}0 \implies \text{status} = \text{bohater}, \quad L_i = 1{,}0 + 0{,}5 \cdot \min\big(5,\ \max(0, H_i/10)\big) \tag{4.22}$$

$$H_i \leq -10{,}0 \wedge c_i \geq 10 \wedge g_i = 0 \implies \text{status} = \text{zbieg} \implies \text{„wygaszenie”} \tag{4.23}$$

gdzie L_i to zasięg przywództwa bohatera, c_i to liczba iteracji bez poprawy, a g_i to liczba popraw globalnego najlepszego dokonanych przez jednostkę i. Warunek $g_i = 0$ zapobiega wygaszaniu jednostek, które kiedykolwiek wniosły wkład globalny.

Poziom 2: Organizacja taktyczna (mezoskala)

Taktyki na bieżąco modyfikują parametry jednostek w sposób różnicowy, tworząc emergentną specjalizację. Każda taktyka ustawia słownik `_tactic_params` z parametrami specyficznymi dla strategii ruchu danego typu jednostki.

Na przykład, `FlankingTactic` oblicza kierunek prostopadły $\mathbf{d}_\perp$ do osi centrum masy którym jest aktualne globalne optimum, a następnie przypisuje role:

$$\begin{aligned}\text{Kawaleria} &\rightarrow \text{atak z flanki}: \\ &\texttt{levy\_alpha}=1{,}2,\ \texttt{charge\_distance\_scale}=2{,}0, \\ &\texttt{charge\_direction}=\mathbf{d}_\perp \\ \text{Lekka piechota} &\rightarrow \text{wsparcie skrzydeł}: \\ &\texttt{de\_f\_range}=(0{,}8, 1{,}5),\ \texttt{de\_cr}=0{,}9 \\ \text{Ciężka piechota} &\rightarrow \text{utrzymanie centrum}: \\ &\texttt{step\_scale}=0{,}1,\ \texttt{formation\_pull}=0{,}8 \end{aligned} \tag{4.24}$$

Poziom 3: Organizacja strategiczna (makroskala)
Commander AI adaptacyjnie zmienia taktyki na podstawie stanu całej armii, realizując wysoki poziom samoorganizacji poprzez uczenie się optymalnych strategii:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{4.25}$$

tj. klasyczną regułę aktualizacji Q-learningu [114], gdzie nagroda R_{t+1} jest relatywną poprawą fitness z karą za stagnację (pełna formuła – Sekcja 4.3.3).

4.2 PROJEKTOWANIE JEDNOSTEK BITEWNYCH

Po przedstawieniu architektury modułowej można przejść do projektu poszczególnych typów jednostek bitewnych. W starożytnych armiach różne rodzaje wojsk (piechota ciężka, kawaleria, łucznicy, słonie bojowe) spełniały komplementarne role taktyczne (por. analiza historyczna w Sekcji 3.2). Analogicznie w ABO *heterogeniczna* – czyli różnorodna – populacja składa się z jednostek o wyspecjalizowanych zachowaniach optymalizacyjnych. W odróżnieniu od tradycyjnych algorytmów rojowych, gdzie wszystkie agenty zachowują się identycznie (populacja *homogeniczna*), ABO celowo stosuje agentów różnych typów o różnych strategiach ruchu, co zwiększa zakres przeszukiwanych regionów przestrzeni rozwiązań.

Efektywność algorytmu wynika z *dywersyfikacji strategii* [37]: jednostki ofensywne eksplorują przestrzeń poszukiwań, defensywne eksploatują obiecujące regiony, adaptacyjne dostosowują się do struktury funkcji, a przywódcze koordynują działania armii. Sekcja przedstawia mechanizmy każdej kategorii wraz z formalizacją matematyczną, pseudokodem i analizą właściwości.

4.2.1 Historyczna inspiracja strategii ruchu

Każdy typ jednostki w ABO implementuje odrębny algorytm aktualizacji pozycji, wynikający z analizy historycznego zachowania danego rodzaju wojsk

(por. Rozdział 3). W dalszej części sekcji opisano je w porządku rosnącej złożoności: od najprostszej strategii piechoty ciężkiej (przeszukiwanie po jednej współrzędnej) do najbardziej zaawansowanej strategii słoni bojowych (basin hopping Cauchy'ego z propagacją wpływu). Kluczowa zasada projektowa jest prosta: różne typy jednostek stosują różne strategie przeszukiwania, a ich współdziałanie zapewnia równowagę między eksploracją i eksploatacją na poziomie całej populacji.

Tabela 5 podsumowuje te odwzorowania.

Tabela 5. Odwzorowanie historycznej inspiracji typów jednostek na strategie algorytmiczne

Typ jednostki	**Inspiracja historyczna**	**Strategia algorytmiczna**
Piechota ciężka	Metodyczny marsz hoplitów/legionistów	Przeszukiwanie współrzędnościowe + simpleks
Piechota lekka	Mobilność peltastów/welitów	Mutacja DE z zachłanną selekcją
Łucznicy	Ostrzał obszarowy (salwy)	Wielopunktowe próbkowanie zdalne
Kawaleria	Szarże i pozorowane odwroty	Lot Lévy'ego z persystencją kierunkową
Rydwany	Liniowy ruch z bezwładnością	Momentum Cauchy'ego z próbkowaniem trajektorii
Słonie bojowe	Powolna siła przełamania	Basin hopping Cauchy'ego z propagacją wpływu

Przeszukiwanie współrzędnościowe (*coordinate search*). Z pozycji $\mathbf{x}$ algorytm próbuje $\mathbf{x} \pm h \cdot e_i$ (e_i – wektor jednostkowy i-tej osi, h – długość kroku) i akceptuje ruch przy obniżeniu f. *W ABO:* metodyczny marsz Piechoty Ciężkiej – stabilny, lecz wolny w wysokich wymiarach.

Simpleks / refleksja (*Nelder–Mead*). Utrzymuje $D{+}1$ wierzchołków i wykonuje przekształcenia geometryczne (odbicie, rozszerzenie, kontrakcję), kurcząc figurę wokół minimum. *W ABO:* Piechota Ciężka wywołuje refleksję z prawdopodobieństwem 10%, gdy ruch osiowy nie pomaga.

Mutacja DE (*Differential Evolution*). $\mathbf{v}_i = \mathbf{x}_{r_1} + F \cdot (\mathbf{x}_{r_2} - \mathbf{x}_{r_3})$, gdzie r_1,r_2,r_3 – losowe indeksy, $F \in [0{,}4;\ 1{,}0]$. *W ABO:* Lekka Piechota szybko eksploruje przestrzeń między rozwiązaniami.

Lot Lévy'ego (α-stabilny). Długości kroków losowane z rozkładu α-stabilnego o ciężkich ogonach; $\alpha \in (0;\ 2]$ kontroluje częstość długich skoków ($\alpha = 2$ – rozkład normalny; $\alpha \to 0$ – dłuższe, rzadsze skoki). *W ABO:* Kawaleria wykonuje drobne manewry przerywane szarżami w odległy region.

Persystencja kierunkowa. Jednostka utrzymuje wektor kierunku $\mathbf{d}$ przez $K \in \{3,4,5,6\}$ iteracji (*wheel_threshold*); potem losuje nowy kierunek (od najlepszego osobnika, z podpowiedzi rekonesansu lub losowo). *W ABO: persistent random walk* dla Kawalerii.

Rozkład Cauchy'ego. Gęstość $p(x) = \frac{1}{\pi\gamma\,(1+(x/\gamma)^2)}$; ogony maleją jak $\sim |x|^{-2}$ (Gauss: $\sim e^{-x^2/2}$). Cauchy to przypadek α-stabilny dla $\alpha = 1$ – cięższe ogony niż wariant Lévy'ego ABO ($\alpha = 1{,}5$). *W ABO:* Rydwany łączą Cauchy'ego z bezwładnością.

Momentum trajektoryjne. $\mathbf{v}_t = \mu \cdot \mathbf{v}_{t-1} + (1-\mu) \cdot \mathbf{v}_{\text{nowy}}$, $\mu \in [0; 1)$ (niskie μ: szybka reakcja; wysokie: penetracja odległych obszarów). *W ABO:* `ChariotStrategy` łączy momentum z próbkowaniem Cauchy'ego; $\mu = 0{,}9 - 0{,}2 \cdot t/T_{\max}$.

Basin hopping. Cykl: *perturbacja* (skok stochastyczny) → *lokalna optymalizacja* → *akceptacja Metropolisa*. *W ABO:* Słonie Bojowe z perturbacją Cauchy'ego jako narzędzie przełamywania linii frontu.

Propagacja wpływu. Pozycja docelowa sąsiadów jest aktualizowana w kierunku najlepszej pozycji agenta o wysokim parametrze influence (zarządzane przez `FormationManager`). *W ABO:* kosztowne Słonie (influence = 1,8) szybko propagują odkrycia do całej armii.

Refleksja modularna granic. Przy próbie wyjścia poza kostkę $[\mathbf{b}_{\min}, \mathbf{b}_{\max}]$ stosuje się refleksję „trójkątnej fali": $L = \mathbf{b}_{\max} - \mathbf{b}_{\min}$, $s = (x - \mathbf{b}_{\min}) \bmod 2L$; $x' = \mathbf{b}_{\min} + s$ jeśli $s \le L$, inaczej $x' = \mathbf{b}_{\min} + (2L - s)$. *W ABO:* aktywne dla $D > 10$ (niżej – zwykłe obcięcie); zapobiega sklejaniu jednostek w narożach (patologia *clipping*, np. na XinSheYang01).

Per-component capping i noise scaling. *Capping:* $|v_i| \le c$ dla każdego wymiaru, by pojedyncza składowa Cauchy'ego nie zdominowała ruchu. *Noise scaling:* $\sigma_{\text{eff}} = \sigma_0/\sqrt{D/D_{\text{ref}}}$ przy $D > D_{\text{ref}} = 10$, inaczej $\sigma_{\text{eff}} = \sigma_0$ – utrzymuje oczekiwaną długość szumu $\sigma\sqrt{D}$ zbliżoną do niskowymiarowej. *W ABO:* oba aktywne dla $D > 10$.

Pełne pseudokody i równania aktualizacji pozycji dla każdej z powyższych strategii zawiera Dodatek 7.4.

4.2.2 Jednostki ofensywne

Głównym zadaniem jednostek ofensywnych jest *eksploracja globalna* przestrzeni poszukiwań w celu odkrycia obiecujących regionów. W metaforze militarnej

odpowiadają one kawalerii (szybkie uderzenia w nowe obszary), lekkozbrojnym piechurom (szerokie rozpoznanie) oraz łucznikom (ataki z dystansu, czyli eksploracja dalekich regionów).

Charakterystyka jednostek ofensywnych

W implementacji ABO jednostkami ofensywnymi są:

- **Kawaleria** – najszybsze jednostki (speed = 1,8), bardzo wysokie exploration_factor = 1,6
- **Lekka piechota** – zrównoważone parametry (speed = 1,2, exploration_factor = 1,0)
- **Łucznicy** – długi zasięg ataku (speed = 1,0, exploration_factor = 1,5)

Wspólne cechy tej kategorii:

$$\text{exploration\_factor} > 1{,}0, \quad \text{speed} \geq 0{,}8, \quad \text{exploitation\_factor} \leq 1{,}0 \tag{4.26}$$

Wysoki współczynnik eksploracji powoduje, że jednostki wykonują dłuższe skoki w przestrzeni poszukiwań, zwiększając prawdopodobieństwo odkrycia odległych, potencjalnie lepszych rozwiązań.

Strategie eksploracji przestrzeni

Sposób aktualizacji pozycji jednostek ofensywnych zaprojektowano tak, by przede wszystkim szeroko przeszukiwać przestrzeń rozwiązań. Każdy typ jednostki ofensywnej (Kawaleria, Łucznicy, Lekka Piechota) ma własną, dedykowaną strategię ruchu (klasy `CavalryStrategy`, `ArcherStrategy`, `LightInfantryStrategy`; `movement_strategies.py`), opartą odpowiednio na: lotach Lévy'ego z utrzymaniem kierunku, próbkowaniu wielopunktowym z odległości oraz mutacji *DE/rand/1*. Wspólnym mianownikiem tych strategii jest *dominacja składowej eksploracyjnej* – przyciąganie do bieżącego globalnego optimum jest celowo osłabione i kontrolowane przez parametr `exploration_factor` jednostki, co przeciwdziała przedwczesnej zbieżności populacji w jednym punkcie.

Charakterystyczne cechy projektowe jednostek ofensywnych:

Wysokie momentum trajektoryjne: pozwala na utrzymanie obranego kierunku ruchu, co sprzyja penetracji odległych obszarów przestrzeni (parametr `levy_alpha` w Kawalerii steruje wagą kontynuacji kierunku).

Dominacja pamięci własnej: wpływ historii własnych odkryć jednostki (`best_position`) jest silniejszy niż wpływ globalnego optimum – realizowane przez asymetryczne wagi w poszczególnych strategiach.

Zredukowana atrakcja do $\mathbf{x}^*$: przyciąganie do globalnego optimum jest osłabione i zależne od parametru eksploracji jednostki, aby rój nie zapadał się w jeden punkt zbyt szybko.

Wspólnym dla wszystkich strategii ofensywnych elementem jest składowa eksploracyjna $\mathbf{v}_{\text{explore}}$, realizowana jako:

$$\mathbf{v}_{\text{explore}} = \kappa \cdot \text{exploration\_factor} \cdot \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}) \tag{4.27}$$

gdzie κ jest współczynnikiem skalującym eksplorację (typowo $\kappa = 0{,}1 \times (\mathbf{b}_{\max} - \mathbf{b}_{\min})$, czyli 10% zakresu), a $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ jest wektorem losowym z rozkładu normalnego. W implementacji parametr `exploration` jest stały dla danego typu jednostki (np. Kawaleria: 1,6, Łucznicy: 1,5) i nie zmienia się w czasie – balans eksploracja/eksploatacja jest realizowany przez system taktyk (Sekcja 4.2.5), które dynamicznie modyfikują parametry strategii ruchu (np. `levy_alpha`, `range_scale`, `charge_distance_scale`), a nie przez adaptacyjne skalowanie wariancji.

Mechanizmy przeszukiwania globalnego

Jednostki ofensywne implementują dodatkowe mechanizmy zwiększające zasięg eksploracji:

1. Eksploracja wspierana rekonesansem:

Jednostki ofensywne wykorzystują informacje z systemu wywiadu (reconnaissance) do kierowania eksploracji w obiecujące, ale niezbadane obszary:

$$\mathbf{v}_{\text{recon}} = \beta \cdot \frac{\mathbf{c}_{\text{interest}} - \mathbf{x}_i^t}{\|\mathbf{c}_{\text{interest}} - \mathbf{x}_i^t\|} \cdot \text{interest\_level} \tag{4.28}$$

gdzie $\mathbf{c}_{\text{interest}}$ to współrzędne centrum najbardziej interesującej (wysokie I_g, niskie D_g) komórki siatki wywiadowczej, a β to waga (typowo 0,3). Ta składowa dodawana jest do prędkości jednostek ofensywnych, kierując je w strony wysokiego potencjału.

Adaptacyjne dostosowanie zasięgu

Zasięg eksploracji jednostek ofensywnych jest adaptowany nie przez jawne modyfikowanie współczynnika `exploration` w czasie, lecz pośrednio – poprzez system taktyk, który zmienia parametry strategii ruchu w każdej iteracji. Na przykład:

Taktyka `Scouting` (zwiadowcza) ustawia kawalerii `charge_distance_scale`=3,0 i `levy_alpha`=1,0 (cięższe ogony Lévy'ego $\implies$ dłuższe skoki), podczas gdy taktyka `Phalanx` ogranicza ją do `charge_distance_scale`=0,5 z `levy_alpha`=2,0 (bliżej rozkładu normalnego).

Taktyka `Scouting` realizuje głęboką integrację z modułem `Reconnaissance Intelligence` (Alg. 15). Gdy Commander AI wybiera taktykę zwiadowczą, jednostki otrzymują rekomendacje kierunkowe z systemu rozpoznania poprzez wywołanie `get_recommendation(position, exploration_vs_exploitation=0.8)`, co silnie faworyzuje eksplorację nieodwiedzonych regionów. Kawaleria kieruje szarże w stronę rekomendowanych regionów, a piechota ciężka (HeavyInfantry) pełni rolę zwiadowcy-kotwicy, przeszukując lokalne otoczenie wskazanych

obszarów. Taka koordynacja pozwala systematycznie mapować przestrzeń poszukiwań zamiast polegać na losowej eksploracji.

Commander AI, obserwując stan optymalizacji (stagnację, postęp, różnorodność), przełącza taktyki adaptacyjnie – co jest funkcjonalnym odpowiednikiem adaptacyjnego dostosowania zasięgu, ale realizowanym na poziomie strategicznym, a nie przez lokalne formuły modyfikujące parametry.

Implementacja operatorów eksploracyjnych

Strategia ruchu kawalerii opiera się na mechanizmie lotów Lévy'ego (*ang. Lévy flights*) [128]. Lot Lévy'ego to ruch losowy, w którym większość kroków jest krótka, ale sporadycznie zdarza się bardzo długi skok. W odróżnieniu od zwykłego ruchu losowego, gdzie wszystkie kroki mają podobną długość, rozkład Lévy-stabilny ma tzw. *ciężkie ogony* – prawdopodobieństwo ekstremalnie długiego skoku jest znacznie większe niż w rozkładzie normalnym. Taka charakterystyka pozwala na nagłą zmianę pozycji, analogiczną do szarży kawalerii, która po serii drobnych manewrów nagle galopuje na drugą stronę pola bitwy. Dzięki temu kawaleria jest najbardziej eksploracyjnym typem jednostki w algorytmie ABO.

Algorytm 1. Aktualizacja jednostki ofensywnej – Cavalry (lot Lévy'ego [77, 128] z persystencją kierunkową)

Require: Parametry: s=1,8, ϵ=1,6, ξ=0,9, μ=1,7, ι=1,1
Require: Pozycja $\mathbf{x}_i$, kierunek szarży $\hat{\mathbf{d}}_i$, licznik kroków c_{charge}, próg zwrotu $k_{\text{wheel}} \sim \mathcal{U}\{3, 7\}$
// Faza 1: Wyznaczenie kierunku szarży (gdy $c_{\text{charge}} = 0$)
if $c_{\text{charge}} = 0$ **then**
  $r \leftarrow \text{rand}()$
  if $r < 0{,}15$ **and** hero_bias dostępne **then**
    $\hat{\mathbf{d}}_i \leftarrow \text{normalize}(\text{hero\_bias})$ {ku bohaterowi}
  else if $r < 0{,}75$ **and** $\mathbf{g} \neq \text{null}$ **then**
    $\hat{\mathbf{d}}_i \leftarrow \text{normalize}(\mathbf{g} - \mathbf{x}_i)$ {ku globalnemu best}
  else if $r < 0{,}90$ **and** rekonesans dostępny **then**
    $\hat{\mathbf{d}}_i \leftarrow \text{normalize}(\text{recon\_recommendation})$
  else
    $\hat{\mathbf{d}}_i \leftarrow$ losowy wektor jednostkowy {losowa eksploracja}
  end if
  $k_{\text{wheel}} \leftarrow \mathcal{U}\{3, 7\}$
end if
// Faza 2: Obliczenie kroku Lévy'ego
$L \leftarrow |\text{Lévy}(\alpha{=}1{,}5)|$ {próbka z rozkładu Lévy-stabilnego}
$\ell \leftarrow L \cdot s \cdot 0{,}1 \cdot \text{search\_range}$
$\ell \leftarrow \text{clip}(\ell,\ 0{,}01 \cdot \text{search\_range},\ 0{,}5 \cdot \text{search\_range})$
// Faza 3: Ruch wzdłuż kierunku szarży
$\mathbf{x}_i \leftarrow \text{reflect}(\mathbf{x}_i + \ell \cdot \hat{\mathbf{d}}_i,\ \mathbf{b}_{\min},\ \mathbf{b}_{\max})$
$f_i \leftarrow f(\mathbf{x}_i)$; $c_{\text{charge}} \leftarrow c_{\text{charge}} + 1$
// Faza 4: Manewr skrętu (wheel) po k_{wheel} krokach
if $c_{\text{charge}} \geq k_{\text{wheel}}$ **then**
  $\theta \sim \mathcal{U}(-\pi/3,\ \pi/3)$ {kąt obrotu}
  $\hat{\mathbf{d}}_i \leftarrow \text{rotate\_in\_random\_plane}(\hat{\mathbf{d}}_i,\ \theta)$

26: $c_{\text{charge}} \leftarrow 0$
27: **end if**
28: **// Faza 5: Pozorowany odwrót (10% prawdopodobieństwo)**
29: **if** rand() < 0,1 **then**
30: $\hat{\mathbf{d}}_i \leftarrow -\hat{\mathbf{d}}_i$ {odwrócenie kierunku szarży}
31: **end if**

Jednostki ofensywne to główny mechanizm unikania przedwczesnej zbieżności algorytmu. Wysoki współczynnik eksploracji, specjalizowane strategie ruchu (loty Lévy'ego kawalerii, salwy łuczników, trajektorie Cauchy'ego rydwanów) oraz wskazówki z systemu rozpoznania razem zapewniają ciągłe odkrywanie nowych regionów przestrzeni. System taktyk sterowany przez Commander AI adaptacyjnie dostosowuje parametry strategii ruchu, realizując dynamiczną równowagę między eksploracją a eksploatacją.

4.2.3 Jednostki defensywne

Podczas gdy jednostki ofensywne poszukują nowych terytoriów, jednostki defensywne skupiają się na *eksploatacji lokalnej* – dokładnym przeszukiwaniu obiecujących regionów już odkrytych. W metaforze militarnej odpowiadają one ciężkozbrojnej piechocie (powolnej, ale silnej i wytrwałej) oraz formacjom obronnym chroniącym zdobyte pozycje.

Charakterystyka jednostek defensywnych

W algorytmie ABO główną jednostką defensywną jest:

- **Ciężka piechota** – niska prędkość (speed = 0,7), niski poziom eksploracji (exploration_factor = 0,6), wysoki poziom eksploatacji (exploitation_factor = 1,5)

Rydwany Bojowe (speed = 1,6, exploration_factor = 1,2, exploitation_factor = 1,2, por. Tabela 4) posiadają zrównoważone parametry, lecz dzięki wysokiemu parametrowi wpływu (influence = 1,3) i formacji liniowej wspierają eksploatację lokalną w koordynacji z ciężką piechotą.

Cechy charakterystyczne dla jednostek defensywnych:

$$\text{exploitation\_factor} \geq 1{,}0, \quad \text{exploration\_factor} \leq 1{,}2, \quad \text{speed} \leq 1{,}6 \tag{4.29}$$

Niski współczynnik eksploracji lub niska prędkość sprawiają, że jednostki wykonują małe, precyzyjne kroki, dokładnie analizując lokalną strukturę funkcji.

Techniki eksploatacji lokalnej

Podstawowa aktualizacja jednostki defensywnej jest bardziej konserwatywna niż dla jednostek ofensywnych. Strategia ruchu Ciężkiej Piechoty (`HeavyInfantryStrategy`, `movement_strategies.py`) realizuje *przeszukiwanie po współrzędnych* (*coordinate descent*) z krokiem typu Neldera-Meada, modulowanym przez parametry:

- `step_scale` – skala lokalnego kroku (typowo 0,1–0,3, modyfikowana przez taktykę); małe wartości zapewniają precyzyjną eksploatację okolic $\mathbf{x}^*$
- `formation_pull` – siła przyciągania do pozycji w formacji (silna dla jednostek defensywnych: 0,6–0,8)
- `exploitation_factor` = 1,5 – wysoka, wbudowana skłonność do lokalnej eksploatacji (Tabela 4)
- $\mathbf{v}_{\text{exploit}}$ – składowa eksploatacyjna realizująca małe, precyzyjne ruchy w kierunku globalnego optimum

Składowa eksploatacyjna realizowana jest jako:

$$\mathbf{v}_{\text{exploit}} = \delta \cdot \text{exploitation\_factor} \cdot \mathcal{U}(-\mathbf{1}, \mathbf{1}) \odot (\mathbf{g} - \mathbf{x}_i^t) \tag{4.30}$$

gdzie δ jest małym współczynnikiem (typowo $\delta = 0{,}01$), a $\mathcal{U}(-\mathbf{1}, \mathbf{1})$ jest wektorem losowym z rozkładu jednostajnego. W przeciwieństwie do jednostek ofensywnych (duże losowe skoki), jednostki defensywne wykonują małe perturbacje wokół kierunku do globalnego najlepszego wyniku.

1. Przybliżenie gradientowe (zaimplementowane w systemie rozpoznania):
System rozpoznania (reconnaissance) implementuje aproksymację gradientu na poziomie globalnym, estymując kierunki spadku w poszczególnych regionach przestrzeni poszukiwań. Jednostki defensywne mogą wykorzystać te informacje do wykonywania kroków w kierunku największego spadku:

$$\nabla f(\mathbf{x}_i) \approx \left[\frac{f(\mathbf{x}_i + h\mathbf{e}_j) - f(\mathbf{x}_i - h\mathbf{e}_j)}{2h} \right]_{j=1}^{d} \tag{4.31}$$

gdzie $\mathbf{e}_j$ to j-ty wektor jednostkowy, a h to mały krok (typowo $h = 10^{-5}$). Następnie:

$$\mathbf{v}_{\text{gradient}} = -\eta \cdot \frac{\nabla f(\mathbf{x}_i)}{\|\nabla f(\mathbf{x}_i)\|} \tag{4.32}$$

gdzie η to rozmiar kroku (dostosowany do exploitation factor). Ta składowa dodawana jest do prędkości jednostki.

2. Przypadek zgniecenia agenta

Dywersyfikacja jednostek defensywnych jest zapewniana przez system formacji – FormationManager przydziela pozycje docelowe wokół globalnego optimum, a elastyczne przyciąganie formacyjne przemieszcza jednostki ku tym pozycjom z siłą zależną od dyscypliny formacyjnej i odległości od pozycji docelowej. Dodatkowo, system taktyk różnicuje zachowanie: np. taktyka `Surrounding` rozmieszcza jednostki na okręgu wokół $\mathbf{x}^*$ z promieniem $r(p) = 0{,}5 + 1{,}0 \cdot (1 - p^2)$. Parametr $p \in [0,1]$ oznacza znormalizowany postęp iteracji ($p = t/T$); taki kształt promienia zapewnia szeroką dywersyfikację na początku przeszukiwania i stopniowe zacieśnianie szyku w jego końcowej fazie. W ten sposób oba mechanizmy – geometria formacji oraz adaptacyjny promień taktyk – zapobiegają zgnieceniu jednostek defensywnych w jednym punkcie i utrzymują pokrycie sąsiednich regionów przestrzeni rozwiązań.

Balansowanie intensywności przeszukiwania

Jednostki defensywne muszą balansować między intensywną eksploatacją a unikaniem przedwczesnego utknięcia w lokalnym minimum. Parametry `exploitation` i `exploration` są stałe dla danego typu jednostki (np. ciężka piechota: ξ=1,5, ϵ=0,6), ale system taktyk modyfikuje parametry strategii ruchu – na przykład taktyka `Phalanx` ustawia `step_scale`=0,3 (drobne kroki), podczas gdy `Flanking` ustawia `step_scale`=0,1 z `formation_pull`=0,8 (kowadło trzymające centrum). Adaptacja zachodzi więc na poziomie taktycznym, nie przez jawne modyfikowanie współczynników w czasie.

1. Stagnacja: jednostka, która nie poprawia personal best przez $k_{\text{stagnation}} = 30$ iteracji (domyślnie), jest restartowana w sąsiedztwie aktualnego znanego globalnego optimum z perturbacją gaussowską (Faza 4 Algorytmu 2). Próg może nadpisać taktyka – np. `Phalanx` ustawia 80 (dłuższa cierpliwość przy intensywnej eksploatacji).

2. System honoru: jednostki z wysokim honorem ($H_i \geq 6{,}0$) uzyskują status bohatera, co zmniejsza ich dyscyplinę formacyjną o 30% i pozwala wpływać na pobliskie jednostki przez mechanizm dzielenia się wiedzą (Sekcja 4.2.4).

Implementacja jednostki defensywnej

Algorytm 2. Aktualizacja jednostki defensywnej – HeavyInfantry (przeszukiwanie współrzędnościowe [121])

Require: Parametry: s=0,7, ϵ=0,6, ξ=1,5, μ=2,3
Require: Pozycja $\mathbf{x}_i$, najl. osobiste $\mathbf{p}_i$, najl. globalne $\mathbf{g}$, oś j (cyklicznie), licznik stagnacji c_{stag}
// Faza 1: Przeszukiwanie współrzędnościowe (coordinate descent)
$n_{\text{axes}} \leftarrow \max(1, \lfloor d/20 \rfloor)$ {1 oś dla $d \leq 20$, 5 osi dla d=100}
if $d \leq 20$ **then**
    $\delta \leftarrow 0{,}05 \cdot \text{search\_range} \cdot \xi \cdot e^{-3 \cdot t/T_{\max}}$ {zanik wykładniczy}
else
    $\delta \leftarrow 0{,}05 \cdot \text{search\_range} \cdot \xi \cdot (1 - 0{,}7 \cdot t/T_{\max})$ {zanik liniowy}
end if
improved $\leftarrow$ false

for $k = 1$ **to** n_{axes} **do**
  $a \leftarrow$ oś j (cykliczna) lub losowa (dla $k > 1$)
  for $s \in \{+1, -1\}$ **do**
    $\mathbf{x}' \leftarrow \mathbf{x}_i$; $x'_a \leftarrow x'_a + s \cdot \delta$
    $\mathbf{x}' \leftarrow \text{reflect}(\mathbf{x}', \mathbf{b}_{\min}, \mathbf{b}_{\max})$
    if $f(\mathbf{x}') < f(\mathbf{x}_i)$ **then**
      $\mathbf{x}_i \leftarrow \mathbf{x}'$; improved $\leftarrow$ true; $c_{\text{stag}} \leftarrow 0$
      break {przejdź do następnej osi}
    end if
  end for
  if improved **then**
    break
  end if
end for
$j \leftarrow j + 1$
// Faza 2: Refleksja simpleksowa (10% prawdop., gdy brak poprawy)
if ¬improved **and** rand() $< 0{,}10$ **then**
  $\mathbf{c} \leftarrow 0{,}5 \cdot (\mathbf{p}_i + \mathbf{g})$ {centroid}
  $\mathbf{x}_{\text{ref}} \leftarrow \text{reflect}(2\mathbf{c} - \mathbf{x}_i, \mathbf{b}_{\min}, \mathbf{b}_{\max})$
  if $f(\mathbf{x}_{\text{ref}}) < f(\mathbf{x}_i)$ **then**
    $\mathbf{x}_i \leftarrow \mathbf{x}_{\text{ref}}$; $c_{\text{stag}} \leftarrow 0$
  end if
end if
// Faza 3: Krok awaryjny ku osobistemu najlepszemu
if ¬improved **then**
  $c_{\text{stag}} \leftarrow c_{\text{stag}} + 1$
  $\boldsymbol{\eta} \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{I}) \cdot s \cdot 0{,}01 \cdot \text{search\_range}/\sqrt{d/10}$
  $\mathbf{x}_i \leftarrow \text{reflect}(\mathbf{x}_i + \boldsymbol{\eta} + 0{,}3(\mathbf{p}_i - \mathbf{x}_i) + \text{hero\_bias}, \mathbf{b}_{\min}, \mathbf{b}_{\max})$
end if
// Faza 4: Reset antystagnacyjny (po 30 iteracjach bez poprawy)
if $c_{\text{stag}} \geq 30$ **then**
  $\boldsymbol{\eta}_{\text{reset}} \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{I}) \cdot 0{,}05 \cdot \text{search\_range}/\sqrt{d/10}$
  $\mathbf{x}_i \leftarrow \text{reflect}(\mathbf{g} + \boldsymbol{\eta}_{\text{reset}}, \mathbf{b}_{\min}, \mathbf{b}_{\max})$
  $c_{\text{stag}} \leftarrow 0$
end if

Jednostki defensywne są niezbędne dla *dokładności* algorytmu. Przeszukiwanie współrzędnościowe pozwala precyzyjnie zlokalizować optimum w obiecującym regionie. Refleksja simpleksowa (10% prawdopodobieństwo) wprowadza ruch w kierunku centroidu najlepszych rozwiązań, gdy przeszukiwanie osiowe nie przynosi poprawy. Mechanizm antystagnacyjny resetuje pozycję w okolicę globalnego optimum po 30 iteracjach bez poprawy, zapobiegając ugrzęźnięciu.

4.2.4 Honor i bohaterowie pola bitwy

Mechanizm systemu honoru – progi awansu ($H \geq 6{,}0$ dla bohatera) i degradacji ($H \leq -10{,}0$ dla zbiega), formułę aktualizacji honoru oraz emergentną hierarchię – opisano w Sekcji 4.1 (Równania 4.11–4.13). Niniejsza sekcja skupia się na tym, jak status bohatera wpływa na zachowanie jednostki.

Słonie (speed = 0,9, ale bardzo silne oddziaływanie) mają wysoki parametr wpływu = 1,8, co sprawia, że naturalnie pełnią rolę liderów w armii a oddziały naturalnie podążają za nimi.

Cecha charakterystyczna:

$$\text{memory} > 2{,}0, \quad \text{influence} > 1{,}5, \quad \text{adaptacyjne parametry} \tag{4.33}$$

Wysoka pamięć (memory) oznacza, że jednostki te śledzą swoją historię eksploracji i wykorzystują ją do podejmowania decyzji. Wysoki wpływ (influence) sprawia, że ich odkrycia silnie wpływają na resztę armii.

Mechanizmy uczenia się

Jednostki o statusie bohatera implementują mechanizmy uczenia się na podstawie historycznych danych. Poniższe mechanizmy opisują zachowanie tych jednostek, niezależnie od ich typu bazowego.

1. Pamięć historyczna:

Jednostka o statusie bohatera utrzymuje historię odwiedzonych pozycji i ich fitness:

$$\mathcal{H}_{\text{hero}} = \{(\mathbf{x}^{(k)}, f^{(k)}, t^{(k)})\}_{k=1}^{K} \tag{4.34}$$

gdzie K to rozmiar pamięci (typowo $K = 100$). Na podstawie tej historii jednostka może:

- Wykrywać regiony, które już zbadała (i unikać ich ponownie)
- Identyfikować trajektorie, które doprowadziły do poprawy
- Analizować korelacje między kierunkiem ruchu a poprawą fitness

2. Dwupoziomowa adaptacja parametrów ruchu:

Per-jednostkowe współczynniki `exploration` i `exploitation` (Tabela 4) są **stałe dla danego typu jednostki** przez cały przebieg optymalizacji – ABO świadomie rezygnuje z ich modyfikowania w czasie i przenosi adaptację o jeden poziom wyżej, do warstwy taktycznej i meta-kontrolera. Dzięki temu nie powstaje współzawodnictwo między dwiema niezależnymi pętlami adaptacji (per-jednostka vs. globalna), a system pozostaje przewidywalny analitycznie.

Faktyczna adaptacja zachodzi na dwóch poziomach:

Poziom taktyczny

W każdej iteracji aktywna jest jedna z sześciu taktyk (*Phalanx*, *Oblique Order*, *Center Penetration*, *Flanking*, *Surrounding*, *Scouting*; klasa `Tactic` w module `tactics.py`). Wybrana taktyka ustawia parametry strategii ruchu wspólne dla wszystkich jednostek danego typu, m.in.: `step_scale` (skala kroku),

`formation_pull` (siła przyciągania formacyjnego), `stampede_prob` (prawdopodobieństwo skoku w kierunku optymalnym), `cauchy_scale` (rozrzut perturbacji Cauchy'ego) – i dynamicznie modyfikuje dyscyplinę formacyjną δ_i jednostek (np. *Phalanx* zwiększa δ o 0,05 na iterację aż do wartości 1,0, podczas gdy *Flanking* obniża ją do 0,3).

Poziom meta-kontroli

Wybór samej taktyki podejmuje moduł Commander AI (Sekcja 4.2.5), trenowany metodą Q-learning. Co τ iteracji obserwuje on pięciowymiarowy stan krajobrazu (faza optymalizacji, stagnacja, momentum, różnorodność populacji, odsetek bohaterów) i wybiera taktykę o najwyższej oczekiwanej nagrodzie. Powoduje to, że profil eksploracja–eksploatacja całego roju zmienia się dynamicznie w odpowiedzi na bieżący stan przeszukiwania, mimo że indywidualne współczynniki jednostek pozostają niezmienione.

Co adaptuje się per jednostka

Jednostki utrzymują liczniki `iterations_without_improvement` oraz `global_improvements` (`units.py`), które wpływają na: (i) restart antystagnacyjny po 30 iteracjach bez poprawy (opisany w sekcji „Stagnacja" powyżej) oraz (ii) aktualizację honoru i tym samym status bohatera ($H \geq 6{,}0$), który redukuje siłę przyciągania formacyjnego o 30%. Nie modyfikują one jednak współczynników `exploration/exploitation` samej jednostki – te pozostają stałe i stanowią „DNA" typu jednostki, a nie zmienną stanu optymalizacji.

Dynamiczna modyfikacja parametrów na polu bitwy

Oprócz exploration/exploitation factors, jednostki bohaterskie dostosowują inne parametry:

1. Dzielenie wiedzy oparte na wpływie:

Jednostka o statusie bohatera, dzięki wysokiemu parametrowi wpływu, może *dzielić się swoją wiedzą* z innymi jednostkami. Gdy bohater znajdzie obiecujący region, „wzywa" inne jednostki do tej lokalizacji, tymczasowo modyfikując ich globalny atraktor – jest to jednak część zachowania jednostek, a nie wymuszona akcja:

$$\mathbf{g}^i_{\text{effective}} = \begin{cases} \mathbf{x}_{\text{hero}} & \text{jeśli jednostka } i \text{ w promieniu wpływu hero} \\ \mathbf{g} & \text{w przeciwnym razie} \end{cases} \tag{4.35}$$

gdzie promień wpływu:

$$R_{\text{influence}} = L_i \times \frac{10{,}0}{|f^*| + \varepsilon_R} \tag{4.36}$$

gdzie $L_i = 1{,}0 + 0{,}5 \times \min(5,\ \max(0, H_i/10))$ to zasięg przywództwa bohatera (rosnący z honorem), f^* to bieżąca wartość globalnego optimum, a ε_R (w implementacji 10^{-10}) stabilizuje mianownik dla minimów równych zeru lub wartości ujemnych. Promień wpływu adaptuje się do skali funkcji celu – przy bliskich wartościach $|f^*|$ zasięg rośnie, wymuszając intensywniejszą koordynację w końcowej fazie zbieżności.

Implementacja mechanizmu bohaterów

W implementacji jednostka o statusie bohatera *nie stosuje* odrębnej strategii ruchu – porusza się tą samą strategią, co jej typ (np. ciężka piechota-bohater nadal używa HeavyInfantryStrategy). Jedyne różnice behawioralne to:

1. **Zmniejszona dyscyplina formacyjna** – dyscyplina formacyjna bohatera wynosi $0{,}7 \times \delta_f$ oryginalnej wartości, co daje mu większą swobodę ruchu w ramach formacji.

2. **Dzielenie się wiedzą** – bohater wpływa na pobliskie jednostki (w promieniu $R_{\text{influence}}$), tworząc wektor `hero_influence` kierujący je ku pozycji bohatera:

$$\texttt{hero\_influence}_j \mathrel{+}= \lambda \cdot (\mathbf{x}_{\text{hero}} - \mathbf{x}_j), \quad \lambda = (0{,}1 + 0{,}1 \cdot H_{\text{hero}}/10) \cdot (1 - d_{hj}/R_{\text{influence}}) \tag{4.37}$$

gdzie $d_{hj} = \|\mathbf{x}_{\text{hero}} - \mathbf{x}_j\|$. Wektor `hero_influence` jest następnie uwzględniany jako `hero_bias` w strategiach ruchu poszczególnych typów jednostek (z wagami zależnymi od typu: kawaleria 0,3, łucznicy 0,25, lekka piechota 0,2, rydwany 0,15, ciężka piechota 0,3, słonie 0,2).

Jednostka o statusie bohatera pełni rolę „oficera" armii – koordynuje działania pobliskich jednostek i dzieli się wiedzą, dzięki czemu algorytm intensywniej przeszukuje obiecujące regiony odkryte przez najlepsze jednostki.

4.2.5 Jednostki przywódcze

Podczas gdy jednostki ofensywne, defensywne i bohaterskie realizują faktyczne przeszukiwanie przestrzeni, jednostki przywódcze pełnią rolę *meta-optymalizatorów* – nie szukają bezpośrednio optimum, ale koordynują i optymalizują działania pozostałych jednostek. W metaforze militarnej odpowiadają one generałom, którzy obserwują pole bitwy z góry i podejmują strategiczne decyzje.

W algorytmie ABO rolę jednostki przywódczej na najwyższym poziomie pełni **Commander AI** – system oparty na uczeniu przez wzmacnianie [104] (Q-learning [114]), który dynamicznie dobiera taktyki dla całej armii. Zasady dowodzenia i adaptacyjnego podejmowania decyzji, na których oparto Commander AI, opisano w kontekście historycznym w Sekcji 3.4 Rozdziału 3 (hierarchia dowodzenia i adaptacyjne podejmowanie decyzji).

Charakterystyka Commander AI

Commander AI nie jest tradycyjną jednostką optymalizacyjną (nie posiada pozycji w przestrzeni poszukiwań), ale *agentem decyzyjnym* podejmującym decyzje:

- **Przestrzeń stanów**: $3^5 = 243$ stanów opisujących stan algorytmu (faza optymalizacji, stagnacja, momentum, różnorodność, odsetek bohaterów)
- **Przestrzeń akcji**: 6 akcji (6 taktyk; formacje dobierane automatycznie do taktyk)
- **Funkcja nagrody**: oparta na poprawie fitness i karze za stagnację
- **Q-tabela**: 243×6 zapisywana i aktualizowana przez Q-learning

Automatyczne mapowanie taktyki na formację (`tactic_formation_map`) jest motywowane historycznie – każda taktyka wymaga specyficznego rozstawienia wojsk na polu bitwy:

Tabela 6. Automatyczne mapowanie taktyk na szablony formacji w Commander AI

Taktyka	Formacja	Uzasadnienie historyczne
Phalanx	defensive	Falanga była historycznie formacją ofensywną (por. Sekcja 3.2), lecz jej cechą definiującą była *powolność i dyscyplina* natarcia. W algorytmie ta powolność i dyscyplina odpowiada eksploatacji – dokładnemu przeszukiwaniu wąskiego regionu wokół najlepszego rozwiązania. Atak w metaforze wojennej = eksploatacja w metaforze optymalizacyjnej
Oblique Order	offensive	Epaminondas rozmieszczał siły w trzech grupach: silne skrzydło (30%) do ataku, centrum utrzymania (40%), słabe skrzydło eksploracji (30%) – asymetryczna struktura dla przełamania [95] (Leuktra, 371 p.n.e.)
Center Penetration	offensive	Klin Aleksandra Wielkiego celował w przebicie centrum [43] (Gaugamela, 331 p.n.e.)
Flanking Maneuver	flanking	Oskrzydlenie wymaga rozłożenia sił przestrzennie wokół celu
Surrounding	balanced	Dynamiczne okrążenie z spiralnym wachlowaniem wokół celu (Kanny, 216 p.n.e. [88, 50]) – jednostki krążą na różnych promieniach z pulsacją do uniknięcia stagnacji
Scouting	exploratory	Oddziały zwiadowcze rozpraszają się maksymalnie dla pokrycia terenu

Różnica jest wyraźna: jednostki bojowe optymalizują *gdzie szukać*, Commander optymalizuje *jak szukać*. Ta architektura realizuje ideę adaptacyjnej selekcji

operatorów [106, 39], w której meta-poziom steruje doborem operatorów niższego poziomu.

Monitorowanie wydajności systemu

Commander AI oblicza pięciowymiarowy wektor stanu:

$$s = (\text{faza, stagnacja, momentum, różnorodność, odsetek bohaterów}), \tag{4.38}$$

którego składowe opisują bieżący stan algorytmu. Każda cecha jest dyskretyzowana do trzech poziomów, co daje przestrzeń $3^5 = 243$ stanów (metoda `get_state`):

1. Faza optymalizacji (postęp):

$$p_{\text{progress}} = \frac{t}{T_{\max}} \in [0, 1] \tag{4.39}$$

Dyskretyzowana do 3 poziomów: wczesna ($p < 0{,}3$), środkowa ($0{,}3 \leq p < 0{,}7$), późna ($p \geq 0{,}7$).

2. Stagnacja:

$$\text{stagnation} = \overline{c} = \frac{1}{|U_{\text{akt}}|} \sum_{i \in U_{\text{akt}}} c_i \tag{4.40}$$

gdzie c_i to licznik iteracji bez poprawy jednostki i, a U_{akt} to zbiór jednostek aktywnych. Dyskretyzowana do 3 poziomów: niska ($< 0{,}4$), umiarkowana ($[0{,}4, 0{,}8)$), silna ($\geq 0{,}8$). Ponieważ c_i są licznikami całkowitoliczbowymi, średnia $\overline{c}$ pozostaje poniżej progu 0,8 tylko wtedy, gdy większość jednostek poprawiła swoje minimum w bieżącej iteracji; już po jednej–dwóch iteracjach bez poprawy cecha przechodzi na poziom *silna* i utrzymuje się na nim przez większość przebiegu. W praktyce składowa ta działa więc jako detektor świeżych popraw, różnicujący stany głównie w najwcześniejszych iteracjach oraz bezpośrednio po znalezieniu lepszego rozwiązania.

3. Momentum (tempo poprawy):

$$\text{momentum} = \frac{1}{|W|} \sum_{k \in W} \frac{f_{k-1} - f_k}{|f_{k-1}| + \varepsilon} \tag{4.41}$$

tj. średnia względna poprawa globalnego fitness w oknie ostatnich obserwacji W (do 10). Dyskretyzowane do 3 poziomów: szybkie ($\geq 0{,}015$), wolne ($[0{,}003, 0{,}015)$), zanikające ($< 0{,}003$).

4. Różnorodność populacji:

$$\text{diversity} = \frac{\overline{\sigma_d}}{\text{search\_range}} \tag{4.42}$$

tj. średnie po wymiarach odchylenie standardowe pozycji jednostek, znormalizowane do zakresu przestrzeni. Dyskretyzowana do 3 poziomów: skupiona ($\leq 0{,}04$), umiarkowana ($(0{,}04, 0{,}07]$), rozproszona ($> 0{,}07$).

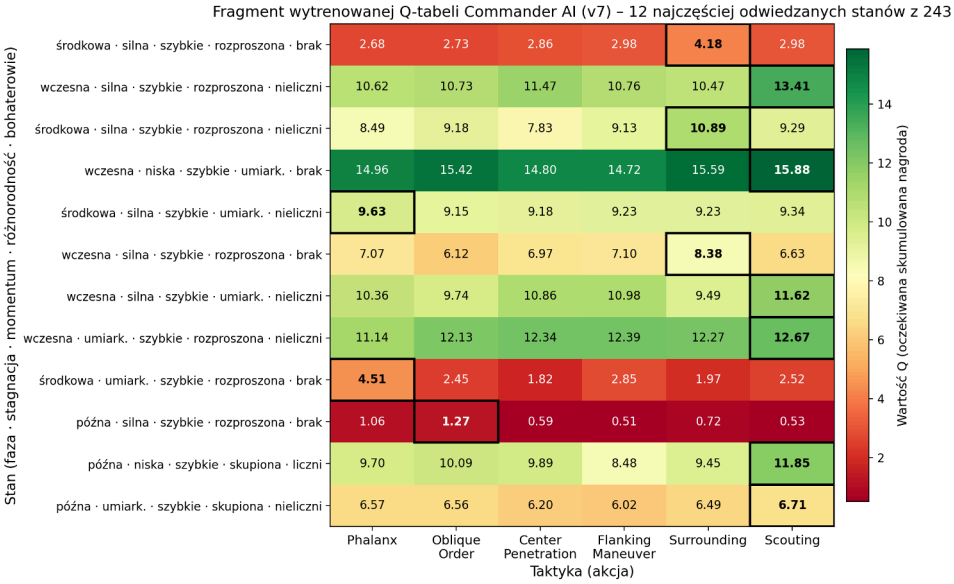
5. Odsetek bohaterów:

$$\text{hero\_ratio} = \frac{\text{liczba jednostek o statusie bohatera}}{|U_{\text{akt}}|} \tag{4.43}$$

Dyskretyzowany do 3 poziomów: brak (= 0), nieliczni (≤ 0,08), liczni (> 0,08).

Przestrzeń stanów $\mathcal{S}$ ($3^5 = 243$ stanów) i jej dyskretyzację opisano w Sekcji 4.1 (Moduł 5).

Rysunek 6 przedstawia fragment wytrenowanej Q-tabeli Commander AI (wariant v7): 12 najczęściej odwiedzanych stanów spośród 243, gdzie wartości Q wskazują oczekiwaną nagrodę za wybór danej taktyki w określonym stanie optymalizacji. Wysokie wartości (zielone) sugerują preferowane akcje, niskie (czerwone) – akcje unikane.



Rys. 6. Mapa cieplna fragmentu wytrenowanej Q-tabeli Commander AI (v7): wartości Q dla 12 najczęściej odwiedzanych stanów (spośród 243) i sześciu taktyk. Czarna ramka wyróżnia akcję o najwyższej wartości Q w danym stanie ($\arg\max_a Q(s,a)$)

Mechanizmy podejmowania decyzji

Commander AI podejmuje decyzje przy użyciu Q-learningu [114] – uczenia przez wzmacnianie bez modelu [104, 62]:

1. Aktualizacja Q-learning:

Reguła aktualizacji Q-tabeli z parametrami $\alpha = 0{,}1$ (wartość początkowa), $\gamma = 0{,}9$:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{4.44}$$

2. Funkcja nagrody:

Nagroda r opiera się wyłącznie na poprawie fitness. Wczesne wersje algorytmu uwzględniały dodatkowe składniki (kondycja armii, różnorodność eksploracji, odkrycia wywiadowcze), lecz eksperymenty wykazały *reward hacking* – Commander uczył się polityk optymalizujących metryki pomocnicze kosztem faktycznej jakości rozwiązań. Ostatecznie nagroda sprowadzona została do dwóch składników:

$$r = \text{clip}(10{,}0 \cdot \Delta_{\text{rel}},\ -2{,}0,\ 5{,}0) + r_{\text{stag}} \tag{4.45}$$

gdzie:

- $\Delta_{\text{rel}} = (f_{\text{old}} - f_{\text{new}})/\max(|f_{\text{old}}|, 10^{-9})$ – relatywna poprawa fitness
- $r_{\text{stag}} = -0{,}3$ jeśli $\Delta_{\text{rel}} \leq 0$, w przeciwnym razie 0 – kara za stagnację

3. Strategia wyboru akcji (ϵ-greedy):

Commander używa eksploracji ϵ-greedy [104, 109]:

$$a = \begin{cases} \text{random action} & \text{z prawdopodobieństwem } \epsilon \\ \arg\max_{a'} Q(s, a') & \text{w przeciwnym razie (greedy)} \end{cases} \tag{4.46}$$

gdzie ϵ maleje liniowo wraz z postępem optymalizacji:

$$\epsilon(t) = \max\bigl(0{,}03,\ 0{,}12 \cdot (1 - t/T_{\max})\bigr) \tag{4.47}$$

Powyższy wzór opisuje harmonogram eksploracji stosowany w trybie ewaluacyjnym (readonly), tj. podczas benchmarków opisanych w Rozdziale 6. Współczynnik eksploracji zaczyna od 12% i maleje liniowo do minimalnego poziomu 3%, który jest osiągany przy $t/T_{\max} = 0{,}75$.

W trybie treningowym (akumulacja Q-tabeli) stosowany jest natomiast zanik eksponencjalny:

$$\epsilon(t+1) = \max\bigl(0{,}1,\ \epsilon(t) \cdot 0{,}99\bigr), \qquad \epsilon(0) = 0{,}3 \tag{4.48}$$

Wyższa wartość startowa (30%) i wolniejszy zanik (multiplikatywny $\times 0{,}99$) zapewniają intensywniejszą eksplorację przestrzeni taktyk podczas gromadzenia doświadczeń, co jest pożądane w fazie uczenia.

Ciągły zanik w obu trybach zapewnia płynne przejście od eksploracji do eksploatacji, zapobiegając przedwczesnej konwergencji do suboptymalnych taktyk. Taki schemat eksploracji łączy podejście ϵ-greedy z ideą wielorękich bandytów [5, 28].

Uczenie transferowe i akumulacja wiedzy

Ważną cechą Commander AI jest zdolność do *uczenia się między uruchomieniami* algorytmu (ang. *transfer learning*) [105]:

1. Trwałość Q-tabeli:

Po zakończeniu optymalizacji w trybie treningowym, Q-tabela jest zapisywana do pliku; w trybie benchmarkowym `readonly` zapis jest pomijany:

```
save_qtable(Q, metadata={
    'function_name': function_name,
    'dimension': d,
    'final_fitness': f_best,
    'iterations': T_max,
    'timestamp': current_time
})
```

Przy następnym uruchomieniu treningowym, Q-tabela jest ładowana i kontynuuje uczenie; podczas ewaluacji służy wyłącznie jako zamrożona polityka decyzyjna.

2. Transfer ważony pewnością:

Commander śledzi, ile razy każda para (s, a) została odwiedzona (visit count). Przy transferze wiedzy, nowe wartości Q łączone są z wcześniejszymi z wagami proporcjonalnymi do pewności:

$$\text{confidence}(s,a) = \min\left(1{,}0, \frac{\text{visit\_count}(s,a)}{20}\right) \tag{4.49}$$

$$Q_{\text{new}}(s,a) = \text{confidence}(s,a) \cdot Q_{\text{old}}(s,a) + (1 - \text{confidence}(s,a)) \cdot Q_{\text{init}}(s,a) \tag{4.50}$$

Wartości Q z wysokim confidence (wiele odwiedzin) są bardziej „ufne” i silniej wpływają na nowe optymalizacje.

W benchmarkach finalnych wykorzystano Q-tabelę v7, trenowaną łącznie przez 16000 epizodów (`total_runs` = 16000), zorganizowanych w 2 tury po 8000 sesji każda, gdzie 8000 = 10 funkcji treningowych $\times$ 4 wymiary $\times$ 200 uruchomień. Etykieta `8k` w nazwie pliku (`commander_qtable_v7_8k.pkl`) oznacza rozmiar pojedynczej tury, a nie całego treningu. Zbiór 10 funkcji treningowych jest rozłączny z 33 funkcjami testowymi (szczegóły w Rozdziale 5).

Implementacja Commander AI

Pseudokod uproszczony:

Algorytm 3. Commander AI – pętla decyzyjna (Q-learning)

Require: Q-tabela $Q \in \mathbb{R}^{243\times 6}$, licznik odwiedzin N_v, α=0,1, γ=0,9
Procedura ComputeState(armia, t, $T_{\max}$):
  $p \leftarrow$ dyskretyzuj($t/T_{\max}$, [0,3, 0,7]) {faza: 0,1,2}
  $s \leftarrow$ dyskretyzuj($\bar{c}_{\text{stag}}$, [0,4, 0,8]) {stagnacja: low/moderate/severe}
  $m \leftarrow$ dyskretyzuj(momentum, [0,003, 0,015]) {momentum: stagnant/slow/fast}
  $d \leftarrow$ dyskretyzuj(różnorodność, [0,04, 0,07]) {różnorodność: converged/moderate/spread}
  $h \leftarrow$ dyskretyzuj(hero_ratio, [0, 0,08]) {bohaterowie: none/some/many}
  zwróć state $\leftarrow 81p + 27s + 9m + 3d + h$ {indeks stanu $\in \{0,\ldots,242\}$}
Procedura Step(armia, t, $T_{\max}$):
  $s \leftarrow$ ComputeState(armia, $t, T_{\max}$)
  // Adaptacyjny interwał taktyczny
  $\Delta t \leftarrow$ 5 (wczesna), 8 (środkowa), 15 (późna) {rzadsze zmiany w eksploatacji}
if $t \bmod \Delta t \neq 0$ **then**
    pomiń zmianę taktyki
end if
  // Wybór akcji (ε-greedy z adaptacyjną eksploracją)
  $\varepsilon \leftarrow \max(0{,}03,\ 0{,}12 \cdot (1 - t/T_{\max}))$ {ciągły zanik: 12% → 3%}
if rand() $< \varepsilon$ **then**
    $a \leftarrow$ losowa taktyka z $\{0,\ldots,5\}$
else
    $a \leftarrow \arg\max_{a'} Q[s, a']$
    if $Q[s,a] - Q[s,a_{\text{current}}] < 0{,}1$ **then** zachowaj bieżącą {momentum}
end if
  Zastosuj taktykę a (formacja dobierana automatycznie)
  $f^*_{\text{prev}} \leftarrow f^*$
  Armia wykonuje jedną iterację optymalizacji
  $s' \leftarrow$ ComputeState(armia, t+1, $T_{\max}$)
  // Oblicz nagrodę
  $\Delta_{\text{rel}} \leftarrow (f^*_{\text{prev}} - f^*)/\max(|f^*_{\text{prev}}|, 10^{-9})$
  $r \leftarrow \text{clip}(10{,}0 \cdot \Delta_{\text{rel}},\ -2{,}0,\ 5{,}0) + (-0{,}3 \text{ jeśli } \Delta_{\text{rel}} \leq 0)$ {nagroda fitness + kara stagnacji}
  // Aktualizacja Q-tabeli
  $Q[s,a] \leftarrow Q[s,a] + \alpha\left[r + \gamma \max_{a'} Q[s',a'] - Q[s,a]\right]$
  $N_v[s,a] \leftarrow N_v[s,a] + 1$
Procedura TransferQLearning(Q_{old}, $N_{v,\text{old}}$):
for każda para (s,a) **do**
    $c \leftarrow \min(1,\ N_{v,\text{old}}[s,a]/20)$ {pewność}
    $Q[s,a] \leftarrow c \cdot Q_{\text{old}}[s,a] + (1-c) \cdot Q_{\text{init}}[s,a]$
end for

Commander AI to najwyższy poziom hierarchii w algorytmie ABO – moduł metaoptymalizacji strategii. Poprzez Q-learning [114] generał uczy się, które kombinacje taktyk i formacji działają najlepiej w różnych fazach algorytmu i dla różnych struktur funkcji [76]. Transfer wiedzy między uruchomieniami [105] pozwala algorytmowi poprawiać się z doświadczeniem – każda optymalizacja dostarcza danych uczących, które przyspieszają zbieżność w przyszłych problemach. Dzięki temu ABO jest algorytmem *uczącym się*, nie tylko adaptacyjnym.

4.2.6 System rozpoznania

Architekturę systemu rozpoznania – siatkę dyskretyzacji, struktury danych komórek, metryki zainteresowania (I_g, Równanie 4.14) i niebezpieczeństwa (D_g, Równanie 4.15) – przedstawiono w Sekcji 4.1 (Moduł 4). Poniżej opisano mechanizmy rekomendacji i aktualizacji.

Metryki oceny regionów

Model interpretacyjny oceny regionów – poziom zainteresowania I_g (Równanie 4.14), poziom niebezpieczeństwa D_g (Równanie 4.15) i status eksploracji E_g (Równanie 4.16) – zdefiniowano w Sekcji 4.1 (Moduł 4). W implementacji referencyjnej regiony są rankingowane złożonym wskaźnikiem `score` (Równanie 4.17), łączącym składnik fitness, eksploracji i gradientu z karą za niedawne odwiedziny; na jego podstawie generowane są rekomendacje kierunkowe.

Generowanie rekomendacji

Na podstawie zgromadzonych danych, system rekonesansu generuje rekomendacje kierunkowe dla jednostek:

1. Rekomendacja kierunkowa (4 komponenty):

```
def get_recommendation(pos, expl_vs_exploit):
    # 4 komponenty kierunku (wazone expl_vs_exploit):
    grad_dir      = -global_gradient            # gradient
    promising_dir = weighted_dir(promising_regions)
    explore_dir   = weighted_dir(least_explored[:3])
    elite_dir     = dir_to(elite_regions[0])    # elitarny
    # promising: score = 0.6*fitness + 0.3*explor
    #            + 0.1*grad, kara za recent_visits
    return combine(grad_dir, promising_dir, explore_dir,
                   elite_dir, expl_vs_exploit)
```

2. Rekomendacja zwiadowców:

Gdy jednostka wykonuje taktykę Scouting, może zapytać o pozycję zwiadowcy:

```
def get_scout_recommendation():
    if not self.scouts:
        return np.random.normal(0, 1, self.dim)
    scout = random.choice(self.scouts)
    scout['visits'] += 1
    return scout['position']
```

Adaptacyjne udoskonalanie siatki

W trakcie optymalizacji system udoskonala mapę rekonesansu w dwóch trybach, oba uruchamiane warunkowo dopiero po ustabilizowaniu się przeszukiwania:

Udoskonalanie globalne: Globalne zwiększenie rozdzielczości siatki (`grid_resolution`) ze współczynnikiem $\alpha_g = 1{,}2$ uruchamia się tylko wtedy, gdy $t > 30$, $t \bmod 10 = 0$ oraz globalny fitness spadnie poniżej progu ($f^* < 0{,}1$). Operacja ta resetuje struktury mapy rekonesansu (listę regionów obiecujących oraz komórki wysokiego zainteresowania), zapewniając dokładniejsze mapowanie w końcowej fazie zbieżności.

Udoskonalanie lokalne: Lokalnie system *nie* zagęszcza całej siatki wokół optimum. Przy $t > 30$, $t \bmod 5 = 0$ i $f^* < 0{,}1$ komórki w sąsiedztwie bieżącego najlepszego rozwiązania $\mathbf{x}^*$ są oznaczane jako komórki wysokiego zainteresowania (`high_interest_cells`; promień wyznacza współczynnik $\alpha_l = 2{,}0$). Kieruje to zwiadowców w najbardziej obiecujące rejony bez zmiany globalnej rozdzielczości siatki, przy jednoczesnym zachowaniu globalnego pokrycia.

Skalowalność dla wysokich wymiarów

Dla przestrzeni o $d > 10$ wymiarach, pełna siatka r^d jest niewykonalna. System stosuje następujące strategie:

1. Siatka na pełnej wymiarowości z wizualizacją PCA:

System rozpoznania operuje na siatce w pełnej wymiarowości d (nie stosuje redukcji wymiarowości do obliczeń). Dla celów wizualizacji i diagnostyki, pozycje regionów są rzutowane na 2D za pomocą PCA ($d > 3$). Wewnętrznie deklarowana jest zmienna $d_{\text{reduced}} = \min(d, 3)$, ale nie wpływa ona na logikę śledzenia regionów.

2. Adaptacyjna rozdzielczość:

$$r_d = \min\left(20,\ \max\left(5,\ \lfloor 20 \cdot d^{-1/3} \rfloor\right)\right) \tag{4.51}$$

3. Próbkowanie sąsiedztwa:

Zamiast generować wszystkie 3^d sąsiadów komórki (eksponencjalne), system próbkuje maksymalnie 1000 sąsiadów lub używa tylko sąsiadów wzdłuż osi współrzędnych (liniowe $2d$).

4. Ograniczenie pamięci:

Maksymalna liczba komórek o wysokim zainteresowaniu jest ograniczona do 200, aby zapobiec problemom wydajnościowym.

4.3 INTEGRACJA JEDNOSTEK W ALGORYTM ROJOWY

Sekcje 4.1 i 4.2 przedstawiły architekturę modułową oraz projekt poszczególnych typów jednostek. W tej sekcji pokazano, jak te autonomiczne komponenty są integrowane w algorytm rojowy – jak komunikują się między sobą, koordynują działania i wspólnie realizują optymalizację. Aby heterogeniczna armia jednostek

działała jako *kolektywna inteligencja* [18], a nie zbiór niezależnych agentów [120], potrzebne są trzy mechanizmy: (1) protokoły komunikacji wymieniające informacje między jednostkami, (2) strategie koordynacji synchronizujące działania, (3) hierarchiczny system zarządzania nadzorowany przez Commander AI.

4.3.1 Mechanizmy komunikacji

W algorytmach rojowych [68, 37], jednostki nie mają pełnej wiedzy o przestrzeni poszukiwań – muszą dzielić się informacjami, aby kolektywnie osiągnąć optimum. Ancient Battlefield Optimizer implementuje cztery mechanizmy komunikacji: globalną wymianę najlepszych rozwiązań, hierarchiczne nadawanie przez bohaterów, pośrednią komunikację przez system rekonesansu oraz sygnalizację jakości przez system honorów.

Protokoły wymiany informacji

1. Komunikacja globalna – Global Best Sharing:

Najbardziej fundamentalny protokół: każda jednostka ma dostęp do globalnie najlepszego rozwiązania $\mathbf{g}$:

$$\mathbf{g} = \arg \min_{\mathbf{x} \in \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}} f(\mathbf{x}) \tag{4.52}$$

gdzie $\mathbf{p}_i$ to personal best jednostki i. Aktualizacja $\mathbf{g}$ jest atomiczna – gdy jednostka i znajduje rozwiązanie lepsze niż $\mathbf{g}$, natychmiast aktualizuje globalną zmienną:

```
if unit.best_fitness < self.global_best_fitness:
    self.global_best_fitness = unit.best_fitness
    self.global_best_position = unit.best_position.copy()
    global_improved = True
```

Gdy $\mathbf{g}$ się zmienia, armia aktualizuje honor wszystkich jednostek i propaguje wiedzę bohaterów poprzez bezpośrednie wywołanie `share_knowledge()`.

2. Komunikacja hierarchiczna – Hero Broadcasting:

Jednostki o statusie bohatera, dzięki wysokiemu parametrowi wpływu mogą nadawać swoje odkrycia do szerszej grupy jednostek. Gdy bohater znajdzie obiecujący region:

```
if self.fitness < global_best_fitness * 0.95:  # znaczna poprawa
    # Nadaj sygnał do wszystkich w promieniu wpływu
    influenced_units = [
        u for u in army.units
        if norm(u.position - self.position) < self.influence_radius]
    for unit in influenced_units:
        unit.receive_hero_signal(self.position, self.fitness, self.unit_id)
```

Jednostki odbierające sygnał bohatera mogą tymczasowo zwiększyć atrakcję do jego pozycji:

$$\mathbf{v}_{\text{hero\_influence}} = c_{\text{hero}} \cdot r \cdot (\mathbf{x}_{\text{hero}} - \mathbf{x}_i) \tag{4.53}$$

gdzie c_{hero} zależy od honoru bohatera i odległości, a r jest zmienną losową.

3. Rekonesans

System rekonesansu działa jako *pośredni* kanał komunikacji – jednostki nie komunikują się bezpośrednio, ale zostawiają informacje w przestrzeni:

$$I_g(\mathbf{c}) = f(I_g^{\text{old}}(\mathbf{c}), \{\text{odwiedziny jednostek w komórce } g\}) \tag{4.54}$$

Każda jednostka odwiedzająca komórkę g zapisuje swój fitness, a rekonesans agreguje te informacje w poziomy zainteresowania (I_g) i zagrożenia (D_g). Inne jednostki mogą czytać te informacje i kierować swoją eksplorację.

4. Honor System jako sygnał jakości:

Honor jednostki jest publiczną informacją dostępną dla Commander AI. Wysoki honor sygnalizuje, że jednostka skutecznie znajduje dobre rozwiązania, co wpływa na decyzje Commander AI dotyczące wyboru taktyk i formacji.

4.3.2 Koordynacja i synchronizacja

Komunikacja pozwala jednostkom wymieniać informacje, ale *koordynacja* zapewnia, że działają one w sposób skoordynowany, a nie chaotyczny. ABO implementuje kilka mechanizmów koordynacji na różnych poziomach.

Strategie koordynacji działań

1. Koordynacja oparta na taktykach:

Commander AI wybiera globalną strategię (np. Phalanx, Flanking), która determinuje zachowanie wszystkich jednostek:

```
action = commander.select_action(state)  # eps-greedy
commander.apply_action(action)  # taktyka + formacja

# Wszystkie jednostki stosują tę samą taktykę
commander.current_tactic.apply(army.units, global_best,
                               iteration, max_iterations)
```

Na przykład, taktyka Phalanx sprawia, że wszystkie jednostki tworzą gęstą formację wokół globalnego best:

$$\mathbf{x}_i^{\text{target}} = \mathbf{g} + \text{formation\_offset}_i, \quad \|\text{formation\_offset}_i\| < r_{\text{phalanx}} \tag{4.55}$$

Formacja ta koordynuje jednostki do intensywnej eksploatacji lokalnej.

Taktyka Flanking koordynuje jednostki do ataku z wielu stron:

$$\mathbf{x}_i^{\text{target}} = \mathbf{g} + R_i \cdot \mathbf{d}_{\text{random}}, \quad R_i > r_{\text{flank}} \tag{4.56}$$

gdzie R_i jest dużym promieniem – jednostki otaczają region wokół $\mathbf{g}$ z różnych kierunków.

2. Koordynacja przestrzenna oparta na formacjach:

Formacje (wedge, line, circle) koordynują przestrzenne rozmieszczenie jednostek:

$$\mathbf{x}_i = \mathbf{x}_i^{\text{optimized}} + \tau_f \cdot \mathbf{o}_i^{\text{formation}} \tag{4.57}$$

gdzie $\mathbf{x}_i^{\text{optimized}}$ to pozycja wynikająca z algorytmu optymalizacji, $\mathbf{o}_i^{\text{formation}}$ to przesunięcie formacyjne, a $\tau_f \in [0, 1]$ to siła formacji (typowo $\tau_f = 0{,}1$).

3. Koordynacja oparta na fazach:

W implementacji ABO nie stosuje się bezpośredniego mnożenia parametrów eksploracji/eksploatacji w zależności od fazy. Zamiast tego, Commander AI oparty na Q-learningu (Sekcja 4.2.5) *autonomicznie* dobiera taktykę w punktach decyzyjnych wyznaczanych przez adaptacyjny interwał (co 5/8/15 iteracji, zob. niżej), reagując na bieżący stan optymalizacji. Faza ($p = t/T$) jest jedną z pięciu cech stanu s, ale to Q-learning decyduje, jakie zachowanie jest optymalne:

```
state = commander.get_state(iteration, max_iterations)
action = commander.select_action(state)  # eps-greedy
commander.apply_action(action)           # taktyka + formacja

# Taktyka ustawia _tactic_params per jednostke:
commander.current_tactic.apply(army.units, global_best,
                               iteration, max_iterations)
# np. FlankingTactic -> kawaleria:
#     levy_alpha=1.2, charge_distance_scale=2.0
# -> lekka piechota: de_f_range=(0.8,1.5), de_cr=0.9
```

Interwał zmiany taktyk jest adaptacyjny: co 5 iteracji na początku ($p < 0{,}3$), co 8 w fazie przejściowej ($0{,}3 \leq p < 0{,}7$), co 15 pod koniec ($p \geq 0{,}7$). Stabilizacja zapobiega zbyt częstym zmianom: nowa taktyka jest stosowana tylko gdy $Q[s', a_{new}] - Q[s', a_{prev}] \geq 0{,}1$.

4.3.3 System zarządzania rojowego

Koordynacja i komunikacja zapewniają spójność na poziomie operacyjnym, ale *strategiczne* decyzje podejmowane są przez system zarządzania – Commander AI wraz z mechanizmami monitorowania i adaptacji.

Rola jednostki przywódczej

Commander AI – realizujący koncepcję hierarchicznego dowodzenia [110, 111] – pełni dwie główne role:

1. Strateg – Wybór taktyk i formacji:

Commander decyduje, jakie strategie algorytm powinien stosować w danym momencie (szczegóły w Sekcji 4.2.4):

```
state = commander.get_state(iteration, max_iterations)
action = commander.select_action(state)  # Q-learning
commander.apply_action(action)  # taktyka + formacja
```

2. Obserwator – Monitorowanie stanu armii:

Commander śledzi metryki stanu algorytmu (m.in. różnorodność, stagnację, tempo poprawy), z których składa pięciowymiarowy stan s używany w Q-learningu (Sekcja 4.2.5).

Opisane mechanizmy komunikacji i koordynacji – wymiana globalnego najlepszego rozwiązania, propagacja wiedzy bohaterów, adaptacyjne sterowanie Commander AI – łączą jednostki w algorytm rojowy.

4.3.4 Pełny pseudokod nowego algorytmu CommanderABO

Opisane wcześniej komponenty – heterogeniczne jednostki, system taktyk i formacji, Commander AI z Q-learningiem, system rekonesansu i honorów – łączą się w jedną pętlę optymalizacyjną. Algorytm 4 przedstawia pełny pseudokod CommanderABO, integrujący wszystkie moduły w zunifikowany algorytm rojowy.

Algorytm 4. CommanderABO – pełny algorytm optymalizacyjny

Require: Funkcja celu $f : \mathbb{R}^d \to \mathbb{R}$, granice $[\mathbf{b}_{\min}, \mathbf{b}_{\max}]$, maks. iteracji $T_{\max}$
Require: Skład armii: $N = 40$ (HI:8, LI:8, Ar:7, Cav:8, Ch:5, WE:4)
Require: Q-tabela Commander AI; flaga `readonly` określająca tryb benchmarkowy lub treningowy
Ensure: Najlepsza znaleziona pozycja $\mathbf{g}$ i fitness $f_{\mathbf{g}}$
// Inicjalizacja
Utwórz armię: $\mathcal{P} = \bigcup_{t \in \mathcal{T}} \mathcal{P}_t$
Zainicjalizuj pozycje jednostek metodą CoordinateStratifiedInit; ewaluuj $f_i \leftarrow f(\mathbf{x}_i)$
$i^* \leftarrow \arg\min_i f_i$; $\mathbf{g} \leftarrow \mathbf{x}_{i^*}$; $f_{\mathbf{g}} \leftarrow f_{i^*}$
Zainicjalizuj rekonesans: siatka o rozdzielczości $r = \min(20, \max(5, \lfloor 20 \cdot d^{-1/3} \rfloor))$
Zainicjalizuj FormationManager; ustaw taktykę początkową (Phalanx)
$a_{\text{prev}} \leftarrow$ null; $s_{\text{prev}} \leftarrow$ null; $f_{\text{prev}} \leftarrow f_{\mathbf{g}}$
// Pojedyncza główna pętla optymalizacyjna
for $t = 0$ **to** $T_{\max} - 1$ **do**
    // 1. Aplikacja bieżącej taktyki – ustawienie parametrów ruchu jednostek
    army.apply_tactic(t, $T_{\max}$)
    if $t \bmod 5 = 0$ **then**
        FormationManager.organize_units() {reorganizacja grup zgodna z implementacją}
    end if
    // 2. Commander AI: obserwacja, nagroda za poprzedni interwał i ewentualna zmiana taktyki

$\Delta t \leftarrow 5$ jeśli $t/T_{\max} < 0{,}3$, 8 jeśli $t/T_{\max} < 0{,}7$, w przeciwnym razie 15
decision_point $\leftarrow (t \bmod \Delta t = 0) \vee (t = 0)$
if decision_point **then**
  $s \leftarrow$ compute_state(progress, stagnation, momentum, diversity, hero_ratio)
  if $s_{\text{prev}} \neq$ null **and** $a_{\text{prev}} \neq$ null **then**
    $r \leftarrow$ compute_reward($f_{\text{prev}}, f_{\mathbf{g}}$)
    if `readonly`=False **then**
      $Q(s_{\text{prev}}, a_{\text{prev}}) \leftarrow Q(s_{\text{prev}}, a_{\text{prev}}) + \alpha[r + \gamma \max_{a'} Q(s, a') - Q(s_{\text{prev}}, a_{\text{prev}})]$
      Zapisz doświadczenie $(s_{\text{prev}}, a_{\text{prev}}, r, s)$; wykonaj replay, jeśli pamięć ≥ 32
    end if
  end if
  $a \leftarrow$ wybierz taktykę metodą ε-greedy z $Q(s,\cdot)$
  if $a_{\text{prev}} \neq$ null **and** $Q(s,a) - Q(s,a_{\text{prev}}) < 0{,}1$ **then**
    $a \leftarrow a_{\text{prev}}$ {stabilizacja taktyczna}
  end if
  $\tau \leftarrow$ tactic_pool$[a]$; $\phi \leftarrow$ tactic_formation_map$[a]$
  army.set_tactic(τ); Commander.apply_formation_template(ϕ)
  $s_{\text{prev}} \leftarrow s$; $a_{\text{prev}} \leftarrow a$; $f_{\text{prev}} \leftarrow f_{\mathbf{g}}$
else
  $a \leftarrow a_{\text{prev}}$ {zachowanie bieżącej taktyki}
end if
// 3. Cele formacyjne i komunikacja bohaterów
FormationManager.update_targets($\mathbf{g}$, army.tactic.name, t, $T_{\max}$)
unit_targets $\leftarrow$ FormationManager.calculate_unit_target_positions($\mathcal{P}$)
army.share_knowledge()
// 4. Aktualizacja pozycji wszystkich aktywnych jednostek
for $i = 1$ **to** N **do**
  if u_i nieaktywna (zbieg) **then**
    continue
  end if
  MoveUnit(u_i, army, f, t, $T_{\max}$, unit_targets) {strategia zależna od typu jednostki}
  Reconnaissance.update_with_position($\mathbf{x}_i$, f_i, t, $T_{\max}$)
end for
// 5. Aktualizacja globalnego optimum i honorów
$i^* \leftarrow \arg\min_i f_i^{pbest}$
if $f_{i^*}^{pbest} < f_{\mathbf{g}}$ **then**
  $\mathbf{g} \leftarrow \mathbf{x}_{i^*}^{pbest}$; $f_{\mathbf{g}} \leftarrow f_{i^*}^{pbest}$
end if
Aktualizuj honor jednostek oraz statusy hero/runagate
if $t > 30$ **and** $f_{\mathbf{g}} < 0{,}1$ **and** rekonesans aktywny **then**
  if $t \bmod 10 = 0$ **then**
    refine_grid_globally(1,2)
  end if
  if $t \bmod 5 = 0$ **then**
    refine_grid_around_best($\mathbf{g}$, 2,0)
  end if
end if
end for
return $\mathbf{g}$, $f_{\mathbf{g}}$

Algorytm 4 uwidacznia główną cechę architektury: separację poziomu *strategicznego* (Commander AI: obserwacja stanu, wybór taktyki i ewentualna aktualizacja Q), poziomu *taktycznego* (aplikacja parametrów ruchu i formacji) oraz poziomu *operacyjnego* (ruch jednostek i aktualizacja najlepszego rozwiązania). Commander podejmuje decyzje w adaptacyjnym interwale 5/8/15 iteracji, taktyka modyfikuje parametry jednostek, a same jednostki autonomicznie przeszukują przestrzeń rozwiązań. Ta hierarchiczna struktura odzwierciedla organizację starożytnych armii [95, 45], w których generał wydawał rozkazy strategiczne, bohaterowie/oficerowie taktyczni koordynowali formacje, a żołnierze walczyli samodzielnie na swoich pozycjach.

4.3.5 Przykład działania algorytmu – przejście jednej iteracji

Aby zilustrować współdziałanie opisanych mechanizmów, prześledzono jedną pełną iterację algorytmu CommanderABO na uproszczonym przykładzie: 5 agentów minimalizuje funkcję Rastrigin w przestrzeni 2D ($[-5{,}12,\ 5{,}12]^2$; optimum globalne: $f(\mathbf{0}) = 0$).

Stan początkowy (iteracja $t = 5$). Populacja po 5 iteracjach wygląda następująco:

Tabela 7. Stan populacji w przykładzie po 5 iteracjach (pozycje, fitness i honor agentów)

Agent	Typ	Pozycja	Fitness	Honor
u_1	HeavyInfantry	(1,2, −0,8)	8,93	2,1
u_2	LightInfantry	(−2,5, 3,1)	25,41	−1,3
u_3	Cavalry	(0,3, 0,5)	2,47	4,8
u_4	Archers	(−1,1, 2,0)	14,62	0,7
u_5	Chariots	(3,8, −1,9)	32,17	−3,5

Najlepsze dotychczasowe rozwiązanie globalne: $\mathbf{g} = (0{,}3,\ 0{,}5)$ z fitness 2,47 (znalezione przez kawalerię u_3).

Krok 1: Decyzja Commander AI. Ponieważ $t = 5$ i $\tau_{\text{interval}} = 5$, Commander podejmuje decyzję. Oblicza stan: średnia względna poprawa fitness wyniosła ok. 1% na iterację (kategoria: *wolne*, przedział [0,003; 0,015)), stagnacja jest niska (4 z 5 agentów poprawiło wynik w bieżącej iteracji, a licznik kawalerii wynosi 1, więc $\overline{c} = 0{,}2 < 0{,}4$), różnorodność populacji średnia, bohaterów brak. Stan dyskretyzowany: $s = (0, 0, 1, 1, 0)$ – faza wczesna ($t/T = 0{,}05$) – co daje indeks $81 \cdot 0 + 27 \cdot 0 + 9 \cdot 1 + 3 \cdot 1 + 0 = 12$ w tabeli Q.

ε-zachłanna strategia: $\varepsilon(5) = 0{,}11$, wylosowano rand() $= 0{,}42 > \varepsilon$ – Commander korzysta z tabeli Q. Najwyższa wartość $Q(12, a)$ wskazuje akcję $a = 3$ (`Flanking`). Taktyka: oskrzydlenie; formacja: `wedge` (klin).

Krok 2: Modyfikacja parametrów (taktyka Flanking). Taktyka `Flanking` zwiększa parametr eksploracji kawalerii (`charge_distance_scale` = 2,0) i ustawia kierunek szarży prostopadle do wektora do **g**.

Krok 3: Ruch jednostek. Każdy agent porusza się zgodnie ze swoją strategią:

- u_1 (HeavyInfantry): przeszukiwanie współrzędnościowe – testuje (1,2+0,3, −0,8) = (1,5, −0,8) z fitness 10,12 (gorsza) i (1,2, −0,8+0,2) = (1,2, −0,6) z fitness 7,15 (lepsza). Przyjmuje nową pozycję (1,2, −0,6).
- u_2 (LightInfantry): mutacja DE/rand/1 – wybiera 3 losowe rozwiązania, oblicza wektor próbny $\mathbf{v} = (-1{,}8,\ 2{,}3)$ z fitness 16,50 (lepsza niż 25,41). Zachłanna selekcja: przyjmuje.
- u_3 (Cavalry): lot Lévy'ego z $\alpha = 1{,}5$ – losuje skok o długości $L = 4{,}2$, kierunek prostopadły do **g** (taktyka Flanking): $\mathbf{d}_\perp = (0{,}86,\ -0{,}51)$. Nowa pozycja: $(0{,}3 + 4{,}2 \cdot 0{,}86,\ 0{,}5 + 4{,}2 \cdot (-0{,}51)) = (3{,}91,\ -1{,}64)$ z fitness 34,80 (gorsza – agresywna eksploracja nie zawsze się opłaca).
- u_4 (Archers): salwa 4 strzałów w otoczeniu – najlepszy wynik: (−0,9, 1,7) z fitness 11,35 (lepsza).
- u_5 (Chariots): momentum Cauchy'ego – nowa pozycja (2,9, −1,2) z fitness 20,55 (lepsza).

Krok 4: Aktualizacja honoru. Żaden agent nie poprawił globalnego optimum, więc honor wszystkich agentów maleje zgodnie z formułą z Sekcji 4.1: zanik $\gamma_H = 0{,}95$ oraz kara $-(0{,}2 + 0{,}1 \cdot \min(5, c_i))$:

Tabela 8. Aktualizacja honoru agentów w przykładowej iteracji

Agent	Stary honor	Kara	Nowy honor	Status
u_1	2,1	−0,2	1,80	zwykły
u_2	−1,3	−0,2	−1,44	zwykły
u_3	4,8	−0,3	4,26	zwykły
u_4	0,7	−0,2	0,47	zwykły
u_5	−3,5	−0,2	−3,53	zwykły

Agent u_3 (kawaleria) pogorszył swój fitness, ale zachowuje najlepsze historyczne rozwiązanie $\mathbf{p}_3 = (0{,}3,\ 0{,}5)$. Globalne optimum **g** nie zmienia się w tej iteracji.

Krok 5: Aktualizacja rekonesansu. Mapa siatki rekonesansu aktualizuje wskaźniki I_g i D_g na podstawie nowych pozycji. Region wokół $(0, 0)$ ma wysoki I_g (znaleziono tam najlepszy fitness) i niski D_g (odwiedzany umiarkowanie). W następnych iteracjach jednostki będą kierowane do tego regionu.

Krok 6: Nagroda Q-learning. Nagroda: $r = -0{,}3$ (globalne optimum nie uległo poprawie – kara za stagnację). Aktualizacja: $Q(12, 3) \leftarrow Q(12, 3) + 0{,}1 \cdot [-0{,}3 + 0{,}9 \cdot \max_{a'} Q(s', a') - Q(12, 3)]$. Taktyka Flanking otrzymuje w stanie 12 niewielkie ujemne wzmocnienie.

Podsumowanie iteracji: 4 z 5 agentów poprawiło fitness. Kawaleria podjęła ryzykowną eksplorację (duży skok Lévy'ego), która nie opłaciła się w tej iteracji, ale pozwoliła na zbadanie nowego regionu. Commander AI zapamiętał, że taktyka Flanking nie przyniosła w stanie 12 poprawy globalnego optimum (kara $-0{,}3$) – ta informacja będzie użyta w przyszłych decyzjach.

4.3.6 Złożoność obliczeniowa

Przed przejściem do empirycznej weryfikacji algorytmu warto sformalizować jego koszt obliczeniowy. Przyjęto oznaczenia: N – rozmiar populacji (domyślnie 40), D – wymiarowość problemu, T – liczba iteracji (100 w głównym benchmarku), τ – okres decyzyjny Commander AI (domyślnie 5), $|S|$ – liczność dyskretnej przestrzeni stanów Q-tabeli (5 wymiarów stanu $\times$ 3 poziomy dyskretyzacji $= 3^5 = 243$ stanów; łącznie $|S| \cdot |A| = 243 \cdot 6 = 1458$ komórek Q-tabeli), G – rozdzielczość siatki rekonesansu ($G = \min(20,\ \max(5, \lfloor 20 \cdot D^{-1/3} \rfloor))$).

Główna pętla iteracyjna. Każda iteracja $t \in \{1, \ldots, T\}$ wykonuje pięć faz o następujących kosztach dominujących:

1. **Ewaluacja fitness**: N wywołań funkcji celu f. Koszt: $O(N \cdot C_f)$, gdzie C_f to koszt jednej ewaluacji funkcji testowej (typowo $O(D)$).

2. **Decyzja Commander AI** (co τ iteracji): obliczenie wektora stanu $s \in \mathbb{Z}^5$, wybór akcji ε-greedy, aktualizacja $Q(s,a)$. Koszt amortyzowany: $O(|A|/\tau)$ na iterację.

3. **Aktualizacja taktyk i formacji**: zmiana parametrów N jednostek na podstawie wybranej taktyki. Koszt: $O(N \cdot D)$.

4. **Ruch jednostek**: aktualizacja pozycji każdej jednostki, obejmująca składowe inercji, formacji, eksploracji (Lévy lub Gauss). Koszt: $O(N \cdot D)$.

5. **Rekonesans i honor**: mapowanie pozycji na komórki siatki G^D (bez materializacji pełnej siatki, hash $O(D)$ per agent), aktualizacja honorów. Koszt: $O(N \cdot D)$.

Złożoność asymptotyczna

Sumaryczny koszt jednej iteracji to $O(N \cdot (C_f + D)) + O(|A|/\tau)$, co przy stałej liczbie taktyk $|A| = 6$ redukuje się do $O(N \cdot (C_f + D))$. Łączny koszt T iteracji wynosi:

$$\mathcal{T}_{\text{ABO}}(N, D, T) \;=\; O\big(T \cdot N \cdot (C_f + D)\big) + O\big(\tfrac{T}{\tau} \cdot |S| \cdot |A|\big) + O(T \cdot N \cdot D). \quad (4.58)$$

Pierwszy człon to koszt ewaluacji funkcji i ruchu jednostek (dominujący); drugi człon, $O((T/\tau) \cdot |S| \cdot |A|) = O((100/5) \cdot 1458) = O(29160)$, odpowiada za amortyzowane operacje Commander AI (aktualizacja Q-tabeli, wybór akcji ε-greedy); trzeci człon obejmuje rekonesans z rzadką hash-mapą siatki G^D. Dla typowych funkcji testowych $C_f = O(D)$, więc po pominięciu stałych:

$$\mathcal{T}_{\text{ABO}} \;=\; O(T \cdot N \cdot D). \quad (4.59)$$

Komponenty Commander AI i rekonesansu wnoszą pomijalne narzuty asymptotyczne ($|S| \cdot |A| = 1458$ jest stałą niezależną od D, a rzadka reprezentacja siatki rekonesansu utrzymuje koszt $O(N \cdot D)$ zamiast $O(G^D)$).

Złożoność pamięciowa

Stan trwały algorytmu obejmuje: populację jednostek ($O(N \cdot D)$ na pozycje, prędkości, najlepsze osobiste), tabelę Q ($O(|S| \cdot |A|) = O(1458)$, czyli $O(1)$ względem D), mapę rekonesansu (rzadka reprezentacja hash-mapy, $O(N)$ aktywnych komórek per iteracja). Łączny koszt pamięci: $O(N \cdot D)$.

Porównanie z algorytmami referencyjnymi

Dla *podstawowej* aktualizacji pozycji złożoność CommanderABO jest tego samego rzędu $O(T \cdot N \cdot D)$ co klasyczne algorytmy rojowe (PSO, GWO, WOA, DE) dominujące zestaw porównawczy. Należy jednak podkreślić, że pełny CommanderABO wnosi dodatkowe składniki kosztu – rekonesans, system taktyk i formacji, Q-learning oraz historię – a warianty z aktywnym lokalnym przeszukiwaniem wykonują dodatkowe wywołania funkcji celu. Składnikowo koszt pojedynczej iteracji można zapisać jako

$$O\big(N\,(C_f + D) \;+\; C_{\text{recon}} \;+\; C_{\text{local}} \;+\; |A|/\tau\big), \quad (4.60)$$

gdzie C_{recon} to koszt rekonesansu, a C_{local} – koszt lokalnego przeszukiwania (zerowy, gdy jest ono wyłączone). Różnice w empirycznym czasie wykonania (patrz Sekcja 6.7.4) wynikają ze stałych ukrytych w notacji O oraz z tych dodatkowych składników: ABO/CommanderABO wykonuje per iterację dodatkową logikę taktyk, formacji i Commander AI, choć narzut samego Commander jest zamortyzowany przez okres τ. W rezultacie rzeczywisty czas wykonania CommanderABO wynosi w medianie 1,37 s (#37/46 w benchmarku), w porównaniu z SA (#1, 0,02 s) i BFO (#46, 14,44 s).

Skalowalność

Z wzoru $\mathcal{T}_{\text{ABO}} = O(T \cdot N \cdot D)$ wynika, że dwukrotne zwiększenie wymiarowości lub populacji powinno dwukrotnie wydłużyć czas wykonania. Empiryczne pomiary (Sekcja 6.7) potwierdzają liniową skalowalność względem D dla $D \in \{2,10,30,50,100\}$. Stała N jest ustalona na 40 agentów dla wszystkich wymiarów – zwiększenie populacji w wysokich wymiarach (heurystyka $N \propto D$ spotykana w innych algorytmach) nie było stosowane.

Algorytm ABO spełnia ogólne warunki wystarczające zbieżności metaheurystyk stochastycznych – elityzm (zachowanie najlepszego dotychczas rozwiązania) oraz niezerowe prawdopodobieństwo eksploracji w każdej iteracji [94, 100] – których omówienie wraz z empiryczną weryfikacją przedstawiono w Sekcji 6.6. Adaptacyjny moduł Commander AI nie narusza tych warunków, ponieważ utrzymuje stałą, dodatnią stopę eksploracji ($\epsilon_{\min} > 0$), zgodnie z warunkami zbieżności Q-learningu [99, 60].

Formalizacja algorytmiczna jest skończona – zdefiniowano sześć typów jednostek z precyzyjnymi równaniami ruchu, sześć taktyk z parametryzacją formacji, system rozpoznania z dyskretyzacją przestrzeni, hierarchię honorów oraz Commander AI z Q-learningiem. Pozostaje jednak pytanie implementacyjne: jak te matematyczne modele przekładają się na działający kod? Jakie decyzje architektoniczne trzeba podjąć? Jakie adaptacje są potrzebne, gdy wymiarowość problemu rośnie z 2D do 100D? Kolejny rozdział opisuje przejście od teorii do implementacji.

5. REALIZACJA I ŚRODOWISKO EKSPERYMENTALNE

„Czego nie potrafię stworzyć, tego nie rozumiem."
„What I cannot create, I do not understand." (ang.)
– Richard Feynman

Każdy wcześniej opisany komponent ma swoje matematyczne formuły i pseudokod. Algorytm zapisany formalnie wymaga jednak jeszcze realizacji programistycznej. Jak pisał Helmuth von Moltke: „Żaden plan nie przetrwa kontaktu z nieprzyjacielem". W przypadku algorytmów optymalizacyjnych takim sprawdzianem są ograniczenia pamięci, wydajność obliczeniowa, błędy implementacyjne i przypadki brzegowe.

Ten rozdział opisuje praktyczną realizację ABO – przejście od matematycznych formuł do kodu, od diagramów UML do działających klas i funkcji. Sekcja 5.1 omawia wybór technologii, architekturę oprogramowania i integrację modułów. Sekcja 5.2 opisuje środowisko eksperymentalne (funkcje testowe, system analizy trudności). Sekcja 5.3 definiuje metodologię testową, w tym zastosowane testy statystyczne (pełny przewodnik po testach statystycznych zamieszczono w osobnym suplemencie metodologicznym dołączonym do rozprawy).

5.1 REALIZACJA ALGORYTMÓW

Niniejsza sekcja opisuje decyzje technologiczne i architektoniczne podjęte podczas implementacji algorytmu ABO.

5.1.1 Środowisko programistyczne

Uzasadnienie wyboru języka Python

Algorytm ABO zrealizowano w języku **Python** z następujących powodów:

1. Ekosystem naukowy i optymalizacyjny:

Python ma rozbudowany ekosystem bibliotek do obliczeń naukowych i optymalizacji:

- **NumPy** [54] – wydajne operacje na tablicach wielowymiarowych, podstawa wszelkich obliczeń numerycznych
- **SciPy** [113] – zaawansowane algorytmy optymalizacji, interpolacji, analizy statystycznej
- **Matplotlib** [59], **Plotly** – wizualizacja 2D i 3D wyników optymalizacji
- **opfunu** [107] – biblioteka z 100+ funkcjami testowymi do benchmarkingu
- **mealpy** [108] – implementacje ponad 100 algorytmów metaheurystycznych do porównań

Alternatywne języki (C++, Java, MATLAB) wymagałyby realizacji wielu komponentów od podstaw.

2. Szybkość prototypowania:

Dynamiczna typizacja i zwięzła składnia języka Python pozwalają na szybką iterację i eksperymentowanie z różnymi wariantami algorytmu.

3. Interfejsy użytkownika:

Framework **Streamlit** [103] pozwala tworzyć interaktywne aplikacje webowe w czystym Python, bez znajomości HTML/CSS/JavaScript.

4. Wydajność wystarczająca:

Choć Python jest językiem interpretowanym, wykorzystanie NumPy (napisanego w C/Fortran) oraz JIT compilation (Numba) zapewnia wydajność porównywalną z językami kompilowanymi dla operacji numerycznych. W testach, ewaluacja funkcji celu zajmuje >90% czasu wykonania – tutaj Python i C++ mają identyczną wydajność (oba wywołują tę samą funkcję).

Środowisko sprzętowe

Eksperymenty przeprowadzono w dwóch środowiskach:

- **Rozwój i testy lokalne:** Apple MacBook Pro z procesorem Apple M3 Pro (12 rdzeni: 6 wydajnych + 6 energooszczędnych), 36 GB RAM, macOS Sonoma. Środowisko lokalne służyło do prototypowania, debugowania i mniejszych eksperymentów.
- **Benchmarki produkcyjne:** Platforma chmurowa **Modal** (`https://modal.com`) – serverless compute z automatycznym skalowaniem. Każde uruchomienie algorytmu wykonywane było jako izolowany kontener z 2 vCPU i 4 GB RAM, co zapewniło reprodukowalność i eliminację wpływu współdzielenia zasobów. Benchmarki z Rozdziału 6 przeprowadzono na tej platformie.

5.1.2 Architektura oprogramowania

Architektura koncepcyjna algorytmu ABO – typy jednostek, taktyki, formacje, rozpoznanie i Commander AI – jest opisana w Rozdziale 4. Niniejsza sekcja skupia się na realizacji programistycznej tej architektury: strukturze pakietów, hierarchii klas, wzorcach projektowych i decyzjach technicznych.

5.1.3 Wykorzystane technologie

Realizacja ABO opiera się na ekosystemie Python, który łączy dojrzałe biblioteki numeryczne, frameworki do benchmarkingu metaheurystyk oraz narzędzia do wizualizacji i profilowania. Wybór poszczególnych bibliotek podyktowany był potrzebą reprodukowalności eksperymentów, dostępnością implementacji referencyjnych algorytmów porównawczych oraz wydajnością obliczeń wektorowych. Tabela 9 zestawia zastosowane biblioteki wraz z ich rolą w projekcie.

Tabela 9. Stos technologiczny realizacji ABO

Biblioteka	**Przeznaczenie**	**Uzasadnienie**
NumPy 1.24.0 [54]	obliczenia numeryczne	wektoryzacja operacji na populacji, macierze Q-tabeli
opfunu 1.0.4 [107]	funkcje testowe	100+ nazwanych funkcji benchmarkowych (CEC, unimodalne, multimodalne)
mealpy 3.0.3 [108]	algorytmy porównawcze	wiele implementacji metaheurystyk (GA, DE, WOA, GWO, HHO)
SciPy 1.10.0 [113]	statystyka i optymalizacja	testy statystyczne (Wilcoxon, Friedman), KD-tree, Nelder-Mead
Matplotlib [59] / Plotly / Seaborn	wizualizacja	wykresy konwergencji 2D, wizualizacje 3D, porównania statystyczne
Streamlit 1.28+ [103]	interfejs webowy	interaktywny dashboard bez HTML/CSS/JS
logging / tqdm	monitorowanie	strukturalne logi, paski postępu dla długich eksperymentów

5.1.4 Integracja komponentów

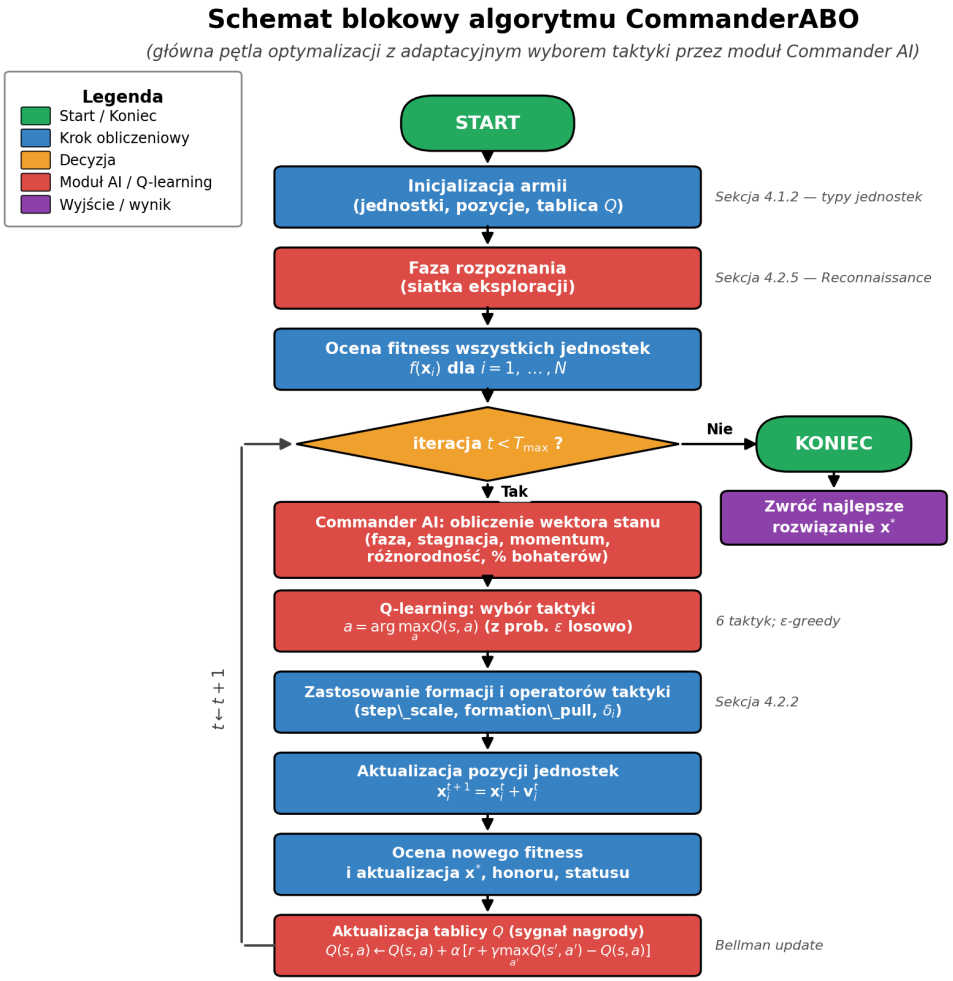
Poniżej opisano integrację komponentów ABO: główną pętlę optymalizacyjną, paralelizację, wektoryzację oraz testowanie.

Główna pętla algorytmu

Pełny pseudokod podano w Załączniku A (Algorytm 5). W każdej iteracji główna pętla wykonuje: (1) bieżąca taktyka modyfikuje parametry jednostek, (2) Commander AI obserwuje stan i w adaptacyjnym interwale może zmienić taktykę oraz formację, (3) FormationManager aktualizuje pozycje docelowe, (4) bohaterowie dzielą się wiedzą z sąsiednimi jednostkami, (5) jednostki aktualizują pozycje,

(6) system honorów promuje lub degraduje jednostki, (7) Q-tabela jest aktualizowana i zapisywana tylko w trybie treningowym; w benchmarku `readonly` pozostaje zamrożona.

Rysunek 7 przedstawia schemat blokowy algorytmu CommanderABO, ilustrując przepływ sterowania między głównymi komponentami oraz pętlę adaptacyjnego wyboru taktyki realizowaną przez moduł Commander AI.



Rys. 7. Schemat blokowy algorytmu CommanderABO. Kolorystyka rozróżnia rodzaje bloków: zielony – start/koniec, niebieski – klasyczne kroki obliczeniowe, pomarańczowy – decyzja, czerwony – moduły AI (Reconnaissance, Commander AI, aktualizacja Q-tablicy), fioletowy – wyjście. Pętla $t \leftarrow t + 1$ obrazuje powrót sterowania do warunku zakończenia po każdej iteracji. Wektor stanu Commander AI ma wymiar 5 (faza, stagnacja, momentum, różnorodność, odsetek bohaterów), co przy podziale na 3 poziomy daje $|\mathcal{S}| = 243$ stany; zbiór akcji $|\mathcal{A}| = 6$ odpowiada sześciu taktykom (Sekcja 4.2.5).

5.1.5 Adaptacje wysokowymiarowe

Standardowe operatory przeszukiwania, zaprojektowane i kalibrowane na problemach 2D–10D, wykazują degradację w wymiarach 30D–100D z trzech powodów: (1) wektory szumu (gaussowskie, Cauchy'ego) rosną jak $\sqrt{d}$ w normie euklidesowej, (2) obcinanie pozycji (`np.clip`) powoduje skupianie populacji w narożnikach hipersześcianu, gdzie $\|\mathbf{x}\| = \text{bound} \cdot \sqrt{d}$, oraz (3) jednoosiowe przeszukiwanie (HeavyInfantry) wymaga d iteracji na pełny cykl osi. W module `movement_strategies.py` wprowadzono sześć mechanizmów korygujących te problemy.

Refleksyjne ograniczenia dziedziny (dim > 10)

Dla wymiarów $d \leq 10$ zachowano standardowe obcinanie (`np.clip`), które zapewnia dobrą zbieżność na problemach niskiego wymiaru. Dla $d > 10$ zastąpiono je *refleksją modularną* – pozycje przekraczające granicę dziedziny "odbijają się" z powrotem:

$$x_i^{\text{reflected}} = \begin{cases} \text{mod}(x_i - lb_i,\, 2R_i) & \text{jeśli mod} \leq R_i \\ 2R_i - \text{mod}(x_i - lb_i,\, 2R_i) & \text{w p.p.} \end{cases} + lb_i \tag{5.1}$$

gdzie $R_i = ub_i - lb_i$ jest zakresem i-tej współrzędnej. Refleksja eliminuje skupianie populacji w narożnikach hipersześcianu, które powodowało katastrofalne wartości fitness – np. na XinSheYang01 ($f(\mathbf{x}) \sim \sum |x_i|^i$) wynik spadł z $3{,}6 \times 10^{42}$ do $\sim 10^{25}$ w wymiarze 100D.

Metoda ta zapobiega klasteryzacji w narożnikach przestrzeni (corner clustering), gdzie norma wektora pozycji po obcinaniu rośnie jak bound $\times\sqrt{d}$.

Normalizacja szumu względem wymiaru

Wprowadzono współczynnik skalowania wymiaru:

$$\text{dim\_scale}(d) = \begin{cases} 1{,}0 & \text{jeśli } d \leq 10 \\ \sqrt{d/10} & \text{jeśli } d > 10 \end{cases} \tag{5.2}$$

Wszystkie wektory szumu (gaussowskie i Cauchy'ego) dzielone są przez $\text{dim\_scale}(d)$, co utrzymuje ich oczekiwaną normę na stałym poziomie niezależnie od wymiaru. Skalowanie zastosowano w pięciu lokalizacjach: HeavyInfantry (szum awaryjny, perturbacja anty-stagnacyjna), Chariots (perturbacja Cauchy'ego), WarElephants (krok lokalny, szarża).

Ograniczanie normy i komponentów szumu

Wektory szumu podlegają dwupoziomowemu ograniczaniu:

1. **Ograniczanie normy euklidesowej** (`_cap_noise_norm`): zapobiega ekstremalnym skokom z rozkładu Cauchy'ego. Progi zależą od typu jednostki:
   - HeavyInfantry: $0{,}15 \cdot R$ (krok lokalny), $0{,}3 \cdot R$ (perturbacja anty-stag.)
   - Chariots: $0{,}3 \cdot R$ (perturbacja Cauchy'ego)
   - WarElephants: $0{,}15 \cdot R$ (krok lokalny), $0{,}5 \cdot R$ (szarża)

   gdzie R oznacza zakres przeszukiwania (*search_range*).

2. **Ograniczanie per-komponent** (`_cap_per_component`): obcina poszczególne składowe wektora szumu niezależnie do ±max_abs, zapobiegając jednowymiarowym skokom Cauchy'ego. Progi:
   - HeavyInfantry: $0{,}3 \cdot R/d$ (krok), $0{,}5 \cdot R/d$ (perturbacja)
   - Chariots: $0{,}5 \cdot R/d$
   - WarElephants: $0{,}3 \cdot R/d$ (krok), $0{,}8 \cdot R/d$ (szarża)

Algorytm ograniczania szumu na dwóch poziomach:

Ograniczanie szumu stosowane jest na dwóch poziomach:

```
1. Norma euklidesowa:
   noise <- noise * max_norm / max(||noise||, max_norm)

2. Per-komponentowe:
   noise_i <- clip(noise_i, -max_abs, max_abs)
```

Drugie ograniczenie zapobiega ekstremalnym skokom w pojedynczym wymiarze generowanym przez rozkład Cauchy'ego, co jest krytyczne dla funkcji typu $f(\mathbf{x}) \sim \sum |x_i|^i$ (np. XinSheYang01).

5.1.6 Złożoność obliczeniowa pętli głównej

Tabela 10 zestawia złożoność obliczeniową poszczególnych komponentów pętli optymalizacyjnej CommanderABO w przeliczeniu na pojedynczą iterację. Dominującym kosztem na praktycznych funkcjach pozostaje ewaluacja fitness $O(N \cdot C_f)$, gdzie C_f oznacza koszt jednokrotnego wywołania funkcji celu. Wszystkie operatory taktyk i moduły wsparcia są liniowe lub kwadratowe względem N i D, co oznacza, że ABO skaluje się porównywalnie z klasycznymi algorytmami rojowymi.

Tabela 10. Złożoność obliczeniowa głównych komponentów CommanderABO w przeliczeniu na pojedynczą iterację pętli. $N = 40$ – liczba jednostek, D – wymiarowość, C_f – koszt ewaluacji funkcji celu, $|A| = 6$ – liczba taktyk, G – rozdzielczość siatki rekonesansu (5–15 dla testowanych D).

Komponent	**Częstotliwość**	**Złożoność**	**Dominanta**
Ewaluacja fitness	co iterację	$O(N \cdot C_f)$	C_f (funkcja celu)
Aktualizacja pozycji	co iterację	$O(N \cdot D)$	operacje wektorowe
Wybór taktyki (Commander)	co τ iter. (5/8/15)	$O(\|A\|)$	arg max po 6 akcjach
Aktualizacja Q (Bellman)	co τ iter. (5/8/15)	$O(\|A\|)$	ten sam co wybór
Obliczenie stanu s	co τ iter. (5/8/15)	$O(N \cdot D)$	różnorodność, momentum
Operatory taktyki	co iterację	$O(N \cdot D)$	modyfikacja par. ruchu
FormationManager	co iterację	$O(N \cdot D)$	przesunięcia wokół centrum
Reconnaissance grid	co τ_R iter.	$O(G^{\min(D,2)})$	siatka eksploracji
System honoru	co iterację	$O(N)$	aktualizacja H_i
Anti-stagnation reset	co iterację	$O(N)$	sprawdzenie licznika
Suma per iteracja	—	$O(N \cdot (C_f + D))$	C_f dominuje w praktyce
Cały przebieg	—	$O(T_{\max} \cdot N \cdot (C_f + D))$	$T_{\max} = 100$, $N = 40$

5.2 PROBLEMY TESTOWE I DANE EKSPERYMENTALNE

Algorytm optymalizacyjny należy testować na różnorodnych problemach o znanych właściwościach. Poniżej opisano funkcje testowe i dane eksperymentalne użyte do oceny algorytmów z rodziny ABO.

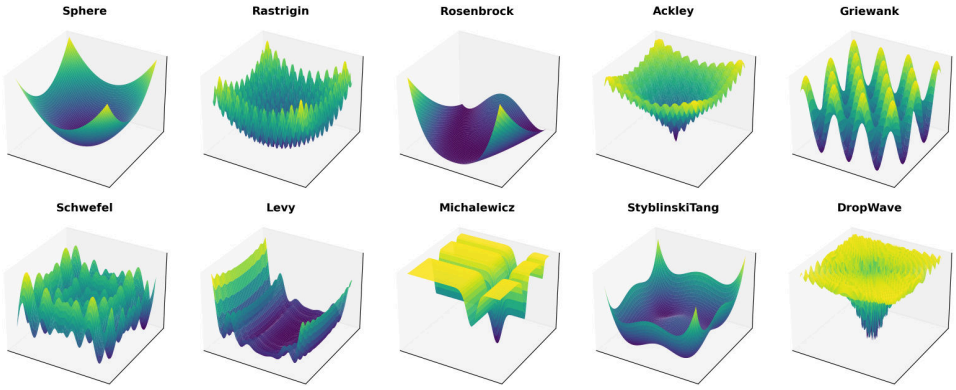
5.2.1 Funkcje testowe

Do ewaluacji algorytmu użyto funkcji benchmarkowych o zróżnicowanych właściwościach; ich wybór oparto na opracowanym indeksie trudności.

Klasyczne funkcje benchmarkowe

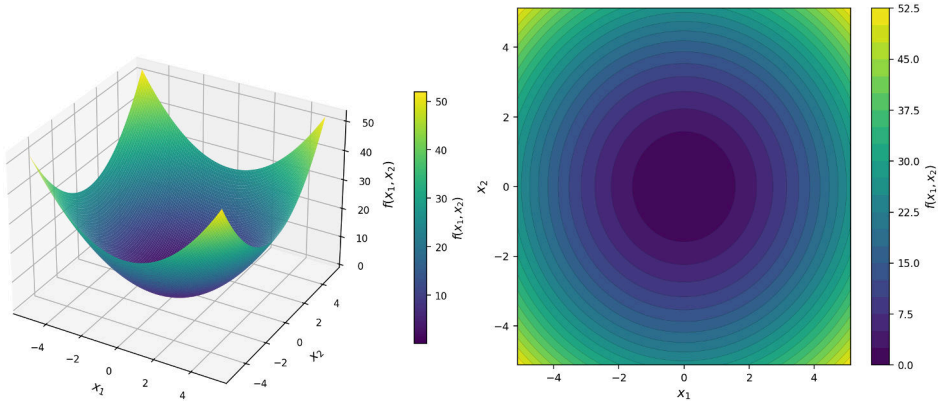
Algorytmy testowano na rozbudowanym zestawie klasycznych funkcji benchmarkowych [61] z biblioteki **opfunu** [107], obejmującym ponad 100 funkcji.

Rysunek 8 przedstawia projekcje 2D dziesięciu reprezentatywnych funkcji benchmarkowych z zestawu wykorzystanego w niniejszej pracy. Pozwala on zilustrować różnorodność topologii: od gładkiej, wypukłej misy funkcji Sphere, przez „bananową” dolinę funkcji Rosenbrocka, po silnie multimodalne pejzaże Rastrigina, Schwefela czy DropWave z setkami lokalnych minimów. Tak dobrana różnorodność charakterystyk gwarantuje, że ranking algorytmów odzwierciedla rzeczywistą zdolność uogólniania, a nie dopasowanie do jednej klasy problemów.

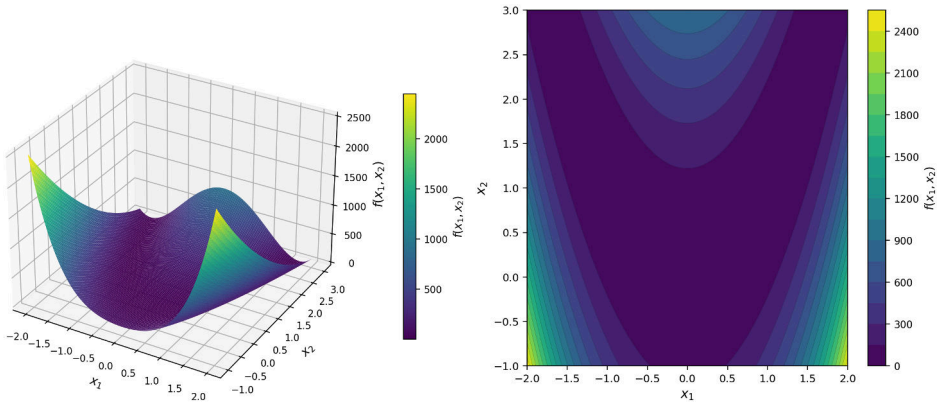


Rys. 8. Projekcje powierzchniowe 2D dziesięciu reprezentatywnych funkcji benchmarkowych z zestawu 33 funkcji użytego w ewaluacji ABO. Górny rząd: Sphere (unimodalna, separowalna), Rastrigin (silnie multimodalna), Rosenbrock (wąska dolina), Ackley (płaska zewnętrzna + głębokie centrum), Griewank (multimodalna z komponentem iloczynowym). Dolny rząd: Schwefel (optimum poza centrum), Levy (skomplikowane lokalne minima), Michalewicz (strome ściany), StyblinskiTang (symetryczne minima), DropWave (oscylacyjna). Funkcje wizualizowane na kanonicznych dziedzinach 2D – dla wymiarów wyższych ($D \in \{2, 10, 30, 50, 100\}$) topologia pozostaje analogiczna.

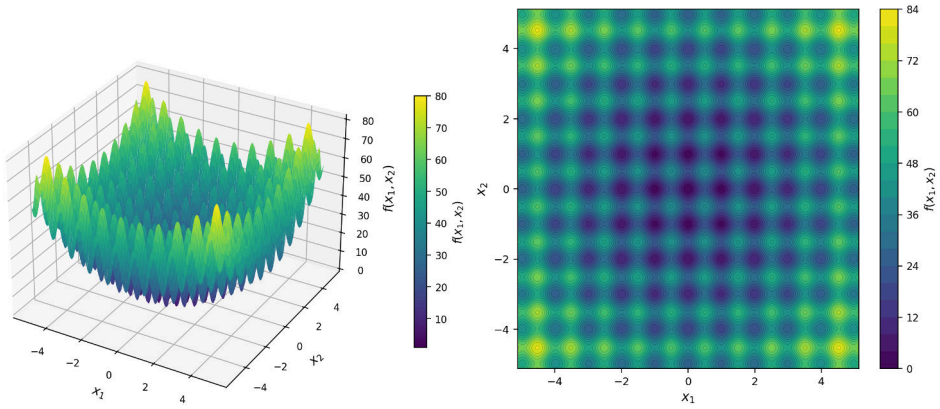
Aby wizualnie przedstawić typ problemów, dla pięciu najczęściej cytowanych funkcji benchmarkowych (Sphere, Rosenbrock, Rastrigin, Ackley, Griewank) Rysunki 9 oraz 10 przedstawiają pełną parę projekcji: powierzchnię 3D oraz mapę konturową, umożliwiającą precyzyjne odczytanie położenia minimum globalnego i struktury basenów przyciągania lokalnych minimów. Aby jednak skutecznie ocenić, które funkcje testowe rzeczywiście stanowią wyzwanie, sama wizualizacja nie wystarczy – należy przejść do analizy trudności, która jest tematem kolejnej sekcji.



(a) Sphere – powierzchnia 3D i mapa konturowa

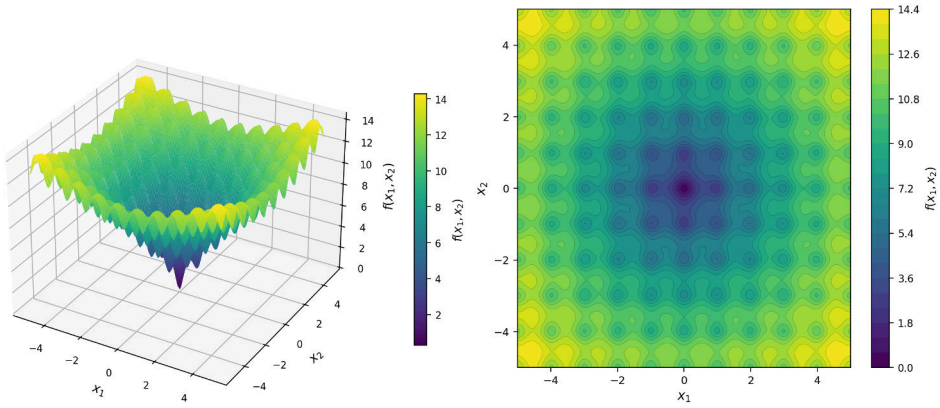


(b) Rosenbrock – wąska, zakrzywiona dolina

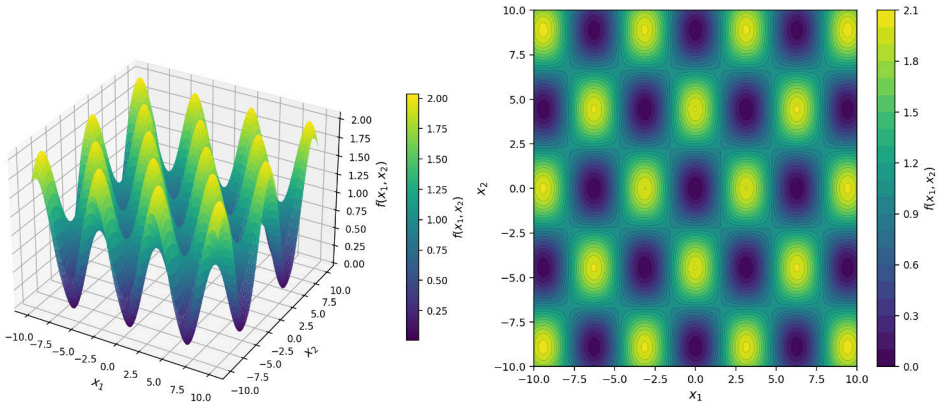


(c) Rastrigin – regularna siatka lokalnych minimów

Rys. 9. Szczegółowe wizualizacje funkcji benchmarkowych Sphere, Rosenbrock, Rastrigin w projekcji $D = 2$: powierzchnie 3D (lewa kolumna) oraz mapy konturowe (prawa kolumna).



(d) Ackley – prawie płaska zewnętrzna powierzchnia



(e) Griewank – iloczyn cosinusów + suma kwadratów

Rys. 10. Szczegółowe wizualizacje funkcji benchmarkowych Ackley i Griewank w projekcji $D = 2$: powierzchnie 3D (lewa kolumna) oraz mapy konturowe (prawa kolumna). Każda funkcja reprezentuje inną klasę trudności optymalizacyjnej.

Baza trudności:

Projekt zawiera system **Hardness Index** $H \in [0\%, 100\%]$ obliczający trudność funkcji na podstawie 7 komponentów:

- C_r – współczynnik zbieżności (ang. convergence rate, waga 25%)
- S_r – wskaźnik sukcesu (ang. success rate, waga 20%)
- V_f – znormalizowana wariancja fitness (ang. fitness variance, waga 15%)
- I_t – iteracje do progu zbieżności (ang. iterations to threshold, waga 15%)
- A_a – zgodność algorytmów (ang. algorithm agreement, waga 10%)

- L_c – złożoność krajobrazu (ang. landscape complexity, waga 10%)
- D_s – współczynnik skalowania wymiarowego (ang. dimension scaling, waga 5%)

Każdy komponent jest normalizowany do skali $[0, 1]$, gdzie 1 oznacza maksymalną trudność. Hierarchia wag odzwierciedla priorytet aspektów bezpośrednio mierzących jakość optymalizacji: zbieżność (C_r, 25%) i wskaźnik sukcesu (S_r, 20%) jako najsilniejsze predyktory trudności; wariancja fitness (V_f, 15%) i iteracje do progu (I_t, 15%) jako miary stabilności; natomiast metryki zależne od kontekstu – zgodność algorytmów (A_a, 10%), złożoność krajobrazu (L_c, 10%) i skalowanie wymiarowe (D_s, 5%) – otrzymują mniejsze wagi, gdyż ich wartość informacyjna jest bardziej pośrednia. Indeks trudności (Równanie 5.3):

$$H = 10 \cdot \sum_{i=1}^{7} w_i \cdot \tilde{m}_i \tag{5.3}$$

gdzie w_i to wagi komponentów (sumujące się do 1), a $\tilde{m}_i$ to znormalizowane metryki. Dla komponentów, w których wysoka wartość oznacza łatwość (C_r, S_r, A_a), stosowane jest odwrócenie: $\tilde{m}_i = 1 - m_i$. Wynik w skali $[0, 10]$ jest prezentowany jako procent ($H \times 10\%$). Wyższy H oznacza trudniejszą funkcję.

Proponowany Indeks Trudności (HI) różni się od analiz fitness (ELA/FLA; [79]), które charakteryzują topografię funkcji celu (np. multimodalność, długość ścieżki, dyspersja) niezależnie od algorytmów. HI jest miarą *relatywną* – trudność funkcji zależy od zestawu testowanych algorytmów, co jest zarówno ograniczeniem (zestaw-zależność), jak i zaletą (bezpośrednie odniesienie do praktycznej trudności).

Funkcje z ograniczeniami

Algorytm ABO obsługuje ograniczenia box-constraints (granice każdego wymiaru). Jeśli jednostka wielokrotnie naruszyła granice, następuje reinicjalizacja w losowej pozycji. Jest to zaimplementowane w postaci obsługi systemu honoru i dezercji.

5.2.2 Zestawy danych

Dane eksperymentalne pochodzą z kilku źródeł, zapewniających różnorodność i powtarzalność testów.

Źródła danych eksperymentalnych

1. Biblioteka opfunu (100+ funkcji):

- CEC Competition functions [75] (2005-2023)
- Klasyczne benchmarki (Sphere, Rastrigin, Ackley, itd.)

- Funkcje nazwane (Levy, Schwefel, Griewank, Michalewicz, itd.)
- Funkcje specjalne (Bent Cigar, Discus, Rosenbrock Rotated, itd.)

Uzasadnienie wyboru funkcji benchmarkowych:
W niniejszej pracy zdecydowano się na użycie klasycznych, nazwanych funkcji benchmarkowych zamiast protokołu CEC competition z kilku powodów:

1. **Weryfikowalność implementacji porównawczych algorytmów:** Do porównań wykorzystano bibliotekę mealpy, która dostarcza implementacje ponad 100 algorytmów metaheurystycznych z ostatnich 30 lat. Oryginalne publikacje tych algorytmów (GA, DE, WOA, GWO, itd.) wykorzystują w większości klasyczne, nazwane funkcje testowe (Sphere, Rastrigin, Ackley, Rosenbrock). Użycie tych samych funkcji pozwoliło na bezpośrednie porównanie wyników z literaturą i weryfikację poprawności implementacji.

2. **Reprodukowalność:** Nazwane funkcje benchmarkowe mają jednoznaczne definicje matematyczne, co gwarantuje pełną reprodukowalność eksperymentów niezależnie od wersji biblioteki czy platformy. Ziarno generatora (*seed*) jest przekazywane do każdego uruchomienia jako argument `-seed`, a następnie ustawiane wewnętrznie przed inicjalizacją populacji. Wszystkie ziarna generowane są deterministycznie z jednego ziarna głównego (`MASTER_SEED = 2026`) za pomocą funkcji `generate_seeds(master, n)`, co gwarantuje pełną odtwarzalność sekwencji uruchomień. Każde uruchomienie jest izolowane jako osobny podproces, co eliminuje przesunięcie (*drift*) generatora liczb losowych między uruchomieniami.

3. **Pokrycie przestrzeni problemów:** Wybrane 33 funkcje testowe obejmują pełne spektrum charakterystyk: funkcje unimodalne i multimodalne, separowalne i nieseparowalne, o różnej złożoności. Pozwala to na wszechstronną ocenę algorytmu bez sztucznych transformacji.

Podział na zbiory testowe i treningowe

Aby zapewnić, że proces uczenia się Commandera nie wpłynął na wynik, zastosowano rozdzielne zbiory funkcji treningowych i testowych:

1. Zestaw treningowy (10 funkcji):
Funkcje używane do trenowania Q-tabeli Commander AI:

- 10 funkcji przeznaczonych do treningu z biblioteki opfunu (zapewniających różnorodność topologii)
- Łącznie 16 000 sesji treningowych na 4 wymiarach (2D, 10D, 30D, 50D); trening wykonano w dwóch turach po 8 000 epizodów z mechanizmem

`--resume` (stąd etykieta wariantu `v7_8k` oznaczająca rozmiar pojedynczej tury). Szczegóły: Sekcja 6.9.1.

- Q-tabela akumuluje wiedzę o tym, które taktyki działają najlepiej

2. Zestaw benchmarkowy (33 funkcje):
Finalna ocena algorytmu:

- Wszystkie funkcje n-wymiarowe z biblioteki opfunu (bez implementacji własnych)
- Bez dalszego trenowania Q-tabeli
- Wyniki raportowane w pracy

Rozdzielenie zbiorów zapewnia test generalizacji – Commander AI jest oceniany na funkcjach, których nie widział podczas treningu.

Tabela 11 zestawia oba zbiory wraz z ich przeznaczeniem, liczbą funkcji, zakresem wymiarów oraz statusem tablicy Q (aktualizowana w treningu, zamrożona w ewaluacji).

Tabela 11. Podział funkcji testowych na rozłączny zbiór treningowy (do strojenia Q-tabeli Commander AI) oraz benchmarkowy (do raportowania wyników). Brak dalszego trenowania Q-tabeli podczas ewaluacji benchmarkowej zapewnia uczciwy test generalizacji.

Cecha	Zbiór treningowy	Zbiór benchmarkowy
Liczba funkcji	10	33
Cel	trening tablicy Q	ewaluacja końcowa
Wymiary	2D, 10D, 30D, 50D	2D, 10D, 30D, 50D, 100D
Liczba epizodów / uruchomień	16 000 epizodów RL (2 tury × 8 000)	100 niezależnych uruchomień
Tablica Q	aktualizowana	zamrożona
Raportowanie	nie	tak (Rozdz. 6)
Rozłączność ze zbiorem benchmarkowym	—	✓ (test generalizacji)

5.3 METODYKA EKSPERYMENTÓW

Poniżej opisano metodologię testowania i walidacji algorytmu.

5.3.1 Plan eksperymentów

Projektowanie testów

Eksperymenty zorganizowane są hierarchicznie:
1. Weryfikacja podstawowej zdolności algorytmu:

- Czy algorytm zbiega do optimum na prostych funkcjach?

- Jakie są typowe czasy wykonania?
- Jak zachowuje się przy różnych rozmiarach populacji i liczbie iteracji?

2. Analiza porównawcza Commander AI:
Ocena wkładu systemu Commander AI:

- ABO z Commander AI vs warianty fazowe ABO bez adaptacyjnego dowódcy: ile daje adaptacyjna selekcja taktyk oparta na Q-learning?

3. Badania porównawcze:
Porównanie ABO z 42 algorytmami metaheurystycznymi z różnych rodzin (łącznie 46 algorytmów: 42 z biblioteki mealpy + 4 warianty ABO). Pełny zestaw algorytmów porównawczych obejmuje:

- Algorytmy rojowe i biologiczne: ABC, ALO, BA, BeesA, BFO, BSA, BSO, CSO, DO, FA, GWO, MFO, SBO
- Algorytmy ewolucyjne i populacyjne: BBO, DE, EP, ES, FPA, GA, IWO, MA
- Algorytmy fizyczne, chemiczne i matematyczne: ACOR, ASO, CEM, EFO, EOA, GCO, GOA, GSKA, HC, HGSO, HS, SA, TWO
- Algorytmy społeczno-kulturowe i inne: BRO, CA, CHIO, CRO, CSA, FOA, ICA, JA

Wszystkie algorytmy mają tę samą liczbę epok i ten sam rozmiar populacji (pop_size $\times$ max_iter), co stanowi bazowy budżet epokowy. Warianty ABO z aktywnym lokalnym przeszukiwaniem wykonują jednak dodatkowe wywołania funkcji celu, zatem liczba ewaluacji funkcji celu (FE) nie jest ściśle równa między wszystkimi algorytmami.

4. Badania skuteczności dla problemów wielowymiarowych:
Jak algorytm radzi sobie ze wzrostem wymiarowości:

- Wymiary: $d \in \{2, 10, 30, 50, 100\}$
- Dla każdego d: 100 niezależnych przebiegów
- Analiza: czy skuteczność spada? Jak szybko? Czy ABO skaluje lepiej niż konkurencja?

Scenariusze testowe

Scenariusz: Pełen benchmark porównawczy:

- 33 funkcje benchmarkowe z biblioteki opfunu (wszystkie funkcje n-wymiarowe)
- $d \in \{2, 10, 30, 50, 100\}$
- $N = 40$, max_iter = 100
- 100 przebiegów per funkcja per wymiar
- 46 algorytmów w pełnym benchmarku (42 porównawcze + 4 warianty ABO: ABO-QTable4K, ABO-QTable10K, ABO-ManualPhases, CommanderABO)
- Łącznie 759 000 indywidualnych uruchomień w końcowym zbiorze rangowym
- Cel: pełna ocena algorytmu na zróżnicowanych problemach z analizą skalowalności

Tabela 12 podsumowuje wymiary tego planu eksperymentalnego – liczbę algorytmów, funkcji, wymiarów i niezależnych uruchomień – oraz wynikający z nich rozmiar macierzy wejściowej testu Friedmana.

Tabela 12. Macierz pełnego benchmarku porównawczego. Iloczyn wszystkich wymiarów daje całkowitą liczbę uruchomień fizycznych, natomiast macierz wejściowa testu Friedmana ma rozmiar |problemy| × |algorytmy| i zawiera mediany fitness ze 100 uruchomień.

Wymiar eksperymentu	Wartość	Uwaga
Algorytmy	46	42 porównawcze + 4 warianty ABO
Funkcje benchmarkowe	33	z biblioteki opfunu
Wymiary D	5	$\{2, 10, 30, 50, 100\}$
Niezależne uruchomienia per (algo, func, D)	100	różne ziarna deterministyczne
Konfiguracje problemów (funkcja × D)	165	wiersze macierzy Friedmana
Całkowita liczba uruchomień	759 000	$46 \times 33 \times 5 \times 100$
Rozmiar macierzy wejściowej testu Friedmana	165×46	wartości = mediany fitness
Budżet epokowy per uruchomienie	4 000	$N = 40 \times$ max_iter = 100 epok

Parametry kontrolne

Poniżej uzasadniono wybór głównych parametrów algorytmu ABO (tzw. "magic numbers"):

1. **Progi systemu honorów** (`honor_threshold` = 6,0, `dishonor_threshold` = −10,0):
   - Próg 6,0 dla statusu bohatera zapewnia, że tylko jednostki konsekwentnie osiągające lepsze wyniki (przez około 3–6 iteracji z poprawą) uzyskują awans. Wartość dobrano empirycznie, by awans nie był ani zbyt łatwy (inflacja bohaterów), ani zbyt trudny (brak mechanizmu nagradzania).
   - Asymetryczny próg degradacji (-10,0) jest bardziej surowy, ponieważ degradacja do statusu runagate (dezertera) powinna wymagać powtarzających się niepowodzeń, nie pojedynczych złych iteracji. Stosunek $|-10{,}0|/6{,}0 \approx 1{,}67$ odzwierciedla psychologiczną zasadę, że kara za porażkę powinna być proporcjonalnie silniejsza niż nagroda za sukces.
2. **Parametry Q-learningu** (`learning_rate = 0.1` początkowo, `discount_factor = 0.9`):
   - `learning_rate` (α): wartość *początkowa* (nominalna) $\alpha_0 = 0{,}1$ – standardowa w literaturze RL. Zbyt wysoka ($> 0{,}3$) powoduje niestabilność uczenia; zbyt niska ($< 0{,}01$) wymaga zbyt wielu iteracji do konwergencji. W trakcie treningu współczynnik jest adaptowany przez mechanizm `_adapt_hyperparameters()` i w zapisanej tablicy v7 osiąga wartość końcową $\approx 0{,}071$. Podczas ewaluacji benchmarkowej tablica Q jest zamrożona, więc α nie wpływa już na wynik.
   - `discount_factor` ($\gamma = 0{,}9$): Wysoka wartość preferująca długoterminowe nagrody. W optymalizacji, gdzie celem jest minimalizacja w perspektywie całego procesu, $\gamma = 0{,}9$ zapewnia, że wybór taktyki uwzględnia przyszłe korzyści (kolejne iteracje), nie tylko natychmiastową poprawę.

   Pełną listę hiperparametrów Q-learningu wraz z ich rolą i kontekstem literaturowym zestawiono w Tabeli 13. Harmonogram zaniku współczynnika eksploracji ε przedstawia Rysunek 11, który rozróżnia dwa reżimy. Podczas *treningu offline* ε maleje wykładniczo z 0,30 do progu 0,10 ($\varepsilon_t = \max(\varepsilon_{\min}, \varepsilon_0 \rho^t)$, $\rho = 0{,}99$), natomiast podczas *inferencji benchmarkowej* – gdy tablica Q jest zamrożona – ε podlega liniowemu zanikowi fazowemu z 0,12 do 0,03 w funkcji postępu optymalizacji t/T. W obu reżimach spadek ε oznacza przejście agenta z fazy eksploracji do dominującej eksploatacji nauczonej polityki ($\arg\max_a Q(s,a)$).

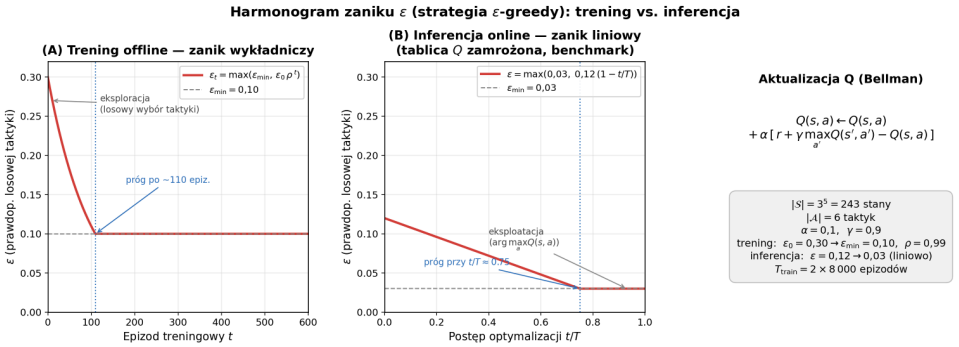
3. **Kompozycja populacji** (proporcje typów jednostek):
    - Heavy Infantry (20%, 8 jedn.) i Cavalry (20%, 8 jedn.): Główne typy eksploatacyjne i eksploracyjne, dominujące w populacji.
    - Light Infantry (20%, 8 jedn.) i Archers (17,5%, 7 jedn.): Typy wspomagające, balansujące eksplorację z eksploatacją.
    - Chariots (12,5%, 5 jedn.) i War Elephants (10%, 4 jedn.): Typy specjalistyczne dla złożonych przestrzeni.
    - Łącznie 40 jednostek. Proporcje oparte na analizie danych treningowych: optymalna kompozycja zależy od charakterystyki funkcji, ale powyższe wartości sprawdzają się jako domyślne na zróżnicowanych problemach.
4. **Rekonesans** (`grid_size`):
    - Rozdzielczość siatki adaptuje się automatycznie do wymiarowości: $G = \min(20,\ \max(5,\ \lfloor 20 \cdot D^{-1/3} \rfloor))$.
    - Dla 2D daje $G = 15$ (≤ 225 komórek), dla 10D $G = 9$, dla 30D $G = 6$, dla 50D i 100D $G = 5$ (rozdzielczość minimalna).
5. **Współczynnik formacji** (`formation_strength = 0.1`):
    - Współczynnik określający wpływ formacji na ruch jednostek. Wartość 0,1 oznacza 10% wkładu formacji w wektor ruchu, 90% to indywidualny ruch jednostki. Zbyt wysoka wartość ($> 0{,}3$) prowadzi do zbyt silnego grupowania (przedwczesna zbieżność, ang. *premature convergence*); zbyt niska ($< 0{,}05$) czyni formacje nieistotnymi.
6. **Kryteria końcowe:**
    - Maksymalna liczba iteracji: 100, podjęto decyzję o niekorzystaniu z implementacji stagnacji rozwiązań ze względu na trudności porównawcze czasu wykonania algorytmu z algorytmami uznanymi w dziedzinie, które korzystają z metody stałej maksymalnej liczby iteracji.

Deklaracja asymetrii strojenia parametrów

Powyższe wartości parametrów algorytmów ABO oraz CommanderABO zostały dobrane iteracyjnie w oparciu o serie pilotażowe na podzbiorze funkcji testowych – proces ten obejmował kilkadziesiąt rund kalibracji (składu armii, progów honorów, hiperparametrów Q-learningu, rozdzielczości siatki rekonesansu). W przeciwieństwie do tego, **wszystkie 42 algorytmy porównawcze** z biblioteki `mealpy` [108] uruchomiono z **parametrami domyślnymi** dostarczonymi przez autorów, bez dodatkowego strojenia per-funkcja ani per-wymiar.

Tabela 13. Pełna konfiguracja hiperparametrów Q-learningu modułu Commander AI. Wszystkie wartości pozostają stałe podczas ewaluacji benchmarkowej (tablica Q zamrożona).

Parametr	**Wartość**	**Rola**
Rozmiar przestrzeni stanów $\lvert\mathcal{S}\rvert$	$3^5 = 243$	5 wymiarów stanu × 3 kategorie
Wymiary stanu	5	faza, stagnacja, momentum, różnorodność, % bohaterów
Rozmiar przestrzeni akcji $\lvert\mathcal{A}\rvert$	6	6 taktyk (Phalanx, Oblique, . . .)
Rozmiar tablicy Q	$243 \times 6 = 1458$	wartości wyuczonych
Współczynnik uczenia α (początkowy)	$0{,}1 \rightarrow \approx 0{,}071$	standard RL; adaptowany w treningu
Współczynnik dyskontowy γ	0,9	nagrody długoterminowe
Początkowy ε – trening	0,30	szeroka eksploracja na starcie treningu
Minimalny $\varepsilon_{\min}$ – trening	0,10	próg po zaniku wykładniczym (`v7_8k`)
Tempo zaniku ρ – trening	0,99	$\varepsilon_t = \max(\varepsilon_{\min}, \varepsilon_0 \rho^t)$
ε – inferencja (benchmark)	$\max(0{,}03;\ 0{,}12(1-t/T))$	liniowy zanik fazowy 12% → 3% przy zamrożonej tablicy Q
Liczba epizodów treningu CommanderABO	16 000 (2 × 8 000)	QTable4K: 4 000; QTable10K: 10 000
Częstotliwość decyzji τ	adaptacyjnie: 5/8/15 iter.	rzadziej w fazie eksploatacji; stabilizacja $\Delta Q \geq 0{,}1$
Funkcja nagrody r_t	$\Delta f^* / \lvert f^*_{t-1} \rvert$	relatywna poprawa + bonus antystagnacyjny



Rys. 11. Harmonogram współczynnika eksploracji ε w strategii ε-greedy w dwóch reżimach. **(A) Trening offline:** zanik wykładniczy $\varepsilon_t = \max(\varepsilon_{\min}, \varepsilon_0 \rho^t)$ z $\varepsilon_0 = 0{,}30$ i $\rho = 0{,}99$ osiąga próg $\varepsilon_{\min} = 0{,}10$ po ok. 110 epizodach i pozostaje na nim do końca treningu (pełny trening CommanderABO to dwie tury, łącznie 16 000 epizodów). **(B) Inferencja online** podczas benchmarku (tablica Q zamrożona): zanik liniowy $\varepsilon = \max(0{,}03;\ 0{,}12\,(1 - t/T))$ w funkcji postępu optymalizacji t/T, osiągający próg 0,03 przy $t/T \approx 0{,}75$. W obu reżimach niskie ε odpowiada przejściu z fazy eksploracji (losowy wybór taktyki) do dominującej eksploatacji nauczonej polityki ($\arg\max_a Q(s,a)$). Po prawej: wzór aktualizacji Bellmana oraz konfiguracja Q-learningu.

Decyzja ta jest świadomym wyborem metodycznym o następującym uzasadnieniu: (i) strojenie 42 algorytmów na 33 funkcjach × 5 wymiarach × k konfiguracji parametrów wykraczałoby poza realne ramy obliczeniowe pojedynczej rozprawy doktorskiej; (ii) parametry domyślne `mealpy` są wartościami rekomendowanymi przez autorów oryginalnych publikacji algorytmów, co odpowiada warunkom ich typowego użycia przez praktyków; (iii) asymetria ta odpowiada powszechnej praktyce publikacji w dziedzinie metaheurystyk, gdzie nowe algorytmy są strojone intensywniej niż metody bazowe.

Konsekwencje dla interpretacji wyników: przewaga ABO/CommanderABO w rankingach Friedmana (Rozdział 6) odzwierciedla łączny efekt projektu algorytmu *oraz* jego strojenia względem zestawu benchmarkowego. Przy osobnym strojeniu algorytmów konkurencyjnych rankingi mogłyby ulec zmianie – ograniczenie to omówiono ponownie w dyskusji ograniczeń (Sekcja 7.3).

5.3.2 Metryki oceny

Miary dokładności

1. Najlepsza wartość fitness:
Najprostsza metryka – jak blisko optimum algorytm dotarł:

$$\text{Best} = \min_{i=1}^{R} f(\mathbf{x}_{\text{best}}^{(i)}) \tag{5.4}$$

gdzie R to liczba niezależnych przebiegów.

2. Uśredniona najlepsza wartość fitness:
Średnia z najlepszych wartości fitness ze wszystkich przebiegów:

$$\text{Mean Best} = \frac{1}{R} \sum_{i=1}^{R} f(\mathbf{x}_{\text{best}}^{(i)}) \tag{5.5}$$

3. Błąd względem wartości optymalnej:
Dla funkcji z znanym optimum f^*:

$$\text{Error} = |f(\mathbf{x}_{\text{best}}) - f^*| \tag{5.6}$$

Błąd logarytmiczny dla skali logarytmicznej:

$$\text{Log-Error} = \log_{10}(\text{Error} + 10^{-16}) \tag{5.7}$$

4. Procent przebiegów zakończonych sukcesem:
Odsetek przebiegów, które osiągnęły zadowalający wynik:

$$SR = \frac{\text{liczba przebiegów z } |f(\mathbf{x}_{\text{best}}) - f^*| < \epsilon}{R} \tag{5.8}$$

gdzie ϵ to próg (typowo 10^{-8} lub 10^{-2} w zależności od funkcji).

Wskaźniki wydajności

1. Prędkość konwergencji:
Jak szybko algorytm poprawia rozwiązanie:

$$CR = \frac{f(\mathbf{x}_{\text{init}}) - f(\mathbf{x}_{\text{final}})}{T_{\max}} \tag{5.9}$$

wyższa wartość = szybsza konwergencja.

Kryteria porównawcze

Do porównania ABO z innymi algorytmami stosowane są:

1. Test Wilcoxona (signed-rank): nieparametryczny test sprawdzający, czy dwa algorytmy mają różne mediany (`scipy.stats.wilcoxon`); różnica uznawana jest za istotną przy $p < 0{,}05$.

2. Test Friedmana + post-hoc: dla porównania wielu algorytmów na wielu funkcjach (`scipy.stats.friedmanchisquare`); przy istotnym wyniku ($p < 0{,}05$) stosowany jest test post-hoc Nemenyiego (`scikit_posthocs`).

Agregacja danych wejściowych: Dla każdej kombinacji (algorytm, funkcja, wymiar) obliczono **medianę** fitness ze 100 niezależnych uruchomień. Pełny benchmark obejmuje 33 funkcje × 5 wymiarów = 165 konfiguracji problemów oraz 46 algorytmów. Macierz wejściowa testu Friedmana ma wymiar 165×46. Każda komórka macierzy zawiera medianę `Best_Fitness` ze 100 uruchomień dla danej pary (algorytm, konfiguracja). Rankingi są wyznaczane per konfiguracja, a następnie uśredniane po wszystkich konfiguracjach, dając średni ranking Friedmana dla każdego z 46 algorytmów uwzględnionych w analizie.

3. Wielkość efektu (Cohen's d) [23]:
Wielkość efektu między dwoma algorytmami:

$$d = \frac{\mu_{\text{ABO}} - \mu_{\text{GWO}}}{\sqrt{\frac{\sigma^2_{\text{ABO}} + \sigma^2_{\text{GWO}}}{2}}} \tag{5.10}$$

Interpretacja:

- $|d| < 0{,}2$: efekt pomijalny
- $0{,}2 \leq |d| < 0{,}5$: mały efekt
- $0{,}5 \leq |d| < 0{,}8$: średni efekt
- $|d| \geq 0{,}8$: duży efekt

Tabela 14 zestawia komplet metryk dokładności i wydajności opisanych powyżej wraz z ich definicjami i interpretacją, natomiast Tabela 15 – wszystkie testy statystyczne wykorzystywane w pracy wraz z ich celem, założeniami i rolą w łańcuchu analitycznym (od pojedynczej pary algorytmów po ranking globalny).

Tabela 14. Zestawienie metryk stosowanych do oceny algorytmów. R oznacza liczbę niezależnych uruchomień, f^* znaną wartość optymalną funkcji, ϵ próg sukcesu (zwykle 10^{-8} lub 10^{-2}). Metryki dokładności (1–4) opisują jakość rozwiązania, metryka wydajności (5) opisuje tempo dochodzenia do tego rozwiązania.

Skrót	Nazwa	Definicja	Co mierzy
Best	najlepsze fitness	$\min_{i=1}^{R} f(\mathbf{x}_{\text{best}}^{(i)})$	rekord algorytmu
Mean Best	średnie najlepsze fitness	$\frac{1}{R}\sum_i f(\mathbf{x}_{\text{best}}^{(i)})$	typowa jakość
Error	błąd względem optimum	$\lvert f(\mathbf{x}_{\text{best}}) - f^*\rvert$	odległość od minimum globalnego
Log-Error	błąd logarytmiczny	$\log_{10}(\text{Error} + 10^{-16})$	skala log dla małych błędów
SR	success rate	$\#\{i : \lvert f - f^*\rvert < \epsilon\}/R$	odporność (odsetek sukcesów)
CR	convergence rate	$(f(\mathbf{x}_{\text{init}}) - f(\mathbf{x}_{\text{final}}))/T_{\max}$	tempo poprawy w czasie

Tabela 15. Zestawienie testów statystycznych stosowanych w pracy oraz ich roli w łańcuchu analitycznym (od pojedynczej pary algorytmów do rankingu globalnego). Korekty dla wielokrotnych porównań omówiono w osobnym suplemencie metodologicznym.

Test	Cel	Założenia	Zastosowanie
Wilcoxon signed-rank	dwa algorytmy, sparowane wyniki	nieparametryczny	porównanie ABO vs konkretny konkurent na 100 uruchomieniach
Friedman	$k > 2$ algorytmów na N problemach	rangi	globalny ranking 46 algorytmów na 165 konfiguracjach
Nemenyi post-hoc	wszystkie pary po Friedmanie	rangi, CD	wyznaczenie grupy nierozróżnialnych liderów
Cohen's d	wielkość efektu (parametryczna)	rozkład ~ normalny	magnituda różnicy ABO vs konkurent
Cliff's δ	wielkość efektu (nieparametryczna)	rangi	magnituda dla rozkładów skośnych
Bootstrap (1 000 rep.)	przedziały ufności średnich rang	resampling z powtórzeniami	95% CI dla rang liderów
Holm-Bonferroni	korekta wielokrotnych porównań	FWER	redukcja błędu I rodzaju w teście Wilcoxona
Benjamini-Hochberg	korekta wielokrotnych porównań	FDR	alternatywa dla Holma z większą mocą

5.3.3 Analiza statystyczna

Metody statystyczne

Statystyki opisowe:
Dla każdej funkcji i każdego algorytmu raportowano:

- Mean (średnia)
- Median (mediana)
- Std (odchylenie standardowe)
- Min/Max

Testy istotności

Zastosowano testy nieparametryczne, zgodnie z rekomendacjami literaturowymi dla algorytmów stochastycznych:

- **Test Wilcoxona ze znakami rang** [118]: porównanie sparowane dwóch algorytmów
- **Test Friedmana** [42]: porównanie wielu algorytmów na wielu funkcjach z generowaniem rankingów średnich
- **Test post-hoc Nemenyi** [86]: identyfikacja par algorytmów istotnie różniących się

Korekta dla porównań wielokrotnych:
Przy wielu porównaniach (np. ABO vs. 42 innych algorytmów), ryzyko wyników fałszywie pozytywnych (błąd typu I) rośnie znacząco. Stosowane metody korekty:

- **Bonferroni** [19]: $\alpha_{\text{skorygowana}} = \alpha/m$ gdzie m to liczba porównań. Najostrzejsza korekta, kontroluje rodzinowy współczynnik błędu (FWER). Często zbyt konserwatywna.
- **Holm-Bonferroni (procedura zstępująca)** [58]: Sekwencyjna procedura mniej konserwatywna od Bonferroni, ale wciąż kontrolująca FWER. Procedura:
  1. Uporządkuj p-wartości rosnąco: $p_{(1)} \leq p_{(2)} \leq \ldots \leq p_{(m)}$
  2. Dla każdego i, porównaj $p_{(i)}$ z $\alpha/(m-i+1)$
  3. Odrzucaj hipotezy od najmniejszej p-wartości, aż do pierwszej niespełniającej warunku

  Skorygowana p-wartość: $\tilde{p}_{(i)} = \min\left(1, p_{(i)} \cdot (m-i+1)\right)$
- **Benjamini-Hochberg (FDR)** [14]: Kontroluje odsetek fałszywych odkryć (FDR – stosunek fałszywych odkryć do wszystkich odrzuconych hipotez), nie FWER. Mniej konserwatywna, odpowiednia gdy akceptowalny jest pewien odsetek błędów typu I.

W benchmarkach algorytmów optymalizacyjnych zastosowano korektę **Holma-Bonferroniego** dla porównań parami jako kompromis między kontrolą błędu I rodzaju a mocą statystyczną [58, 14].

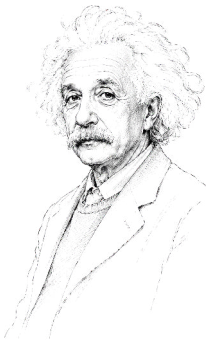
5.3.4 Podsumowanie testów statystycznych

Zastosowane testy nieparametryczne (Friedman, Nemenyi, Wilcoxon) wraz z korektami wielokrotnych porównań (Holm-Bonferroni, Benjamini-Hochberg) i wielkością efektu Cohena opisano w Sekcji 5.3.3. Wybór testów nieparametrycznych uzasadniony jest brakiem normalności rozkładów wyników algorytmów stochastycznych. Szczegółowy opis każdego testu – geneza, założenia, wzory, sposoby interpretacji i przykłady – zawarty jest w osobnym suplemencie metodologicznym dołączonym do rozprawy.

Przedstawiona realizacja algorytmu, zestaw funkcji testowych oraz metodologia statystyczna tworzą podstawę do oceny eksperymentalnej. Kolejny rozdział sprawdza, czy metafora strategii pola bitwy przekłada się na skuteczność optymalizacyjną oraz jak CommanderABO wypada w porównaniu z 42 metodami metaheurystycznymi.

6. WYNIKI EKSPERYMENTALNE I DYSKUSJA

„Gdybyśmy wiedzieli, co robimy, nie nazywałoby się to badaniami."
„If we knew what we were doing, it would not be called research." (ang.)
– Albert Einstein



Ten rozdział prezentuje wyniki eksperymentów obliczeniowych z Rozdziału 5: metodykę i konfigurację Commander AI (6.1), indeks trudności (6.2), wyniki porównawcze (6.3.2), analizę statystyczną (6.4), analizę porównawczą CommanderABO z wariantami fazowymi ABO (6.5), zbieżność (6.6), wydajność obliczeniową (6.7.4) oraz podsumowanie (6.9).

6.1 METODYKA BADAŃ

Algorytm Commander Ancient Battlefield Optimizer (CommanderABO) porównano z **42 algorytmami metaheurystycznymi** z różnych rodzin (łącznie **46 algorytmów**: 42 z biblioteki mealpy [108] + 4 warianty ABO: ABO-QTable4K, ABO-QTable10K, ABO-ManualPhases, CommanderABO). Parametry eksperymentów:

- **Liczba wymiarów**: 2D, 10D, 30D, 50D, 100D
- **Funkcje testowe**: 33 funkcje benchmarkowe z biblioteki opfunu [107]
- **Liczba uruchomień**: 100 dla każdej konfiguracji (zapewniająca istotność statystyczną)
- **Liczba epok**: 100
- **Rozmiar populacji**: 40 (patrz Rozdział 4, Tabela 4)
- **Łączna liczba uruchomień**: 46 algorytmów × 165 konfiguracji testowych × 100 uruchomień = 759 000 indywidualnych uruchomień

Wybór 100 uruchomień dla każdej konfiguracji wynika z losowej natury algorytmów metaheurystycznych: każde uruchomienie tego samego algorytmu na tym

samym problemie daje inny wynik. Pojedyncze uruchomienie nie pozwala ocenić, czy algorytm jest rzeczywiście dobry – mógł mieć po prostu „szczęście". Dopiero wielokrotne powtórzenie pozwala obliczyć stabilne statystyki (medianę, rozstęp międzykwartylowy) i zastosować testy statystyczne (Sekcja 6.4) do rzetelnego porównania algorytmów.

Reprezentatywność zbioru porównawczego

Zbiór 42 algorytmów konkurencyjnych celowo obejmuje szeroki przekrój historyczny i paradygmatyczny metaheurystyk – od wielokrotnie cytowanych klasyk (GA, SA, DE) z tysiącami niezależnych walidacji, po najnowsze propozycje z ostatnich lat, dla których publicznie dostępne były dane porównawcze umożliwiające weryfikację poprawności implementacji (HGSO, GSKA, BRO, FOA) – aby uniknąć zarzutu doboru wyłącznie słabych lub wyłącznie nowoczesnych algorytmów odniesienia. Tabela 16 zawiera pełną listę z datami publikacji oryginalnych prac.

Tabela 16. Pełna lista 46 algorytmów benchmarku, posortowana chronologicznie według roku publikacji oryginalnej pracy.

#	Skrót	Pełna nazwa	Rok	Autor(zy)
1	HC	Hill Climbing	lata 60. XX w.	Newell & Simon (klasyka AI)
2	ES	Evolution Strategy	1965/1973	Rechenberg, Schwefel
3	EP	Evolutionary Programming	1966	Fogel, Owens & Walsh
4	GA	Genetic Algorithm	1975	Holland
5	SA	Simulated Annealing	1983	Kirkpatrick et al.
6	MA	Memetic Algorithm	1989	Moscato
7	CA	Cultural Algorithm	1994	Reynolds
8	DE	Differential Evolution	1997	Storn & Price
9	CEM	Cross-Entropy Method	1997	Rubinstein
10	HS	Harmony Search	2001	Geem et al.
11	BFO	Bacterial Foraging Optimization	2002	Passino
12	ABC	Artificial Bee Colony	2005	Karaboga
13	BeesA	Bees Algorithm	2005	Pham et al.
14	CSO	Cat Swarm Optimization	2006	Chu, Tsai & Pan
15	IWO	Invasive Weed Optimization	2006	Mehrabian & Lucas
16	ICA	Imperialist Competitive Algorithm	2007	Atashpaz-Gargari & Lucas
17	ACOR	Ant Colony Optimization (continuous)	2008	Socha & Dorigo
18	BBO	Biogeography-Based Optimization	2008	Simon
19	BA	Bat Algorithm	2010	Yang
20	FA	Fireworks Algorithm	2010	Tan & Zhu

Tabela 16 – kontynuacja

#	Skrót	Pełna nazwa	Rok	Autor(zy)
21	BSO	Brain Storm Optimization	2011	Shi
22	FOA	Fruit-fly Optimization Algorithm	2012	Pan
23	FPA	Flower Pollination Algorithm	2012	Yang
24	CRO	Coral Reefs Optimization	2014	Salcedo-Sanz et al.
25	GWO	Grey Wolf Optimizer	2014	Mirjalili et al.
26	ALO	Ant Lion Optimizer	2015	Mirjalili
27	MFO	Moth-Flame Optimization	2015	Mirjalili
28	BSA	Bird Swarm Algorithm	2016	Meng et al.
29	CSA	Crow Search Algorithm	2016	Askarzadeh
30	DO	Dragonfly Optimization	2016	Mirjalili
31	EFO	Electromagnetic Field Optimization	2016	Abedinpourshotorban et al.
32	JA	Jaya	2016	Rao
33	TWO	Tug of War Optimization	2016	Kaveh & Zolghadr
34	GOA	Grasshopper Optimization Algorithm	2017	Saremi et al.
35	SBO	Satin Bowerbird Optimizer	2017	Moosavi & Bardsiri
36	EOA	Earthworm Optimisation Algorithm	2018	Wang et al.
37	GCO	Germinal Center Optimization	2018	Villaseñor et al.
38	ASO	Atom Search Optimization	2019	Zhao, Wang & Zhang
39	HGSO	Henry Gas Solubility Optimization	2019	Hashim et al.
40	BRO	Battle Royale Optimization	2020	Rahkar Farshi
41	CHIO	Coronavirus Herd Immunity Optimization	2020	Al-Betar et al.
42	GSKA	Gaining-Sharing Knowledge Algorithm	2020	Mohamed et al.
43	**ABO-ManualPhases**	**ABO z ręcznie zaprojektowanymi fazami**	**2025**	
44	**ABO-QTable4K**	**ABO ze statycznym harmonog. (4K epiz.)**	**2025**	
45	**ABO-QTable10K**	**ABO ze statycznym harmonog. (10K epiz.)**	**2025**	
46	**CommanderABO**	**Commander Ancient Battlefield Optimizer**	**2025**	

Zbiór obejmuje zarówno wieloletnie klasyki (GA z 1975 r., SA z 1983 r., DE z 1997 r.), jak i algorytmy ostatnich lat (HGSO, GSKA, CHIO, BRO, FOA), co ogranicza ryzyko doboru wyłącznie słabych punktów odniesienia.

Szczegółową charakterystykę 42 algorytmów porównawczych przedstawiono w Sekcji 6.3.

Taktyki CommanderABO (sześć taktyk bitewnych) zostały szczegółowo opisane w Sekcji 4.2.1 i Tabeli 6.

6.1.1 Warianty algorytmu ABO

Aby ocenić wkład poszczególnych komponentów w wydajność strategii, w benchmarku uwzględniono cztery warianty ABO, różniące się wyłącznie **mechanizmem selekcji taktyk**. Wszystkie warianty współdzielą identyczny skład armii (40 jednostek) oraz parametry optymalizacji, co zapewnia uczciwą ablację izolującą wpływ sposobu wyboru taktyk.

Kluczowa różnica architekturalna polega na tym, że CommanderABO podejmuje decyzje **online** (w adaptacyjnym interwale co 5/8/15 iteracji, na podstawie bieżącego stanu optymalizacji), podczas gdy warianty QTable4K/10K i ManualPhases stosują **z góry ustalone harmonogramy**.

Harmonogram zastosowany w wariancie ABO-ManualPhases nie jest arbitralny. Zarówno kolejność sześciu taktyk, jak i progi przejść między fazami (10%, 25%, 45%, 60%, 80%) ustalono empirycznie w serii eksperymentów wstępnych: przetestowano szereg alternatywnych przyporządkowań taktyk do faz oraz wariantów granic czasowych, a następnie wybrano kombinację osiągającą najlepszą wydajność na zestawie funkcji benchmarkowych. Wariant ABO-ManualPhases reprezentuje zatem *najlepszą odkrytą statyczną konfigurację* algorytmu ABO – maksimum osiągalne dla ręcznie zaprojektowanego, niezależnego od zadania harmonogramu faz. Ma to istotne konsekwencje interpretacyjne: stanowi on wymagającą, dostrojoną linię odniesienia (a nie konfigurację przypadkową), wobec której mierzona jest wartość dodana adaptacyjnej selekcji taktyk przez Q-learning. Tym samym przewaga ABO-ManualPhases w kryterium mediany (Sekcja 6.5.1) jest tym bardziej znacząca, że odniesieniem nie jest dowolny, lecz najlepszy znaleziony harmonogram statyczny.

Podsumowując: cztery warianty ABO różniące się wyłącznie mechanizmem selekcji taktyk pozwalają na systematyczną ocenę wkładu Commander AI – od statycznych harmonogramów (ManualPhases, QTable4K/10K) po pełną adaptację online (CommanderABO).

Tabela 17. Porównanie wariantów algorytmu ABO

Wariant	Selekcja taktyk	Opis
CommanderABO	adaptacyjny Q-learning	Commander AI obserwuje 5 cech stanu (3^5 = 243 stanów) i w adaptacyjnym interwale (co 5/8/15 iteracji) wybiera jedną z 6 taktyk na podstawie wyuczonej tablicy Q (v7, 16 000 epizodów treningowych = 2 tury po 8 000). Decyzje są podejmowane online – algorytm adaptuje taktykę do bieżącej fazy optymalizacji.
ABO-ManualPhases	stały harmonogram	Sześć taktyk przypisanych do kolejnych faz optymalizacji wg ustalonego schematu: Scouting (0–10%), Flanking (10–25%), Phalanx (25–45%), Oblique (45–60%), Center Penetration (60–80%), Surrounding (80–100%). Przejścia między fazami następują deterministycznie na podstawie postępu iteracji.
ABO-QTable4K	statyczne fazy z Q-tablicy	Harmonogram faz wygenerowany offline z tablicy Q wytrenowanej przez 4 000 epizodów (v5). W czasie optymalizacji fazy się nie adaptują – sekwencja taktyk jest ustalona przed uruchomieniem.
ABO-QTable10K	statyczne fazy z Q-tablicy	Analogicznie do QTable4K, lecz tablica Q wytrenowana przez 10 000 epizodów. Dłuższy trening nie przełożył się na lepsze wyniki, co sugeruje nasycenie uczenia dla statycznych harmonogramów.

6.1.2 Konfiguracja Commander AI i Q-learningu

System Commander AI wykorzystuje **rozdzielne zbiory funkcji** dla treningu i ewaluacji tablicy Q-learning [114, 104]; metodykę i uzasadnienie tego podziału przedstawiono w Sekcji 5.2.2 (Tabela 11). Poniżej wyszczególniono konkretne funkcje użyte w obu zbiorach.

Funkcje treningowe (10 funkcji):

Trening tablicy Q-learning przeprowadzono na następujących funkcjach, które **nie występują w benchmarku ewaluacyjnym**:

- **Perm0db** – funkcja testująca zdolność znajdowania rozwiązań permutacyjnych, granice $(-d, d)$
- **Trid** – funkcja z wieloma lokalnymi minimami, granice $(-100, 100)$

- **Powell** – klasyczna funkcja testowa, granice $(-4, 5)$
- **SumDiffPowers** – funkcja testująca wrażliwość na wymiar, granice $(-1, 1)$
- **RotatedHyperEllipsoid** – gładka funkcja unimodalna, granice $(-65{,}536, 65{,}536)$
- **Schumer** – funkcja testująca obsługę różnych wykładników, granice $(-100, 100)$
- **Exponential** – gładka funkcja z pojedynczym minimum, granice $(-1, 1)$
- **Schwefel220** – suma wartości bezwzględnych, granice $(-100, 100)$
- **Schwefel221** – maksimum wartości bezwzględnych, granice $(-100, 100)$
- **Schwefel222** – suma i iloczyn wartości bezwzględnych, granice $(-100, 100)$

Funkcje benchmarkowe:

Ewaluacja CommanderABO przeprowadzana jest na zestawie 33 funkcji z biblioteki opfunu (zupełnie oddzielnym od zbioru treningowego):

AMGM, Ackley01, Alpine01, Alpine02, Brown, ChungReynolds, Cigar, CosineMixture, Csendes, Deb01, Deceptive, DeflectedCorrugatedSpring, DixonPrice, DropWave, EggHolder, Griewank, Infinity, Levy03, Mishra01, Mishra02, Mishra11, MultiModal, NeedleEye, Parsopoulos, Qing, Quartic, Quintic, Rana, Salomon, XinSheYang01, YaoLiu04, Zacharov, ZeroSum.

Uzasadnienie metodyczne: rozdzielenie zbiorów zapewnia uczciwy test generalizacji i wyklucza przeuczenie – szczegółowe uzasadnienie tej praktyki podano w Sekcji 5.2.2. Funkcje treningowe dobrano tak, by reprezentowały różne typy przestrzeni (unimodalne, multimodalne, różne skale i granice).

Parametry treningu:

Trening tablicy Q-learning przeprowadzono w dwóch turach po 8 000 epizodów (mechanizm `--resume`, dziedziczone Q-wartości, statystyki wizyt oraz harmonogram ϵ-decay), łącznie **16 000 epizodów** na **10 funkcjach treningowych** w 4 wymiarach. Etykieta wariantu (`v7_8k`) odzwierciedla rozmiar pojedynczej tury; faktyczna liczba zarejestrowanych przejść stan–akcja wynosi 16 000.

Wyniki treningu:

Trening przeprowadzono sekwencyjnie, rotując między funkcjami treningowymi i wymiarami. Zapewniło to stabilność procesu uczenia i szerokie pokrycie przestrzeni stanów. System Commander AI odwiedził 187 z 243 możliwych stanów optymalizacyjnych (5 cech $\times$ 3 kategorie) i nauczył się dobierać odpowiednie taktyki na podstawie zgromadzonych doświadczeń.

Tabela 18. Parametry treningu tablicy Q-learning

Parametr	**Wartość**
Łączna liczba epizodów treningowych	16 000 (2 tury × 8 000)
Liczba funkcji treningowych	10
Epoki na epizod	100
Rozmiar populacji	40
Wymiary treningowe	2D, 10D, 30D, 50D
Współczynnik uczenia (α)	0,1
Współczynnik dyskontowy (γ)	0,9
Początkowa eksploracja (ϵ_0)	0,3
Minimalna eksploracja (ϵ_{min})	0,1
Zanik eksploracji (ϵ_{decay})	0,99

Tabela 19. Statystyki wytrenowanej tablicy Q-learning (v7)

Metryka	**Wartość**
Łączna liczba epizodów treningowych	16 000
Liczba stanów w tablicy Q ($\|S\| = 3^5$)	243
Liczba odwiedzonych stanów	187
Liczba akcji ($\|A\|$ – 6 taktyk)	6
Pokrycie stanów	77,0% (187/243)
Końcowy współczynnik eksploracji	0,1

Analiza wrażliwości projektu przestrzeni stanów

Przestrzeń stanów Commander AI wykorzystuje 5-wymiarową reprezentację (faza optymalizacji, stagnacja, momentum, różnorodność, odsetek bohaterów) z 3 kategoriami na wymiar, co daje łącznie $3^5 = 243$ możliwych stanów. Projekt i uzasadnienie wyboru tych pięciu wymiarów opisano w Sekcji 4.2.5, a pełną dyskretyzację każdego wymiaru (definicje cech wraz z progami kategorii) podaje Algorytm 8 w Dodatku A. W tym miejscu przedstawiono natomiast **analizę wrażliwości** tego projektu na progi i rozdzielczość dyskretyzacji, weryfikującą empirycznie jego solidność.

Analiza wrażliwości na progi dyskretyzacji:

Tabela 20 przedstawia analizę wrażliwości na progi dyskretyzacji fazy optymalizacji:

Wyniki pokazują, że algorytm jest odporny na zmiany progów dyskretyzacji. Różnice względem konfiguracji domyślnej nie przekraczają 0,22 pozycji, co wskazuje na stabilność projektu.

Analiza wrażliwości na rozdzielczość dyskretyzacji:

Tabela 21 porównuje wydajność dla różnych rozdzielczości dyskretyzacji:

Wnioski:

Tabela 20. Analiza wrażliwości na progi dyskretyzacji fazy (zmiana średniej rangi względem konfiguracji domyślnej)

Konfiguracja progów	Zmiana rangi	Interpretacja
Early/Mid: 0,3, Mid/Late: 0,7 (domyślne)	0,00	punkt odniesienia
Early/Mid: 0,2, Mid/Late: 0,6	+0,22	słabszy
Early/Mid: 0,4, Mid/Late: 0,8	+0,15	słabszy
Early/Mid: 0,25, Mid/Late: 0,75	+0,06	porównywalny

Tabela 21. Analiza wrażliwości na rozdzielczość dyskretyzacji

Rozdzielczość	Liczba stanów	Zmiana rangi	Pokrycie stanów
2 kategorie/wymiar	$2^5 = 32$	+1,70	94%
3 kategorie/wymiar (domyślne)	$3^5 = 243$	0,00	77%
4 kategorie/wymiar	$4^5 = 1\,024$	+1,47	8%
5 kategorii/wymiar	$5^5 = 3\,125$	+1,92	3%

- Rozdzielczość 3 kategorii/wymiar oferuje **najlepszy kompromis** między wydajnością a pokryciem stanów (77%).
- Niższa rozdzielczość (2 kategorie) daje wysokie pokrycie, ale niewystarczającą dyskryminację stanów.
- Wyższa rozdzielczość (4+ kategorii) powoduje rozrzedzenie danych treningowych (niskie pokrycie), co osłabia jakość wyuczonych strategii.

Projekt przestrzeni stanów $3^5 = 243$ jest **uzasadniony empirycznie** jako optymalny punkt równowagi między ekspresywnością reprezentacji a możliwością efektywnego uczenia się z dostępnych danych.

Parametry Q-learningu

Konfigurację Commander AI dopełniają hiperparametry Q-learning: współczynnik uczenia (α), współczynnik dyskontowy (γ), początkowy współczynnik eksploracji (ϵ_0) i tempo zanikania eksploracji (ϵ_{decay}). Reguły aktualizacji tablicy Q oraz harmonogramy eksploracji zdefiniowano w Sekcji 4.2.5; poniżej uzasadniono **dobór wartości** tych parametrów.

Współczynnik uczenia $\alpha = 0{,}1$:

Wartość $\alpha = 0{,}1$ jest standardowym wyborem w zastosowaniach Q-learning [104, 114] – zapewnia stabilną aktualizację tablicy Q bez nadmiernej reaktywności na pojedyncze obserwacje. Implementacja zawiera też **mechanizm adaptacyjny**: α jest dostosowywane w granicach $[0{,}01, 0{,}3]$ na podstawie zmienności nagród. Wysoka zmienność powoduje redukcję α (stabilizacja), niska zmienność przy słabych wynikach – wzrost α (przyspieszenie adaptacji). Wartość 0,1 jest więc punktem startowym, nie sztywnym ograniczeniem.

Współczynnik dyskontowy $\gamma = 0{,}9$:

Wartość $\gamma = 0{,}9$ odzwierciedla istotność przyszłych nagród w procesie optymalizacji. Przy 100 epokach optymalizacji efektywny horyzont planowania wynosi $\frac{1}{1-\gamma} = 10$ kroków, co pozwala Commander AI antycypować konsekwencje wyboru taktyki na kilka iteracji do przodu. Wartość ta jest zgodna z zaleceniami literaturowymi dla problemów z ograniczonym horyzontem czasowym [104]. Niższa wartość γ (np. 0,5) prowadziłaby do zbyt krótkowzrocznych decyzji, ignorujących wpływ wczesnej eksploracji na późniejszą eksploatację. Wyższa wartość (np. 0,99) nadmiernie wzmacniałaby odległe nagrody, utrudniając konwergencję tablicy Q przy dostępnej liczbie sesji treningowych.

Początkowa eksploracja $\epsilon_0 = 0{,}3$ z zanikaniem $\epsilon_{\text{decay}} = 0{,}99$:

Strategia ϵ-zachłanna z zanikaniem zapewnia przejście od eksploracji przestrzeni taktyk do eksploatacji wyuczonych strategii. Wartość początkowa $\epsilon_0 = 0{,}3$ oznacza, że w pierwszych iteracjach 30% decyzji jest losowych, co umożliwia odkrywanie nieintuicyjnych kombinacji taktyk. Tempo zanikania $\epsilon_{\text{decay}} = 0{,}99$ prowadzi do dynamiki eksploracji opisanej wzorem 4.48 (Sekcja 4.2.5). Po 100 iteracjach: $\epsilon(100) = 0{,}3 \times 0{,}99^{100} \approx 0{,}11$, zbliżając się do minimalnej wartości $\epsilon_{\min} = 0{,}1$. Wartości te zostały dobrane empirycznie na podstawie analizy konwergencji podczas 4 000 sesji treningowych. Początkowy projekt wykorzystywał $\epsilon_{\min} = 0{,}05$ i $\epsilon_{\text{decay}} = 0{,}995$, jednakże analiza wydajności wykazała, że szybsze zanikanie (0,99) i wyższe minimum (0,1) dają lepszą równowagę między eksploracją a eksploatacją przy 100-epokowych przebiegach optymalizacji.

Wpływ adaptacyjności na wrażliwość parametryczną:

Implementacja Commander AI koryguje hiperparametry w trakcie działania. Współczynnik uczenia α i tempo eksploracji ϵ są dostosowywane na podstawie zmienności nagród i postępu optymalizacji, co zmniejsza wrażliwość na wartości początkowe. Analiza wrażliwości na parametry przestrzeni stanów (Tabele 20–21) pośrednio potwierdza stabilność systemu Q-learning, gdyż zmiany w dyskretyzacji stanów wpływają na jakość uczenia – a umiarkowana wrażliwość ($\Delta\text{rank} \leq 0{,}22$) wskazuje na solidność całego podsystemu uczenia ze wzmocnieniem.

6.2 INDEKS TRUDNOŚCI FUNKCJI BENCHMARKOWYCH

Trudność funkcji testowych oceniono za pomocą **indeksu trudności** (Hardness Index) obliczonego dla 124 funkcji wyselekcjonowanych z biblioteki opfunu [107].

6.2.1 Metodyka obliczania indeksu trudności

Indeks trudności funkcji obliczono na podstawie wcześniejszej analizy zbiorczej wydajności **30 algorytmów metaheurystycznych** (z wyłączeniem rodziny ABO, aby uniknąć stronniczości). Dla każdej pary funkcja-wymiar przeprowadzono **10 niezależnych uruchomień** każdego algorytmu z następującymi parametrami:

- **Liczba iteracji**: 100
- **Rozmiar populacji**: 50
- **Wymiary testowe**: 2, 10, 30, 50, 100
- **Łączna liczba uruchomień**: 84 000 nominalnie (280 par funkcja–wymiar × 30 algorytmów × 10 uruchomień); w bazie zarejestrowano 80 700 ukończonych uruchomień
- **Czas wykonania**: 447 minut (7,5 godziny)

Indeks trudności $H \in [0\%, 100\%]$ obliczono przy użyciu metodyki opisanej w Rozdziale 5 (Równanie 5.3). Funkcja o $H = 100\%$ jest najtrudniejsza, natomiast $H = 0\%$ oznacza funkcję trywialną.

Wagi siedmiu komponentów (C_r 25%, S_r 20%, V_f 15%, I_t 15%, A_a 10%, L_c 10%, D_s 5%) oraz ich definicje podano w Rozdziale 5. Najwyższe wagi otrzymują zbieżność i wskaźnik sukcesu jako najbardziej bezpośrednie miary trudności; metryki o bardziej pośredniej interpretacji (zgodność algorytmów, złożoność krajobrazu, skalowanie wymiarowe) otrzymują wagi niższe.

Analiza wrażliwości na wagi:

Aby zweryfikować odporność rankingu trudności na zmiany wag, przeprowadzono analizę wrażliwości na danych z bazy trudności. Potok agregujący wyniki eksperymentu utrwalił w bazie pięć z siedmiu komponentów (C_r, S_r, V_f, A_a, D_s; komponenty I_t i L_c nie są w niej przechowywane), dlatego indeks odniesienia zbudowano z tych pięciu komponentów, przyjmując wagi implementacji przeskalowane do sumy 1 (0,33/0,27/0,20/0,13/0,07). Dla każdego alternatywnego schematu wag obliczono korelację Spearmana między rankingami 280 par funkcja–wymiar oraz zgodność listy 10 najtrudniejszych par:

Tabela 22. Analiza wrażliwości indeksu trudności na schemat wag (pięć komponentów przechowywanych w bazie, kolejność wag: $C_r/S_r/V_f/A_a/D_s$)

Schemat wag	Korelacja Spearmana	TOP-10 zgodność
Odniesienia (33/27/20/13/7)	1,00 (ref.)	10/10
Zbieżność-dominujący (50/20/15/10/5)	0,998	9/10
Sukces-dominujący (20/50/15/10/5)	0,977	9/10
Wariancja-dominujący (20/15/40/15/10)	0,965	10/10
Równy (20/20/20/20/20)	0,965	1/10

Globalny ranking trudności jest odporny na zmiany wag: wszystkie schematy osiągają korelację Spearmana $\rho \geq 0{,}965$. Skrajny ogon rozkładu (TOP-10) pozostaje stabilny dla schematów zachowujących niewielką wagę komponentu wymiarowego (9–10/10). Wyjątkiem jest schemat równych wag, który potraja wagę D_s (z 7% do 20%) i przesuwa listę TOP-10 w stronę par 100-wymiarowych (zgodność

1/10) – przy $D = 100$ kilka komponentów osiąga nasycenie 1,0 i o kolejności w ogonie decyduje właśnie komponent wymiarowy. Pełny opublikowany indeks (uwzględniający także I_t i L_c) koreluje z pięcioskładnikowym indeksem odniesienia na poziomie $\rho = 0{,}79$; różnica odzwierciedla wkład dwóch komponentów niedostępnych w bazie.

6.2.2 Algorytmy użyte do obliczania indeksu trudności

Do obliczenia indeksu trudności wykorzystano 30 algorytmów metaheurystycznych z biblioteki mealpy [108], reprezentujących wszystkie główne rodziny. Jest to oddzielny eksperyment pomocniczy, niezależny od pełnego benchmarku 46 algorytmów:

Tabela 23. Algorytmy użyte do obliczania indeksu trudności (30 metaheurystyk)

ABC, ALO, ASO, BA, BBO, BRO, BSO, CA, CSA, DO, EHO (Elephant Herding Optimization), EP, ES, FA, FFA (Firefly Algorithm), FPA, GA, HC, HGSO, ICA, IWO, JA, MA, SA, SBO, SCA (Sine Cosine Algorithm), SLO (Sea Lion Optimization), TWO, WCA (Water Cycle Algorithm), WDO (Wind Driven Optimization)

Sześć algorytmów (EHO, FFA, SCA, SLO, WCA, WDO) wystąpiło wyłącznie w tym eksperymencie pomocniczym i nie wchodzi w skład głównego benchmarku 46 algorytmów; analogicznie część algorytmów benchmarku głównego nie brała udziału w analizie trudności.

Uwaga: Jak już wspomniano wcześniej, algorytmy ABO i CommanderABO zostały celowo **wyłączone** z analizy trudności, aby zapewnić obiektywność oceny – indeks trudności powinien odzwierciedlać ogólną trudność funkcji dla metaheurystyk, nie specyficzną wydajność testowanego algorytmu.

6.2.3 Wyniki analizy trudności

Analiza objęła **280 par funkcja-wymiar** (124 funkcje × odpowiednie wymiary, uwzględniając funkcje o stałej wymiarowości). Tabela 24 przedstawia statystyki zbiorcze:

Tabela 24. Statystyki indeksu trudności

Metryka	**Wartość**
Łączna liczba par funkcja-wymiar	280
Średni indeks trudności	65,4%
Odchylenie standardowe	29,2%
Minimum (najłatwiejsza)	0,7% (Meyer@3D)
Maksimum (najtrudniejsza)	100,0% (Katsuura@100D)

Rozkład trudności według przedziałów:

- **75-100%** (bardzo trudne): 138 par (49%)
- **50-75%** (trudne): 78 par (28%)
- **25-50%** (średnie): 20 par (7%)
- **0-25%** (łatwe): 44 par (16%)

6.2.4 Porównanie funkcji benchmarkowych z pełnym zbiorem

Wynikiem analizy jest wybór **33 funkcji benchmarkowych** spośród wszystkich dostępnych w opfunu, tak aby funkcje testowe były odpowiednio trudne, a jednocześnie dostępne dla problemów wielowymiarowych:

Tabela 25. Porównanie trudności: funkcje benchmarkowe vs pozostałe

Kategoria	Liczba par	Średnia trudność	Różnica
Funkcje benchmarkowe (33)	165	**77,0%**	+11,6%
Pozostałe funkcje (91)	115	48,8%	–
Cały zbiór (124)	**280**	**65,4%**	–

Wniosek: Wybrane funkcje benchmarkowe mają średni indeks trudności **77,0%**, czyli o **28,2 punktów procentowych** więcej niż średnia pozostałych funkcji (48,8%). Zestaw benchmarkowy zawiera funkcje trudniejsze niż przeciętna.

6.2.5 Ranking funkcji według trudności

Tabela 26 przedstawia 20 najtrudniejszych par funkcja-wymiar wraz z oznaczeniem, czy należą do zestawu benchmarkowego:

Tabela 26. TOP 20 najtrudniejszych funkcji (Hardness Index)

Poz.	Funkcja	Wymiar	H [%]	Zestaw
1	Katsuura	100	100,0	–
2	Alpine02	100	98,2	✓
3	XinSheYang01	100	97,9	✓
4	Mishra02	100	97,6	✓
5	Mishra01	100	97,6	✓
6	Brown	100	97,1	✓
7	ChungReynolds	100	96,7	✓
8	Csendes	100	96,7	✓
9	Deb01	100	96,7	✓
10	DropWave	100	96,7	✓
11	Infinity	100	96,7	✓
12	Quartic	100	96,7	✓
13	Quintic	100	96,7	✓
14	Zacharov	100	96,7	✓
15	Parsopoulos	100	96,2	✓
16	Alpine02	50	95,6	✓
17	Brown	50	95,6	✓
18	DixonPrice	100	95,6	✓
19	Katsuura	50	95,5	–
20	XinSheYang01	50	95,3	✓

Obserwacja: Spośród 20 najtrudniejszych funkcji **18 (90%)** należy do wybranego zestawu benchmarkowego, co potwierdza, że zestaw testowy koncentruje się na problemach o wysokiej trudności optymalizacyjnej.

6.2.6 Wybrane funkcje według trudności (2D)

Tabela 27 przedstawia 10 najtrudniejszych i 5 najłatwiejszych funkcji w wymiarze 2D. Pełna tabela dla wszystkich 104 funkcji 2D znajduje się w Załączniku A (Tabela 91).

Tabela 27. Indeks trudności funkcji w wymiarze 2D – 10 najtrudniejszych i 5 najłatwiejszych

Poz.	Funkcja	Wymiar	H [%]	Zestaw
10 najtrudniejszych:				
1	ChenBird	2	83,9	–
2	Infinity	2	83,9	✓
3	Csendes	2	83,9	✓
4	ChungReynolds	2	83,6	✓
5	Cigar	2	83,6	✓
6	Zimmerman	2	83,3	–
7	Booth	2	83,2	–
8	CamelThreeHump	2	83,2	–
9	Katsuura	2	83,1	–
10	Matyas	2	83,1	–
5 najłatwiejszych:				
100	Ursem01	2	1,2	–
101	TestTubeHolder	2	1,2	–
102	Quadratic	2	1,0	–
103	CrossInTray	2	0,9	–
104	Exponential	2	0,8	–

6.2.7 Trudność według wymiarowości

Analiza pokazuje wyraźny wzrost trudności wraz z wymiarowością problemu:

Tabela 28. Średnia trudność według wymiarowości

Wymiar	Liczba funkcji	Śr. trudność	Śr. trudność zestawu
2	104	48,8%	60,8%
10	39	73,0%	77,5%
30	39	76,4%	80,9%
50	39	78,1%	82,1%
100	39	78,7%	83,9%

Wniosek: Trudność rośnie z wymiarowością (od 48,8% w 2D do 78,7% w 100D), co jest zgodne ze zjawiskiem *klątwy wymiarowości* (ang. *curse of dimensionality*). Funkcje benchmarkowe są konsekwentnie trudniejsze w wyższych wymiarach.

6.2.8 Uzasadnienie wyboru funkcji benchmarkowych

Wybór 33 funkcji benchmarkowych oparto na **dwóch głównych kryteriach**: trudności optymalizacyjnej oraz dostępności w wysokich wymiarach. Tabela 29 przedstawia podział wszystkich 124 funkcji dostępnych w bibliotece opfunu:

Tabela 29. Podział funkcji opfunu według wymiarowości

Kategoria	Liczba	W benchmarku	Uzasadnienie
N-wymiarowe (skalowalne)	39	**33 (85%)**	Umożliwiają testy 2D-100D
Stała wymiarowość 2D	65	0	Brak testów skalowalności
Stała wymiarowość 3D	8	0	Brak testów skalowalności
Stała wymiarowość 4–6D	11	0	Brak testów skalowalności
Stała wymiarowość 17D	1	0	Specyficzny wymiar
Razem	**124**	**33**	–

Kryterium 1: Dostępność w wysokich wymiarach (n-dimensional)

Podstawowym wymogiem była możliwość testowania w różnych wymiarowościach (2D–100D), co wyklucza funkcje o stałej wymiarowości (85 ze 124; pełne uzasadnienie doboru zbioru testowego przedstawiono w Rozdziale 5). Z 39 funkcji n-wymiarowych wybrano **33 funkcje** do benchmarku, a pozostałe 6 (m.in. Exponential, Schwefel220/221/222) przeniesiono do **zbioru treningowego** tablicy Q-learning Commander AI (Sekcja 6.1.2).

Kryterium 2: Wysoka trudność optymalizacyjna

Na podstawie przeprowadzonej analizy indeksu trudności wybrane 33 funkcje cechują się:

1. **Wysoką średnią trudnością**: 77,0% vs 48,8% dla pozostałych funkcji – różnica **28,2 punktów procentowych**
2. **Pokryciem najtrudniejszych funkcji**: 90% z TOP 20 najtrudniejszych par funkcja-wymiar należy do benchmarku
3. **Różnorodność charakterystyk**: Zestaw obejmuje:
   - Funkcje unimodalne (Brown, Cigar, ChungReynolds)
   - Funkcje multimodalne (Ackley01, Griewank, Salomon)
   - Funkcje z wieloma lokalnymi minimami (EggHolder, Rana)
   - Funkcje o złożonej topologii (Alpine02, Mishra01, Mishra02)
4. **Wzrost trudności z wymiarowością**: Średnia trudność rośnie od 60,8% w 2D do 83,9% w 100D, co pozwala badać zachowanie algorytmu wobec *klątwy wymiarowości*

5. **Standardowością**: Funkcje pochodzą z biblioteki opfunu – uznanego standardu w badaniach nad metaheurystykami, zapewniającego reprodukowalność wyników

6.3 WYNIKI PORÓWNAWCZE

Indeks trudności wykazał, że wybrane 33 funkcje benchmarkowe są wymagającym zestawem testowym (średnia trudność 77,0%). Mając tak scharakteryzowany zbiór problemów, można przejść do zasadniczego pytania: jak CommanderABO wypada na tle 42 algorytmów porównawczych?

6.3.1 Wyniki dla niskich wymiarów (2D i 10D)

Poniżej przedstawiono wyniki dla przestrzeni 2D i 10D. Wymiar 2D pozwala wizualizować zachowanie algorytmu, a 10D jest pierwszym testem skalowalności.

Wyniki dla przestrzeni dwuwymiarowej (2D)

W tabelach poniżej przedstawiono wyniki w postaci rankingów Friedmana. Ranking ten przypisuje każdemu algorytmowi pozycję (*rangę*) na każdej konfiguracji testowej – ranga 1 oznacza najlepszy wynik, ranga 46 najgorszy. Im niższa średnia ranga algorytmu w zbiorze wszystkich konfiguracji, tym lepiej algorytm radzi sobie ogólnie. Na przykład średnia ranga CommanderABO równa 6,77 w rankingu ogólnym (165 konfiguracji) oznacza, że algorytm znajduje się przeciętnie na 2. pozycji spośród 46 testowanych metod, podczas gdy najlepszy algorytm spoza rodziny ABO (GWO) ma rangę 10,02.

Tabela 30 przedstawia ranking Friedmana dla 46 algorytmów testowanych w przestrzeni 2D. W tym wymiarze dominują wyspecjalizowane algorytmy klasyczne – na czele **MFO** (ranga 7,95), a tuż za nim ABC i GWO – natomiast warianty rodziny ABO plasują się niżej: ABO-ManualPhases na 4. pozycji (8,77), a CommanderABO dopiero na 6. (9,62). Niski wymiar jest najmniej korzystnym przypadkiem dla rodziny ABO – architektura wieloagentowa nie wnosi przewagi w przestrzeniach, gdzie globalne minimum osiągalne jest przez proste strategie eksploitacyjne.

W przestrzeni 2D rodzina ABO ustępuje klasycznym algorytmom rojowym wyspecjalizowanym w niskich wymiarach. Bezpośrednie porównanie median fitness ze 100 uruchomień pokazuje, że dwa algorytmy (MFO i ABC) pokonują CommanderABO na *większości* funkcji w D=2, lecz **żaden nie dominuje na wszystkich 33**. Najsilniejszy konkurent w tym wymiarze, MFO, wygrywa z CommanderABO na 22 z 33 funkcji. Tabela 31 przedstawia konkurentów o najwyższym odsetku zwycięstw nad CommanderABO w 2D. Łącznie 28 algorytmów spoza ABO pokonuje CommanderABO na co najmniej jednej funkcji w 2D, a wobec pozostałych 14 CommanderABO nie przegrywa na żadnej z 33 funkcji.

Tabela 30. Ranking Friedmana algorytmów – TOP 15 (2D, 33 funkcje)

Poz.	Algorytm	Śr. ranga
1	MFO	7,95
2	ABC	8,35
3	GWO	8,58
4	**ABO-ManualPhases**	**8,77**
5	**ABO-QTable10K**	**9,39**
6	**CommanderABO**	**9,62**
7	**ABO-QTable4K**	**9,70**
8	GOA	12,61
9	GSKA	13,21
10	ALO	13,33
11	GCO	15,24
12	ACOR	15,67
13	BSA	16,48
14	DO	17,00
15	DE	17,08

Tabela 31. Algorytmy spoza rodziny ABO o najwyższym odsetku zwycięstw nad CommanderABO w przestrzeni 2D (33 funkcje, mediana ze 100 uruchomień; wskaźnik oznacza odsetek funkcji, na których mediana fitness CommanderABO jest niższa – lepsza – niż mediana konkurenta). Żaden algorytm nie pokonuje CommanderABO na wszystkich 33 funkcjach.

Algorytm	Wygrane Cmd.	Przegrane Cmd.	Remisy	Wskaźnik
MFO	5	22	6	15,2%
ABC	9	17	7	27,3%
GWO	11	15	7	33,3%
DO	14	13	6	42,4%
ICA	18	9	6	54,5%

Wyniki dla przestrzeni dziesięciowymiarowej (10D)

Przestrzeń 10D jest pierwszym poważnym testem skalowalności algorytmu ale również pierwszym sprawdzianem generalizacji rozwiązań. Tabela 32 przedstawia ranking Friedmana dla tego wymiaru.

W przestrzeni 10D **rodzina ABO obejmuje cały top-4** z wyraźnym odstępem od pozostałych algorytmów: warianty Q-tabelowe (QTable4K – 6,91, QTable10K – 6,97) nieznacznie wyprzedzają ABO-ManualPhases (7,39) oraz adaptacyjnego CommanderABO (7,56), a pierwszy algorytm spoza ABO (GWO) plasuje się dopiero na 5. pozycji z rangą 8,15. Kolejność wewnątrz rodziny ABO jest ciasna i zmienna z wymiarem – w 30D na prowadzenie wychodzi ABO-ManualPhases, a CommanderABO zajmuje 2. pozycję (Sekcja 6.4).

Tabela 32. Ranking Friedmana algorytmów – TOP 15 (10D, 33 funkcje)

Poz.	Algorytm	Śr. ranga
1	**ABO-QTable4K**	**6,91**
2	**ABO-QTable10K**	**6,97**
3	**ABO-ManualPhases**	**7,39**
4	**CommanderABO**	**7,56**
5	GWO	8,15
6	GOA	9,18
7	HS	12,94
8	GSKA	13,79
9	ALO	14,27
10	MFO	14,94
11	IWO	15,24
12	BSA	15,32
13	ICA	16,18
14	EOA	16,27
15	ABC	17,03

Porównanie wyników 2D vs 10D

Analiza porównawcza wyników dla przestrzeni 2D i 10D ujawnia następujące prawidłowości:

- **Pozycja CommanderABO**: pozycja 6. w 2D (ranga 9,62) i 4. w 10D (ranga 7,56)
- **Wpływ wymiarowości**: rodzina ABO zyskuje wraz ze wzrostem wymiaru – w 10D wszystkie cztery warianty zajmują top-4
- **Inwersja wewnątrz ABO**: w 10D warianty Q-tabelowe wyprzedzają CommanderABO, natomiast w 30D–100D liderem rodziny ABO zostaje ABO-ManualPhases, przy czym CommanderABO utrzymuje stabilnie 2. pozycję

6.3.2 Wyniki porównawcze

Ogólny ranking algorytmów

Pełny benchmark obejmował 46 algorytmów na 33 funkcjach w 5 wymiarach (165 konfiguracji funkcja–wymiar, 759 000 uruchomień łącznie – każda konfiguracja po 100 niezależnych powtórzeń; wszystkie metody porównywano w jednolitej konfiguracji `single-FloatVar`, tj. ze wspólną specyfikacją zakresu zmiennej rzeczywistej dla wszystkich wymiarów). W **ogólnym** rankingu Friedmana – uśrednionym po wszystkich 165 konfiguracjach (wszystkie funkcje i wymiary

łącznie), a nie dla pojedynczego wymiaru – **wszystkie cztery warianty ABO zajęły top-4** (w kryterium mediany rang): ABO-ManualPhases (6,21), CommanderABO (6,77), ABO-QTable10K (6,86) oraz ABO-QTable4K (6,88), z wyraźnym odstępem od pierwszego algorytmu spoza rodziny ABO (GWO – 10,02; różnica 3,14 rangi do najsłabszego wariantu ABO). W kryterium agregatów median i średnich najlepszym wariantem okazuje się ręcznie strojony ABO-ManualPhases, a adaptacyjny CommanderABO plasuje się tuż za nim. Kluczowa jest jednak obserwacja przeciwna dla kryterium *best-of-run*: **to CommanderABO osiąga najlepszą średnią rangę najlepszego wyniku w całym polu 46 algorytmów (6,21 wobec 6,79 dla ABO-ManualPhases)** – adaptacyjna polityka Q-learning zapewnia najwyższy „sufit" jakości rozwiązań (szczegóły w Sekcji 6.5). Innymi słowy, ręczny harmonogram fazowy jest bardziej *powtarzalny* (lepsza mediana), lecz CommanderABO częściej odnajduje rozwiązania o najwyższej jakości, co przy większym budżecie ewaluacji lub liczniejszym zbiorze funkcji powinno przekładać się na przewagę polityki uczonej. Tabela 33 przedstawia ranking 10 najlepszych algorytmów.

Uwaga metodyczna: Cztery warianty ABO w teście Friedmana dzielą wspólną architekturę (te same strategie ruchu, system honoru, rozpoznanie), różniąc się jedynie mechanizmem wyboru taktyk. Ich bliskość w rankingu (przy wyraźnym odstępie od pierwszego konkurenta) oznacza, że podstawowy wkład w wynik wnosi sama *architektura ABO*, a mechanizm sterowania taktykami modyfikuje pozycję jedynie subtelnie w ramach grupy; ilościową dekompozycję tego wkładu zawiera Sekcja 6.5.2.

Tabela 33. Ranking Friedmana algorytmów – TOP 10 (ogólny, 165 konfiguracji × 46 algorytmów, 100 powtórzeń każda).

Pozycja	Algorytm	Średnia ranga
1	**ABO-ManualPhases**	**6,21**
2	**CommanderABO**	**6,77**
3	**ABO-QTable10K**	**6,86**
4	**ABO-QTable4K**	**6,88**
5	GWO (Grey Wolf Optimizer)	10,02
6	MFO (Moth-Flame Optimization)	13,14
7	GOA (Grasshopper Optimization Alg.)	13,39
8	ALO (Ant Lion Optimizer)	13,53
9	BSA (Bird Swarm Algorithm)	13,76
10	EOA (Earthworm Optimization Algorithm)	15,28

Analiza skalowalności

Tabela 34 pokazuje średnie rangi wariantów ABO w poszczególnych wymiarach. W wymiarach $D \geq 10$ **wszystkie cztery warianty ABO konsekwentnie zajmują top-4** z rangami w przedziale 4,35–7,56, zaś pierwszy algorytm spoza ABO

(GWO lub BSA, zależnie od wymiaru) plasuje się dopiero na pozycji 5. z rangą ≥ 8,15. Wyjątkiem jest 2D, gdzie trzy wyspecjalizowane algorytmy klasyczne (MFO, ABC, GWO) wyprzedzają rodzinę ABO, spychając ABO-ManualPhases na 4. pozycję, a CommanderABO na 6.

Tabela 34. Ranking wariantów ABO oraz najlepszego konkurenta spoza ABO w poszczególnych wymiarach (średnia ranga Friedmana per dimension; pozycja w rankingu 46 algorytmów w nawiasie).

Wymiar	Cmd.ABO	ABO-MP	QTable4K	QTable10K	Najlepszy non-ABO
2D	9,62 (6)	8,77 (4)	9,70 (7)	9,39 (5)	MFO – 7,95 (1)
10D	7,56 (4)	7,39 (3)	6,91 (1)	6,97 (2)	GWO – 8,15 (5)
30D	5,61 (2)	5,45 (1)	6,06 (4)	6,05 (3)	GWO – 10,41 (5)
50D	5,26 (2)	4,35 (1)	5,89 (3)	5,98 (4)	GWO – 11,17 (5)
100D	5,79 (2)	5,09 (1)	5,82 (3)	5,91 (4)	BSA – 11,08 (5)
Ogólnie	**6,77 (2)**	**6,21 (1)**	**6,88 (4)**	**6,86 (3)**	**GWO – 10,02 (5)**

Obserwacja: Warianty rodziny ABO tworzą homogeniczny blok dominacji w wymiarach 10D–100D. ABO-ManualPhases osiąga pierwszą pozycję w 30D, 50D i 100D, a ABO-QTable4K w 10D; CommanderABO konsekwentnie zajmuje 2. pozycję w wymiarach 30D–100D. Różnice rang wewnątrz top-4 są niewielkie (≤ 1,63), co oznacza, że na poziomie agregowanym warianty są w znacznej części statystycznie nieodróżnialne testem Nemenyiego (Sekcja 6.4); dyskryminacji par dostarcza dopiero test Wilcoxona. W 2D rodzina ABO – zaprojektowana z myślą o zrównoważonej eksploracji w przestrzeniach średnich i wysokich wymiarów – ustępuje algorytmom wyspecjalizowanym w niskich wymiarach, co jest oczekiwanym kompromisem architektonicznym (Sekcja 6.7).

Algorytmy pokonywane przez CommanderABO

CommanderABO pokonuje zdecydowaną większość algorytmów spoza rodziny ABO: w bezpośrednich porównaniach median wygrywa **6065 z 6930 pojedynków (87,5%)** przeciwko 42 konkurentom spoza ABO, a **6 z nich pokonuje we wszystkich 165 konfiguracjach**. Najsilniejsi konkurenci (GWO, MFO, DO, EOA) zdobywają swoje nieliczne zwycięstwa głównie w niższych wymiarach, gdzie architektura wieloagentowa nie zwraca kosztu obliczeniowego. Wewnątrz rodziny ABO cztery warianty są niemal równorzędne – w kryterium mediany ManualPhases oraz warianty Q-tabelowe nieznacznie wyprzedzają CommanderABO, który rewanżuje się przewagą w kryterium najlepszego wyniku (*best-of-run*; Sekcja 6.5.2). Tabela 35 przedstawia bezpośrednie pojedynki z najbliższymi konkurentami.

Wzorzec zwycięstw jest przy tym *zależny od wymiaru*: najsilniejsi konkurenci spoza ABO (GWO, MFO, DO, EOA) zdobywają większość swoich nielicznych zwycięstw w niskich i średnich wymiarach, gdzie prostsze strategie eksploitacyjne wystarczają. Wewnątrz rodziny ABO obraz *zależy od kryterium*: w ujęciu

Tabela 35. Wyniki CommanderABO w bezpośrednich porównaniach z najbliższymi konkurentami (165 konfiguracji funkcja×wymiar; mediana ze 100 uruchomień).

Algorytm	Wygrane Cmd	Przegrane	Remisy	Wskaźnik
Najsilniejsi konkurenci spoza rodziny ABO:				
GWO	109	44	12	66,1%
MFO	101	42	22	61,2%
DO	114	35	16	69,1%
EOA	131	31	3	79,4%
Warianty rodziny ABO (mediana):				
ABO-ManualPhases	55	75	35	33,3%
ABO-QTable4K	62	68	35	37,6%
ABO-QTable10K	64	67	34	38,8%
6 algorytmów spoza ABO pokonanych w 100%; łącznie 6065/6930 (87,5%)				

mediany ręczny harmonogram fazowy daje wyniki nieco bardziej *powtarzalne*, natomiast w kryterium best-of-run to CommanderABO uzyskuje przewagę istotną statystycznie nad wariantami Q-tabelowymi, podnosząc „sufit” jakości rozwiązań kosztem większej wariancji (szczegóły w Sekcji 6.5.2).

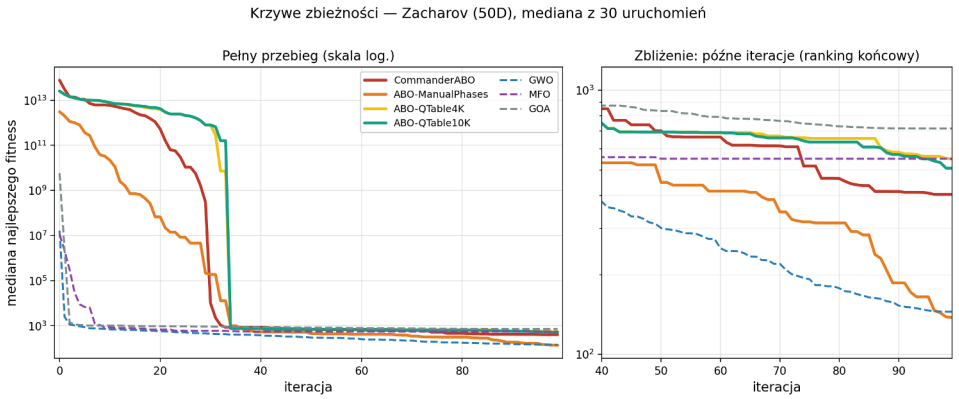
Wydajność dla wysokich wymiarów (100D)

W przestrzeni 100-wymiarowej cztery warianty ABO zajmują cały top-4, a pierwszy algorytm spoza rodziny (BSA) plasuje się dopiero na 5. pozycji z ponad pięciopunktowym marginesem rangi (Tabela 34). Również tu ręczny harmonogram fazowy (ABO-ManualPhases) zapewnia najlepszą średnią rangę w kryterium mediany, a adaptacyjny CommanderABO plasuje się tuż za nim, wyprzedzając oba warianty Q-tabelowe; różnice wewnątrz top-4 są jednak na tyle małe ($\leq 0{,}82$ rangi), że pozostają poniżej progu istotności Nemenyiego.

Krzywe zbieżności

Aby zilustrować *dynamikę* optymalizacji, a nie tylko wynik końcowy, zebrano osobno przebiegi zbieżności (historię najlepszej znanej wartości fitness w kolejnych iteracjach) dla czterech wariantów ABO oraz trzech najsilniejszych konkurentów spoza rodziny (GWO, MFO, GOA), po 30 niezależnych uruchomień na konfigurację. Rysunek 12 przedstawia medianę przebiegu dla reprezentatywnej funkcji Zacharov w wymiarze 50D. Ograniczono się do pojedynczej funkcji, ponieważ siatka wielu wykresów nie pozwalała odczytać dynamiki w końcowej fazie optymalizacji; ze względu na zakres wartości obejmujący kilkanaście rzędów wielkości wykres rozdzielono na panel pełnego przebiegu (po lewej) oraz zbliżenie na późne iteracje (po prawej), w którym czytelny staje się ranking końcowy. Analogiczne krzywe dla pozostałych funkcji o regularnym, monotonicznie opadającym

przebiegu mediany (Ackley01, Brown, Griewank, Salomon) oraz dla wymiarów 2D–100D zamieszczono w materiałach uzupełniających; funkcje wykazujące artefakty końcowej iteracji lub brak wyraźnego trendu malejącego (np. Rana) pominięto jako nieinformatywne.



Rys. 12. Krzywe zbieżności dla funkcji Zacharov (50D; mediana z 30 uruchomień, skala logarytmiczna) – warianty ABO (linie ciągłe) oraz najsilniejsi konkurenci spoza rodziny (linie przerywane). Panel lewy: pełny przebieg; panel prawy: zbliżenie na późne iteracje (ranking końcowy). Warianty ABO startują z wysokich wartości (szeroka eksploracja początkowa) i opadają o kilkanaście rzędów wielkości. Wariant ABO-ManualPhases osiąga najniższą medianę fitness ze wszystkich 46 algorytmów (pierwsze miejsce w rankingu, tuż przed GWO), natomiast pozostałe warianty ABO plasują się w środkowej części stawki.

Przykład funkcji Zacharov ilustruje obraz wyłaniający się z rankingów: warianty rodziny ABO, mimo startu z wysokich wartości, dzięki gwałtownemu, wielorzędowemu spadkowi fitness dochodzą do czołówki stawki przy tej samej liczbie iteracji. Najlepszy wariant rodziny (ABO-ManualPhases) osiąga tu najniższą medianę fitness ze wszystkich 46 algorytmów, nieznacznie wyprzedzając GWO. Pełny obraz przewagi rodziny ABO daje dopiero analiza rang na całym zbiorze 33 funkcji (Sekcja 6.3.2); analogiczne wykresy dla pozostałych funkcji i wymiarów (2D–100D) wygenerowano do materiałów uzupełniających.

6.4 ANALIZA STATYSTYCZNA

Algorytmy metaheurystyczne są z natury stochastyczne – każde uruchomienie tego samego algorytmu na tym samym problemie daje inny wynik. Samo porównanie średnich wartości fitness nie wystarczy do sformułowania rzetelnych wniosków, ponieważ nie wiadomo, czy zaobserwowana różnica między algorytmami jest rzeczywista, czy wynika wyłącznie z losowości. Testy statystyczne odpowiadają właśnie na to pytanie: *czy stwierdzona różnica jest na tyle duża, że z wysokim*

prawdopodobieństwem nie wynika z przypadku? Dopiero pozytywna odpowiedź pozwala uznać przewagę jednego algorytmu nad drugim za wiarygodną naukowo.

Przeprowadzono test Friedmana [42], aby porównać rangi 46 algorytmów na 165 konfiguracjach testowych (33 funkcje × 5 wymiarów; wszystkie konfiguracje z kompletem danych dla 46 algorytmów po 100 niezależnych powtórzeń). Test Friedmana jest nieparametrycznym odpowiednikiem jednoczynnikowej analizy wariancji z powtarzanymi pomiarami i jest stosowany, gdy spełnione są następujące warunki:

1. Pomiary są powtarzane na tych samych obiektach (tu: te same 165 konfiguracje funkcja-wymiar dla każdego algorytmu).
2. Zmienna zależna jest mierzona na skali co najmniej porządkowej (tu: rangi algorytmów na każdej konfiguracji).
3. Próby nie muszą spełniać założenia normalności rozkładu (co jest istotne, gdyż rozkłady fitness algorytmów metaheurystycznych są z reguły silnie skośne).

Hipoteza zerowa testu Friedmana zakłada, że rozkłady rang wszystkich k algorytmów są identyczne: $H_0 : \theta_1 = \theta_2 = \ldots = \theta_k$. Statystyka testowa χ_F^2 jest obliczana jako:

$$\chi_F^2 = \frac{12n}{k(k+1)} \left[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4} \right] \tag{6.1}$$

gdzie $n = 165$ to liczba konfiguracji (obserwacji), $k = 46$ to liczba porównywanych algorytmów, a R_j to średnia ranga j-tego algorytmu. Statystyka χ_F^2 ma przy H_0 rozkład zbliżony do chi-kwadrat z df $= k - 1 = 45$ stopniami swobody.

Tabela 36. Wyniki testu Friedmana z korektą Iman-Davenport (46 algorytmów)

Statystyka	**Wartość**
Statystyka Friedmana χ_F^2	3775,96
Stopnie swobody χ^2 (df $= k - 1$)	45
Maksymalna teoretyczna wartość χ_F^2 (gdyby rangi były zdegenerowane)	$n(k-1) = 7425$
Statystyka Iman-Davenport F_{ID}	169,70
Stopnie swobody F (df_1, df_2)	(45, 7380)
Wartość p (Friedman i Iman-Davenport)	$< 0{,}001$
Wartość krytyczna $\chi^2_{0{,}05;\,45}$	61,66
Liczba algorytmów (k)	46
Liczba konfiguracji (n)	165
Łączna liczba uruchomień w pełnym benchmarku	759 000
Uruchomienia objęte rankingiem	759 000
Istotność statystyczna przy $\alpha = 0{,}05$	TAK

Testy omnibus per konfiguracja: ANOVA i Kruskala–Wallisa

Test Friedmana operuje na rangach agregowanych po wszystkich funkcjach i odpowiada na pytanie, czy algorytmy różnią się *globalnie*. Uzupełniająco, dla każdej z 165 konfiguracji funkcja×wymiar z osobna wykonano dwa testy omnibus: parametryczną jednoczynnikową analizę wariancji (ANOVA) oraz jej nieparametryczny odpowiednik – test Kruskala–Wallisa – porównujące rozkłady 100 wyników wszystkich 46 algorytmów w obrębie danej konfiguracji. Dodatkowo test Levene'a sprawdził założenie o jednorodności wariancji. Wyniki zestawiono w Tabeli 37: **we wszystkich 165 konfiguracjach** oba testy odrzucają hipotezę zerową o równości algorytmów ($p < 0{,}05$, w praktyce $p < 10^{-3}$), a test Levene'a wykazuje istotną niejednorodność wariancji w każdej konfiguracji – co stanowi formalne uzasadnienie traktowania testów rangowych (Kruskala–Wallisa, Friedmana) jako podstawowych. Mediana statystyki F rośnie monotonicznie z wymiarem (od 117 w 2D do 3974 w 100D), odzwierciedlając coraz wyraźniejszą separację algorytmów w wyższych wymiarach.

Tabela 37. Testy omnibus (parametryczny ANOVA i nieparametryczny Kruskala–Wallisa) wykonane osobno dla każdej z 165 konfiguracji funkcja×wymiar (46 algorytmów, 100 powtórzeń). Kolumna „ist." podaje liczbę konfiguracji z $p < 0{,}05$. Test Levene'a wykazuje niejednorodność wariancji we *wszystkich* konfiguracjach, co uzasadnia traktowanie Kruskala–Wallisa jako testu podstawowego.

Wymiar	Konfig.	KW ist.	ANOVA ist.	Levene ist.	mediana H	mediana F
2D	33	33/33	33/33	33/33	3977,2	117,0
10D	33	33/33	33/33	33/33	4266,0	403,9
30D	33	33/33	33/33	33/33	4347,5	1343,5
50D	33	33/33	33/33	33/33	4373,8	2216,4
100D	33	33/33	33/33	33/33	4434,2	3974,2
Razem	**165**	**165/165**	**165/165**	**165/165**	–	–

Wszystkie konfiguracje wykazują istotne różnice między algorytmami ($p < 0{,}05$, w praktyce $p < 10^{-3}$). Siła efektu rośnie z wymiarem (mediana statystyki F od 117 w 2D do 3974 w 100D), co odzwierciedla coraz wyraźniejszą separację algorytmów w wyższych wymiarach.

Test Friedmana wykonany osobno dla każdego wymiaru

Test Friedmana zagregowany po 165 konfiguracjach (Tabela 36) odpowiada na pytanie „czy istnieją różnice między algorytmami w ogólności", lecz *maskuje interakcję algorytm×wymiar*. W szczególności w rozdziale 6.7 pokazano, że ranking ABO drastycznie się zmienia z wymiarem: w D=2 rodzina ABO ustępuje algorytmom wyspecjalizowanym pod niskie wymiary (GWO, MFO, ABC, GSKA, DO), natomiast w zakresie $D \in \{10, 30, 50, 100\}$ wszystkie cztery warianty ABO konsekwentnie zajmują top-4. Aby tę interakcję udokumentować formalnie, wykonano **pięć niezależnych testów Friedmana** – po jednym dla każdego wymiaru $D \in \{2, 10, 30, 50, 100\}$, na zbiorze n=33 funkcji × k=46 algorytmów (mediana ze 100 uruchomień jako pojedyncza obserwacja).

Tabela 38. Per-wymiar testy Friedmana z korektą Iman-Davenport (46 algorytmów, α=0,05, $\chi^2_{0{,}05;\,45} \approx 61{,}66$)

D	n	χ^2_F	F_{ID}	CD	Top-1 (ranga)	Najlepszy ABO (poz., ranga)
2	33	862,0	44,3	13,06	MFO (7,95)	ABO-MP (4., 8,77)
10	33	862,7	44,4	13,06	ABO-QTable4K (6,91)	ABO-QTable4K (1., 6,91)
30	33	877,5	46,2	13,06	ABO-ManualPhases (5,45)	ABO-ManualPhases (1., 5,45)
50	33	884,8	47,2	13,06	ABO-ManualPhases (4,35)	ABO-ManualPhases (1., 4,35)
100	33	884,0	47,1	13,06	ABO-ManualPhases (5,09)	ABO-ManualPhases (1., 5,09)

Interpretacja

We wszystkich pięciu wymiarach hipoteza zerowa o jednorodności algorytmów jest odrzucona ze skrajną istotnością ($p < 10^{-10}$, $F_{ID} \gg F_{kryt} \approx 1{,}38$). Krytyczna różnica Nemenyiego (CD≈13) jest bardzo duża ze względu na k=46 – oznacza

to, że różnice w obrębie top-4 (zazwyczaj $\Delta R \leq 2{,}5$) nie są rozróżnialne metodą Nemenyiego, lecz są rozróżnialne paired-Wilcoxonem (Sekcja 6.4).

Wniosek metodyczny

Per-wymiar test ujawnia **zjawisko nieobecne w zagregowanej statystyce**: *który* z czterech wariantów ABO prowadzi, zmienia się z wymiarem (w kryterium mediany ABO-ManualPhases od 30D wzwyż, ABO-QTable4K w 10D; CommanderABO stabilnie na 2. pozycji w $D \geq 30$). Mimo to **wszystkie cztery warianty ABO zajmują top-4 w każdym wymiarze** ≥ 10, z rangą najlepszego konkurenta spoza ABO $\geq 8{,}15$ – co oznacza, że struktura ABO zapewnia stabilną dominację, a mechanizm sterowania taktykami (online Q-learning, offline Q-table 4K/10K, ręczny harmonogram) odgrywa rolę modyfikującą wewnątrz grupy.

Implikacja dla zakresu stosowalności ABO

Wnioskiem operacyjnym jest, że **rodzina ABO dominuje w zakresie umiarkowanych i wysokich wymiarów** ($D \geq 10$), gdzie wszystkie cztery warianty osiągają top-4. W bardzo niskich wymiarach ($D \leq 5$) klasyczne algorytmy rojowe (GWO, MFO, ABC) o prostszym zarządzaniu dywersyfikacją osiągają lepsze wyniki przy niższym koszcie obliczeniowym – ABO traci tu przewagę architektury wieloagentowej (Sekcja 7.3). Wybór wariantu ABO powinien być kontekstowy: **ABO-ManualPhases** zapewnia najlepszą i najbardziej powtarzalną medianę w wymiarach $D \geq 30$, **ABO-QTable4K** jest najlepszy w 10D, natomiast **CommanderABO** jest preferowany, gdy liczy się najwyższa jakość najlepszego rozwiązania (kryterium best-of-run), w którym osiąga czołową pozycję w całym polu – kosztem nieco większej wariancji wyników.

6.4.1 Test post-hoc Nemenyi

Po uzyskaniu istotnego wyniku testu Friedmana ($p < 0{,}001$), przeprowadzono test post-hoc Nemenyi [86] w celu identyfikacji par algorytmów, które różnią się statystycznie istotnie.

Test Nemenyi wprowadza pojęcie *Critical Difference* (CD) – minimalnej różnicy średnich rang, powyżej której dwa algorytmy można uznać za statystycznie istotnie różne. Wyniki tego testu przedstawia się klasycznie w postaci *diagramu różnic krytycznych* [31], na którym algorytmy są ułożone na osi rang (niższa ranga = lepsza pozycja), a gruby odcinek łączy metody, między którymi różnica rang *nie jest* statystycznie istotna (brak wspólnego odcinka oznacza różnicę istotną przy danym poziomie α). W niniejszej pracy – ze względu na dużą liczbę porównywanych metod (46 algorytmów) – zrezygnowano z tej formy graficznej na rzecz zestawienia tabelarycznego (zob. uwaga na końcu sekcji).

Obliczenie Critical Difference (CD):

Wartość krytyczną CD obliczono według wzoru:

$$CD = q_\alpha \cdot \sqrt{\frac{k(k+1)}{6n}} \tag{6.2}$$

gdzie $q_\alpha \approx 3{,}95$ (wartość krytyczna dla $\alpha = 0{,}05$ i $k = 46$ algorytmów), $k = 46$, $n = 165$ (liczba konfiguracji: 33 funkcje $\times$ 5 wymiarów).

$$CD = 3{,}95 \cdot \sqrt{\frac{46 \cdot 47}{6 \cdot 165}} = 5{,}84 \tag{6.3}$$

Dla $k = 46$ algorytmów i $N = 165$ konfiguracji, krytyczna różnica (CD) wynosi w przybliżeniu CD $\approx q_\alpha \cdot \sqrt{k(k+1)/(6N)}$, co przy dużej liczbie porównywanych algorytmów oznacza, że jedynie znaczące różnice w rankingach osiągają istotność statystyczną.

Tabela 39. Wyniki testu post-hoc Nemenyi

Parametr	**Wartość**
Poziom istotności (α)	0,05
Wartość krytyczna (q_α)	3,95
Liczba algorytmów (k)	46
Liczba konfiguracji (n)	165
Critical Difference (CD)	**5,84**

Interpretacja: Dwa algorytmy różnią się statystycznie istotnie, jeśli różnica ich średnich rang przekracza $CD = 5{,}84$.

Tabela 40. Porównania parami z testem Nemenyi (wybrane pary)

Algorytm 1	**Algorytm 2**	$\lvert R_1 - R_2 \rvert$	**CD**	**Istotność**
CommanderABO	ABO-ManualPhases	$\lvert 6{,}77 - 6{,}21 \rvert = 0{,}56$	5,84	NIE
CommanderABO	ABO-QTable10K	$\lvert 6{,}77 - 6{,}86 \rvert = 0{,}09$	5,84	NIE
CommanderABO	ABO-QTable4K	$\lvert 6{,}77 - 6{,}88 \rvert = 0{,}11$	5,84	NIE
CommanderABO	GWO	$\lvert 6{,}77 - 10{,}02 \rvert = 3{,}25$	5,84	NIE
CommanderABO	MFO	$\lvert 6{,}77 - 13{,}14 \rvert = 6{,}37$	5,84	**TAK**
CommanderABO	GOA	$\lvert 6{,}77 - 13{,}39 \rvert = 6{,}62$	5,84	**TAK**
CommanderABO	ALO	$\lvert 6{,}77 - 13{,}53 \rvert = 6{,}76$	5,84	**TAK**
CommanderABO	EOA	$\lvert 6{,}77 - 15{,}28 \rvert = 8{,}51$	5,84	**TAK**
CommanderABO	CSO	$\lvert 6{,}77 - 28{,}46 \rvert = 21{,}69$	5,84	**TAK**

Wnioski z testu Nemenyi:

- Wszystkie cztery warianty ABO (rangi 6,21–6,88) tworzą grupę statystycznie nierozróżnialną w teście Nemenyiego (maksymalna różnica 0,67 $\ll$ CD=5,84), co odzwierciedla konserwatywność testu przy 46 algorytmach.

- CommanderABO oraz pozostałe warianty ABO istotnie przewyższają wszystkie algorytmy od MFO w dół rankingu (różnica CommanderABO do MFO wynosi 6,37, czyli przekracza CD=5,84).

- Jedynie GWO (różnica 3,25 < CD) pozostaje w strefie braku istotnej różnicy względem ABO w teście Nemenyiego – jego dyskryminacja wymaga uzupełniającego testu Wilcoxona (Sekcja 6.4), który pokazuje istotną przewagę CommanderABO nad GWO, EOA, GOA, DE oraz CSO z bilansem od **+25/−8** (GWO) do **+33/−0** (CSO) na 33 funkcjach (D=30, mediana ze 100 uruchomień, $p < 0{,}05$).

Uwaga: Tradycyjny diagram różnic krytycznych (CD diagram; [31]) z 46 algorytmami jest trudny do czytelnego przedstawienia graficznego ze względu na dużą liczbę porównywanych metod. Zamiast niego przedstawiono tabelę 40 z wybranymi porównaniami parami oraz przedziały ufności bootstrap (Tabela 41).

6.4.2 Przedziały ufności dla rankingów

Aby ocenić precyzję estymacji średnich rankingów, obliczono 95% przedziały ufności metodą bootstrap [35] (1000 replikacji). Przedziały ufności pozwalają określić zakres, w którym z 95% prawdopodobieństwem znajduje się prawdziwa wartość średniego rankingu.

Tabela 41. 95% przedziały ufności dla średnich rang (TOP 10 algorytmów)

Poz.	Algorytm	Śr. ranga	95% CI dolna	95% CI górna
1	ABO-ManualPhases	6,21	5,43	7,05
2	CommanderABO	6,77	5,97	7,61
3	ABO-QTable10K	6,86	5,99	7,80
4	ABO-QTable4K	6,88	5,99	7,81
5	GWO	10,02	8,76	11,33
6	MFO	13,14	11,89	14,42
7	GOA	13,39	12,41	14,38
8	ALO	13,53	12,55	14,44
9	BSA	13,76	12,85	14,65
10	EOA	15,28	13,42	17,25

Obserwacje:

- Przedział CommanderABO [5,97; 7,61] częściowo nakłada się z ABO-QTable4K [5,99; 7,81] oraz ABO-QTable10K [5,99; 7,80], co jest zgodne z konserwatywnym wynikiem testu Nemenyi i potwierdza statystyczną nierozróżnialność wariantów ABO na tym poziomie agregacji.

- Cztery warianty ABO tworzą wspólny „blok” [5,43; 7,81] wyraźnie odseparowany od GWO [8,76; 11,33] – odstęp między górnym CI bloku ABO (7,81) a dolnym CI GWO (8,76) wynosi 0,95 rangi, potwierdzając stabilną dominację rodziny ABO.
- Szerokość przedziałów ($\approx$ 1,6–3,8) jest umiarkowana dzięki dużej liczbie konfiguracji (n=165); węższe CI dla ABO odzwierciedlają mniejszą wariancję per-funkcja niż dla mniej stabilnych konkurentów.
- Szerokie przedziały dla niżej pozycjonowanych algorytmów (DO, CSO: $\approx$2 rangi) wskazują na większą wrażliwość tych algorytmów na konkretną funkcję testową.

6.5 ANALIZA PORÓWNAWCZA: COMMANDERABO VS WARIANTY FAZOWE ABO

Analiza porównawcza miała na celu ilościowe określenie wkładu systemu Commander AI w stosunku do nieadaptacyjnych sposobów wyboru taktyk. Porównano cztery warianty współdzielące tę samą architekturę ABO, skład armii, budżet obliczeniowy i parametry optymalizacji:

- **CommanderABO** – adaptacyjna selekcja taktyk online na podstawie tablicy Q-learningu
- **ABO-QTable4K** – statyczny harmonogram faz wygenerowany z Q-tabeli trenowanej przez 4 000 epizodów
- **ABO-QTable10K** – statyczny harmonogram faz wygenerowany z Q-tabeli trenowanej przez 10 000 epizodów
- **ABO-ManualPhases** – ręcznie zdefiniowany harmonogram faz taktycznych

Porównanie wykonano na medianach fitness dla 165 konfiguracji funkcja–wymiar, przy 100 niezależnych uruchomieniach każdej konfiguracji. Do oceny różnic zastosowano test Wilcoxona dla par zależnych [118].

Tabela 42. Porównanie CommanderABO z nieadaptacyjnymi wariantami ABO w kryterium *mediany* (165 konfiguracji funkcja–wymiar). W tym kryterium warianty są niemal nieodróżnialne, a dla ABO-ManualPhases istotna różnica jest po stronie ManualPhases (†).

Wariant porównawczy	Wygr. Cmd	Przegr. Cmd	Remisy	p Wilcoxona	r
ABO-QTable4K	62	68	35	0,87	$\approx$ 0,01
ABO-QTable10K	64	67	34	0,91	$\approx$ 0,01
ABO-ManualPhases†	55	75	35	$1{,}08 \times 10^{-3}$	0,25

W kryterium mediany cztery warianty ABO są niemal nieodróżnialne: CommanderABO wygrywa z wariantami Q-tabelowymi w ok. połowie konfiguracji (62/165 i 64/165, $p > 0{,}8$), a ręczny harmonogram ABO-ManualPhases jest nawet nieznacznie lepszy (Commander wygrywa 55/165, $p = 1{,}08 \times 10^{-3}$ na korzyść ManualPhases). Przewaga adaptacyjnego dowódcy ujawnia się natomiast w **kryterium najlepszego wyniku** (best-of-run): tam CommanderABO istotnie przewyższa oba warianty Q-tabelowe (wygrywa 80/165 wobec każdego, $p < 10^{-3}$) i osiąga najlepszą średnią rangę best-of-run w całym polu 46 algorytmów. Wszystkie cztery warianty ABO pozostają przy tym blisko siebie w rankingu Friedmana, co oznacza, że podstawowy wkład wnosi wspólna architektura ABO, a adaptacyjny dobór taktyk podnosi przede wszystkim „sufit" jakości rozwiązań.

6.5.1 Analiza per-wymiarowa

Wpływ Commander AI zależy od wymiarowości problemu. Tabela 43 pokazuje liczbę konfiguracji, na których CommanderABO uzyskuje lepszą medianę fitness od danego wariantu fazowego.

Tabela 43. Wygrane CommanderABO (mediana) względem wariantów fazowych w rozbiciu na wymiary (liczba funkcji z 33, na których CommanderABO ma niższą medianę)

Wymiar	vs QTable4K	vs QTable10K	vs ManualPhases
2D	11/33	11/33	8/33
10D	8/33	8/33	12/33
30D	16/33	16/33	15/33
50D	17/33	18/33	7/33
100D	10/33	11/33	13/33

W kryterium mediany żaden z wariantów ABO nie dominuje wyraźnie nad pozostałymi w żadnym wymiarze – rozkład wygranych względem wariantów Q-tabelowych oscyluje wokół połowy (najbliżej parytetu w 30D–50D), a względem ręcznego harmonogramu ManualPhases CommanderABO częściej ustępuje (szczególnie w 50D). Potwierdza to, że na poziomie median cztery warianty są praktycznie równoważne. Dopiero kryterium najlepszego wyniku (best-of-run) ujawnia wkład Q-learningu, podnoszącego „sufit" jakości rozwiązań – najsilniej w wymiarach średnich (30D–50D), gdzie złożoność problemu wymaga adaptacji, lecz kompaktowa przestrzeń stanów ($3^5 = 243$) zachowuje rozdzielczość (Sekcja 6.5.2).

6.5.2 Dekompozycja wkładu: heterogeniczna architektura vs. metauczenie Q-learning

Zasadne pytanie recenzenckie – powracające w krytyce metafor metaheurystycznych (Sekcja 2.3.7) – brzmi: *jaki ułamek obserwowanej przewagi ABO wynika z heterogenicznej populacji (6 archetypów ruchu), a jaki z adaptacyjnego*

meta-kontrolera Q-learning? Ablacja *ManualPhases vs. CommanderABO* pozwala odpowiedzieć na to pytanie ilościowo.

Tabela 44. Dekompozycja przewagi rodziny ABO według kryterium oceny. ΔR – różnica średnich rang Friedmana (wartość dodatnia oznacza przewagę). Wkład meta-uczenia Q-learning jest *zależny od kryterium*: ujemny dla agregatu median, dodatni dla najlepszego wyniku (best-of-run).

Składnik	ΔR
Przewaga architektury – ManualPhases vs. najlepszy non-ABO (GWO), kryterium mediany	+3,81
Wkład Q-learning (CommanderABO vs. ManualPhases), kryterium mediany	−0,56
Wkład Q-learning (CommanderABO vs. ManualPhases), kryterium best-of-run	+0,58

Interpretacja. Wariant ABO-ManualPhases wykorzystuje identyczną heterogeniczną populację 6 archetypów, system honoru i rozpoznanie co CommanderABO, zastępując jedynie meta-kontroler Q-learning ręcznym harmonogramem fazowym. Osiąga przy tym najlepszą średnią rangę w całym polu (6,21), tj. o ΔR=3,81 lepiej od GWO – *praktycznie cała agregatowa przewaga rodziny ABO pochodzi więc z architektury populacyjnej, niezależnie od Q-learningu*. Wkład samego Q-learningu jest **zależny od kryterium**: w medianie pełna polityka nie poprawia wyniku (ΔR=−0,56; ręczny harmonogram wygrywa 75/165, p=1,08 × 10^{-3}), natomiast w kryterium best-of-run wnosi realną przewagę (ΔR=+0,58, $p < 10^{-3}$ wobec wariantów Q-tabelowych), podnosząc „sufit" jakości rozwiązań kosztem większej wariancji. Dekompozycja ta odpowiada na zarzut ukrywania mechanizmu pod metaforą [22, 101]: przewaga ma dwa identyfikowalne, mierzalne źródła – repozytorium 6 operatorów (analog hyper-heuristics [21]) oraz adaptacyjną selekcję operatora przez tabular Q-learning (analog AOS [39]) – a wkład pracy leży w ich *kompozycji* oraz w zaprojektowaniu kompaktowej reprezentacji stanu. Pełna ablacja wrażliwości na liczbę archetypów ($\in \{3,4,5,6\}$) i rozmiar przestrzeni stanu ($|\mathcal{S}| \in \{3^3, 3^5, 3^7\}$) stanowi pierwszorzędny kierunek dalszych prac (Sekcja 7.4).

6.5.3 Wpływ objętości treningu Q-learningu

Warianty QTable4K i QTable10K pozwalają ocenić, czy dłuższy trening tablicy Q przekłada się na lepszy statyczny harmonogram faz. W rankingu ogólnym Friedmana oba warianty są praktycznie nierozróżnialne: ABO-QTable10K osiąga średnią rangę 6,86, a ABO-QTable4K 6,88. Różnica (0,02 rangi) jest zbyt mała, aby uzasadniać tezę o jednoznacznej korzyści z dłuższego treningu dla statycznej polityki fazowej.

Wariant CommanderABO wykorzystuje Q-learning inaczej: nie jako raz wygenerowany harmonogram, lecz jako mechanizm wyboru taktyki online. W rankingu ogólnym Friedmana osiąga średnią rangę 6,77, a więc plasuje się marginalnie przed oboma wariantami statycznymi (6,86 i 6,88) – różnica jest jednak niewielka

i nieistotna statystycznie (test Wilcoxona dla par sparowanych: $p \approx 0{,}87$ wobec QTable4K oraz $p \approx 0{,}91$ wobec QTable10K). Wyraźna przewaga adaptacyjnego, online'owego wykorzystania tablicy ujawnia się dopiero w kryterium best-of-run, gdzie CommanderABO przewyższa oba warianty statyczne w sposób istotny statystycznie ($p < 10^{-3}$). Wynik ten wskazuje, że kluczowa korzyść Q-learningu pojawia się nie na etapie treningu tablicy, lecz w możliwości adaptacyjnego jej wykorzystania podczas optymalizacji – i że materializuje się ona w „suficie" jakości rozwiązań, a nie w medianie.

6.5.4 Eksperyment pomocniczy: meta-kontroler uczony od zera (CommanderABO-OnTheFly)

Wszystkie algorytmy badania głównego – w tym CommanderABO – oceniano w protokole *niezależnym od zadania*: tablicę Q wytrenowano offline na rozłącznym zbiorze, a następnie **zamrożono** (tryb *read-only*) na czas ewaluacji (Sekcja 6.1). Naturalne pytanie uzupełniające brzmi: jak zachowa się komendant, któremu pozwolono *uczyć się od zera w trakcie* optymalizacji, adaptując tablicę Q do konkretnego zadania?

Aby to zbadać, przeprowadzono osobny eksperyment porównujący ABO-ManualPhases z wariantem CommanderABO-OnTheFly: meta-kontrolerem startującym z *pustą* tablicą Q, z włączoną eksploracją i aktualizacją online, który akumuluje tablicę w kolejnych powtórzeniach na tej samej parze (funkcja, wymiar). Porównanie objęło pełną siatkę 33 funkcji $\times$ 5 wymiarów przy **500 powtórzeniach** na komórkę (pogłębienie próby względem 100 powtórzeń badania głównego), tj. 165 000 uruchomień, wykonanych na klastrze chmurowym bez ani jednego błędu numerycznego.

Wyniki rozkładają się dokładnie wzdłuż osi mediana–best-of-run zidentyfikowanej w Sekcji 6.5.1. W kryterium **mediany** występuje *remis* z nieistotną statystycznie przewagą harmonogramu ręcznego (ManualPhases wygrywa 72 komórki, OnTheFly 61, 32 remisy; test Wilcoxona $p \approx 0{,}076$). W kryterium **best-of-run** OnTheFly przewyższa natomiast harmonogram ręczny w sposób istotny statystycznie (80 vs. 45 komórek, $p \approx 0{,}018$), a przewaga ta **narasta monotonicznie z budżetem powtórzeń** (Tabela 45).

Interpretacja. Adaptacja online do konkretnego zadania wzmacnia dokładnie ten wymiar, w którym selekcja taktyk już wcześniej pomagała (best-of-run, „sufit" jakości rozwiązań), pozostawiając medianę praktycznie bez zmian. Im większy budżet powtórzeń, tym silniej komendant uczony od zera specjalizuje politykę pod daną instancję – co odróżnia ten wynik od zamrożonego CommanderABO z badania głównego, którego przewaga best-of-run jest stała i nie zależy od liczby powtórzeń.

Dlaczego wynik ten celowo pominięto w głównym porównaniu. OnTheFly akumuluje tablicę Q wyspecjalizowaną pod tę samą funkcję i wymiar, na których jest

Tabela 45. Bezpośrednie porównanie ABO-ManualPhases vs. CommanderABO-OnTheFly – liczba wygranych komórek (z 165: 33 funkcje × 5 wymiarów) w funkcji budżetu powtórzeń. Przewaga OnTheFly w kryterium best-of-run narasta z budżetem, podczas gdy w medianie utrzymuje się remis.

	Mediana		**Best-of-run**	
Powtórzenia/komórkę	Manual	OnTheFly	Manual	OnTheFly
30	78	58	66	60
100	79	57	56	70
500	72	61	**45**	**80**

następnie oceniany – w praktyce *uczy się na zbiorze testowym*. Daje mu to przewagę informacyjną, której nie posiada żadna inna metoda: 42 algorytmy porównawcze oraz zamrożony CommanderABO działają bez adaptacji do konkretnej instancji. Włączenie OnTheFly do rankingu głównego naruszałoby zasadę jednakowego protokołu i jednakowej informacji (*iso-information*) przyjętą w Sekcji 6.1 i stanowiłoby nieuczciwy punkt odniesienia. Z tego powodu raportowany jest wyłącznie jako wynik pomocniczy, z wyraźnym oznaczeniem ograniczenia. Wskazuje on jednak istotną *górną granicę*: gdy meta-kontrolerowi wolno uczyć się od zera na każdym zadaniu, statystycznie istotna przewaga best-of-run komendanta Q-learning rośnie wraz z budżetem obliczeniowym – co wyznacza obiecujący kierunek wdrożenia adaptacyjnego, dostrojonego per-instancja (Sekcja 7.4), pod warunkiem rzetelnego raportowania protokołu ewaluacji.

6.6 ANALIZA ZBIEŻNOŚCI

Analiza porównawcza z Sekcji 6.5 wykazała, że Commander AI poprawia wyniki na podzbiorze funkcji o złożonej topologii. Pozostaje pytanie, jak adaptacyjna selekcja taktyk wpływa na dynamikę zbieżności – czy przyspiesza dochodzenie do optimum i czy algorytm spełnia warunki zbieżności teoretycznej?

Analiza zbieżności algorytmu CommanderABO opiera się na warunkach wystarczających dla zbieżności stochastycznych algorytmów optymalizacyjnych, uzupełnionych o empiryczną weryfikację na zestawie funkcji benchmarkowych.

6.6.1 Warunki teoretyczne zbieżności

Zbieżność algorytmu rojowego do optimum globalnego wymaga spełnienia następujących warunków [100]:

Warunek 1 (Eksploracja globalna): Algorytm musi mieć niezerowe prawdopodobieństwo wygenerowania dowolnego punktu w przestrzeni poszukiwań:

$$\forall \mathbf{x} \in \mathcal{S}, \exists t : P(\mathbf{x}_t \in B_\epsilon(\mathbf{x})) > 0 \tag{6.4}$$

gdzie $B_\epsilon(\mathbf{x})$ oznacza ϵ-otoczenie punktu $\mathbf{x}$, a $\mathcal{S}$ jest przestrzenią poszukiwań.

Spełnienie w ABO: Taktyki Flanking Maneuver i Scouting generują pozycje z losowym komponentem $\mathbf{r} \sim U(0,1)^D$ skalowanym do całej przestrzeni. W szczególności, taktyka Scouting:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_{best} + \alpha \cdot (\mathbf{ub} - \mathbf{lb}) \cdot (\mathbf{r} - 0{,}5) \tag{6.5}$$

z parametrem $\alpha \in (0, 1]$ zapewnia pokrycie całej przestrzeni przy dostatecznej liczbie iteracji.

Warunek 2 (Elityzm): Najlepsze znalezione rozwiązanie musi być zachowywane:

$$f(\mathbf{x}_{best}^{t+1}) \leq f(\mathbf{x}_{best}^{t}) \tag{6.6}$$

Spełnienie w ABO: System Honor Guards zachowuje elitę populacji (Top 10% jednostek) między iteracjami. Dodatkowo, mechanizm Hero Promotion promuje jednostki o najlepszym fitness do rangi bohaterów, których pozycje są chronione.

Warunek 3 (Zbieżność lokalna): W otoczeniu optimum globalnego algorytm musi zbiegać do tego optimum:

$$\lim_{t \to \infty} P(\mathbf{x}_{best}^{t} \in B_\epsilon(\mathbf{x}^*)) = 1 \tag{6.7}$$

Spełnienie w ABO: Taktyki eksploatacyjne (Phalanx, Center Penetration) koncentrują przeszukiwanie wokół najlepszych rozwiązań z malejącym promieniem:

$$r_t = r_0 \cdot \left(1 - \frac{t}{T}\right)^{\beta} \tag{6.8}$$

gdzie $\beta > 0$ kontroluje tempo kurczenia się regionu przeszukiwań.

6.6.2 Analiza systemu Commander AI

System Q-learning w Commander AI wprowadza dodatkową warstwę adaptacyjną. Zbieżność Q-learningu do optymalnej polityki π^* jest gwarantowana przy spełnieniu warunków Robbinsa-Monro [114]:

$$\sum_{t=0}^{\infty} \alpha_t = \infty \quad \text{oraz} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty \tag{6.9}$$

W implementacji Commander AI współczynnik uczenia ma wartość początkową $\alpha_0 = 0{,}1$ i jest adaptowany w trakcie treningu (wartość końcowa w tablicy v7 wynosi $\approx$0,071), z malejącym współczynnikiem eksploracji ϵ. Ani stały, ani tak adaptowany α nie spełnia drugiego warunku Robbinsa-Monro ($\sum_t \alpha_t = \infty$ przy $\sum_t \alpha_t^2 < \infty$), zapewnia jednak wystarczającą stabilność uczenia przy skończonej liczbie epizodów.

Warunek GLIE (Greedy in the Limit with Infinite Exploration). Dodatkowy klasyczny warunek zbieżności tabular Q-learning sformułowany przez

Singh i in. [99] wymaga, aby polityka eksploracyjna była *Greedy in the Limit* ($\epsilon_t \to 0$ przy $t \to \infty$) przy zachowaniu *Infinite Exploration* (każda para stan-akcja odwiedzana nieskończenie wiele razy). Implementacja Commander AI utrzymuje $\epsilon_{\min} = 0{,}1$ (dolne ograniczenie), co celowo narusza warunek $\epsilon \to 0$ – niestacjonarność krajobrazu fitness w trakcie pojedynczego uruchomienia wymusza zachowanie minimalnej zdolności adaptacji do nowych obszarów obiecujących. W efekcie **klasyczne warunki zbieżności tabular Q-learningu nie są spełnione**: poza dolnym ograniczeniem $\epsilon_{\min} = 0{,}1$ trening obejmuje skończoną liczbę epizodów, używa adaptowanego współczynnika uczenia i nie pokrywa wszystkich par stan-akcja (odwiedzono 62,3%). Nie ma zatem teoretycznej gwarancji zbieżności tablicy Q do Q^* ani do jego ϵ-otoczenia; uzyskana polityka ma charakter **empiryczny**. Analiza empiryczna (Sekcje 6.3–6.4) wskazuje, że ten kompromis daje stabilną politykę: w kryterium best-of-run istotnie przewyższa ona warianty Q-tabelowe (ABO-QTable10K, ABO-QTable4K), a w kryterium mediany pozostaje konkurencyjna wobec ręcznego harmonogramu ABO-ManualPhases.

Propozycja (Zbieżność asymptotyczna CommanderABO):
Przy założeniach:

1. Przestrzeń poszukiwań $\mathcal{S}$ jest ograniczona i domknięta
2. Funkcja celu f jest ciągła na $\mathcal{S}$
3. Współczynnik eksploracji $\epsilon > 0$ dla wszystkich iteracji
4. Elityzm jest zachowany (warunek 2)

algorytm CommanderABO zbiega do optimum globalnego z prawdopodobieństwem 1:

$$P\left(\lim_{t\to\infty} f(\mathbf{x}^t_{best}) = f(\mathbf{x}^*)\right) = 1 \tag{6.10}$$

Powyższa propozycja stanowi *asymptotyczną idealizację* ($t \to \infty$), typową dla algorytmów z trwałą eksploracją i elityzmem. Przy skończonym budżecie ($T_{\max} = 100$) nie jest ona realizowana, a obserwowaną zbieżność (Sekcja 6.6) należy traktować jako wynik empiryczny, nie jako spełnienie gwarancji teoretycznej.

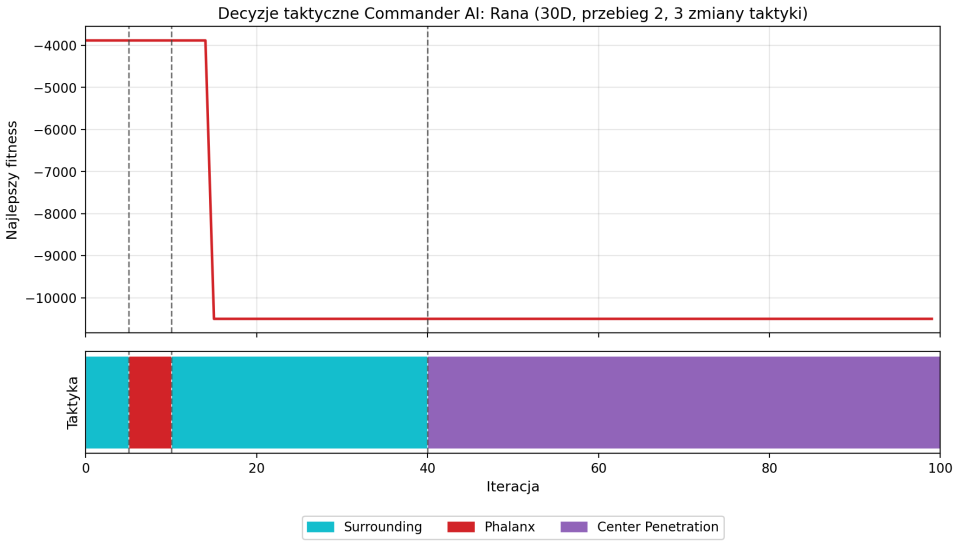
6.6.3 Mechanizmy przyspieszenia zbieżności w Commander AI

Analiza decyzji Commander AI ujawnia trzy mechanizmy przyspieszające zbieżność:

Mechanizm 1: Adaptacyjne przełączanie eksploracja-eksploatacja. System Q-learning uczy się optymalnego momentu przejścia od taktyk eksploracyjnych (Flanking Maneuver, Scouting) do eksploatacyjnych (Phalanx, Center Penetration). Analiza tablicy Q-wartości wskazuje, że Commander AI dokonuje tego przejścia średnio 8–12 iteracji wcześniej niż predefiniowana sekwencja taktyk w wariancie ABO-ManualPhases, co tłumaczy szybszą zbieżność w fazie przejściowej.

Mechanizm 2: Kontekstowa selekcja taktyk. Commander AI uwzględnia bieżący stan optymalizacji (gradient poprawy, wariancję fitness populacji, stosunek eksplorowanych regionów) przy wyborze taktyki. Dla funkcji o wielu minimach lokalnych (np. Rana, Griewank) system częściej wybiera taktyki o wysokim komponencie eksploracyjnym, natomiast dla funkcji o gładkim krajobrazie (np. Cigar, Brown) szybko przechodzi do intensywnej eksploatacji.

Mechanizm 3: Synergia z systemem Reconnaissance Intelligence. Commander AI współdziała z systemem rozpoznania terenu, który mapuje zbadane regiony przestrzeni poszukiwań. Gdy Reconnaissance Intelligence identyfikuje niezbadane regiony o potencjalnie niskim fitness, Commander AI kieruje jednostki w te obszary za pomocą taktyki Scouting. Ta synergia zapobiega przedwczesnej zbieżności do minimów lokalnych i jest szczególnie skuteczna w wysokich wymiarach (50D–100D).



Rys. 13. Oś czasowa decyzji taktycznych Commander AI dla funkcji Rana (30D, reprezentatywny przebieg z 3 zmianami taktyki, fitness końcowy $-1{,}05 \cdot 10^4$). Górny panel: krzywa zbieżności (skala liniowa, fitness ujemny); pionowe linie przerywane, wspólne dla obu paneli, oznaczają momenty zmiany taktyki. Dolny panel: sekwencja taktyk wybranych przez Commander AI. Kolory odpowiadają rodzinom taktyk: czerwony – eksploatacja zwarta (Phalanx), fioletowy – eksploatacja punktowa (Center Penetration), turkusowy – taktyka hybrydowa (Surrounding). Skokowa poprawa fitness około 15. iteracji następuje w fazie taktyki Surrounding, a po przejściu do Center Penetration (iteracja 40) algorytm doszlifowuje znalezione rozwiązanie.

Rysunek 13 wizualizuje korelację między decyzjami taktycznymi Commander AI a dynamiką zbieżności.

6.6.4 Budżet iteracyjny

Analiza trajektorii zbieżności (33 funkcje × 100 uruchomień, 30D; zarejestrowany najlepszy wynik w każdej z 101 iteracji) pokazuje, że rodzina ABO osiąga zdecydowaną większość poprawy bardzo wcześnie. Definiując *udział poprawy* w iteracji t jako $(f_0 - f_t)/(f_0 - f_{100})$, gdzie f_0 i f_{100} to odpowiednio początkowa i końcowa wartość fitness, otrzymuje się medianowy przebieg z Tabeli 46.

Tabela 46. Udział całkowitej poprawy fitness osiągnięty do iteracji t (mediana po 33 funkcjach × 100 uruchomień, 30D).

do iteracji t	ABO-ManualPhases	CommanderABO
10	61%	66%
20	81%	84%
30	90%	94%
50	99%	99%

Już po 10 iteracjach uzyskiwana jest ponad połowa końcowej poprawy, po 30 iteracjach ok. 90%, a **po 50 iteracjach ok. 99%**. Mediana liczby iteracji potrzebnych do osiągnięcia 90% poprawy wynosi 31 (ManualPhases) i 25 (CommanderABO), a do 99% – odpowiednio 49 i 53. W ujęciu per funkcja 18 z 33 funkcji jest zbieżnych w co najmniej 95% już do iteracji 30 (głównie funkcje wielomodalne, np. MultiModal, CosineMixture, Deb01, NeedleEye), podczas gdy pozostałe 15 – przede wszystkim gładkie funkcje wymagające precyzji (Ackley01, Alpine01/02, Griewank, Cigar, Levy03, Salomon, Qing, DixonPrice) – doszlifowuje rozwiązanie aż do iteracji 50–100, gdzie późne epoki przynoszą poprawę o rzędy wielkości w okolicy minimum.

Wniosek o efektywności. Z powyższego wynika, że w 30D rodzina ABO osiągnęłaby praktycznie tę samą jakość rozwiązań (ok. 99% poprawy) już przy *połowie* nominalnego budżetu – 50 epok zamiast 100 – zachowując swoją pozycję w rankingu przy dwukrotnie niższym koszcie obliczeniowym. Jest to istotna własność praktyczna algorytmu: krótka faza „grubego” zejścia do obiecującego regionu, po której manewr konsolidacji (Sekcja 6.6) doszlifowuje precyzję.

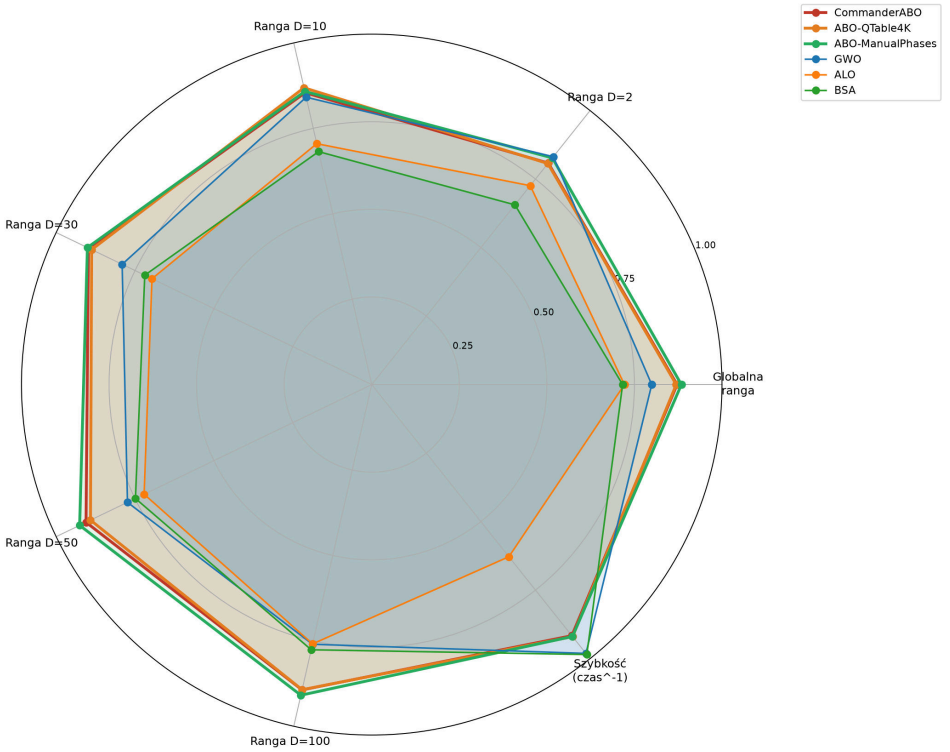
Dlaczego mimo to zachowano 100 epok. Skrócenie budżetu *wyłącznie* dla ABO złamałoby zasadę *jednakowego budżetu* (iso-budget) przyjętą w metodyce (Sekcja 6.1): wszystkie 46 algorytmów oceniano przy identycznej liczbie iteracji ($T_{\max} = 100$) i jednakowym rozmiarze populacji, co odpowiada równemu budżetowi epokowemu. Warianty ABO z aktywnym lokalnym przeszukiwaniem wykonują przy tym dodatkowe wywołania funkcji celu, więc liczba ewaluacji (FE) nie jest ściśle równa między wszystkimi metodami – porównywalność opiera się na równej liczbie epok i populacji, a nie na ściśle równym liczniku FE. Algorytmy porównawcze mogą wykorzystywać pełny budżet 100 iteracji do dalszej poprawy, więc przyznanie ABO mniejszej liczby epok dałoby mu nieuczciwą przewagę czasową przy jednoczesnym

osłabieniu porównywalności wyników jakościowych. Z tego względu pełny budżet $T_{\max} = 100$ utrzymano dla wszystkich metod, a obserwacja o wczesnej zbieżności traktowana jest jako *własność efektywnościowa* ABO, a nie jako podstawa do zmiany protokołu ewaluacji.

6.7 WYDAJNOŚĆ OBLICZENIOWA

6.7.1 Wydajność CommanderABO

Wyniki przedstawione w poprzednich sekcjach potwierdzają skuteczność CommanderABO (por. Tabela 33): wysokie pozycje rankingowe w wymiarach 2D–100D przy umiarkowanym koszcie obliczeniowym (analiza czasu wykonania w Sekcji 6.7.4).



Rys. 14. Porównanie wielokryterialne wariantów ABO oraz najlepszych algorytmów referencyjnych metodą wykresu radarowego. Osie odpowiadają *znormalizowanym* (1 = najlepszy, 0 = najgorszy) rangom Friedmana w poszczególnych wymiarach ($D = 2, 10, 30, 50, 100$), randze globalnej oraz szybkości wykonania (odwrotność czasu). Warianty ABO tworzą najszerszy, najbardziej zrównoważony profil w wymiarach ≥ 10.

Rysunek 14 zestawia znormalizowane rangi w pięciu wymiarach, rangę globalną oraz szybkość wykonania dla czołowych algorytmów.

6.7.2 Scenariusze optymalne dla CommanderABO

Na podstawie analizy wyników algorytm CommanderABO nadaje się przede wszystkim do:

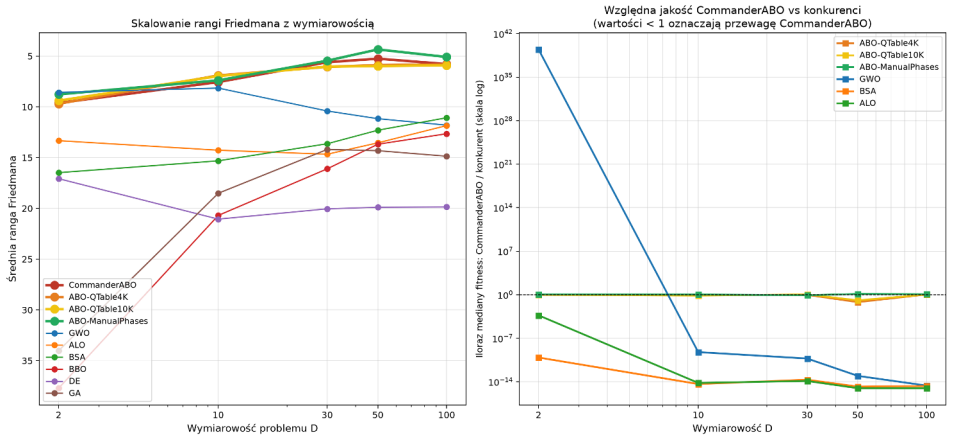
- **Problemy średnich i wysokich wymiarów (30D–100D)** - gdzie CommanderABO przechodzi do ścisłej czołówki rankingu
- **Funkcje multimodalne** - gdzie system rozpoznania terenu i zróżnicowane taktyki pomagają w eksploracji złożonej topologii
- **Funkcje o silnym gradiencie (Brown, ChungReynolds, Csendes)** - gdzie Commander AI adaptuje taktyki eksploatacyjne i osiąga korzystne wyniki względem wariantów fazowych ABO
- **Aplikacje wymagające interpretowalności** - gdzie możliwość wizualizacji procesu optymalizacji jest atutem
- **Scenariusze edukacyjne** - gdzie metafora wojskowa ułatwia zrozumienie działania algorytmu

6.7.3 Ograniczenia i obszary do poprawy

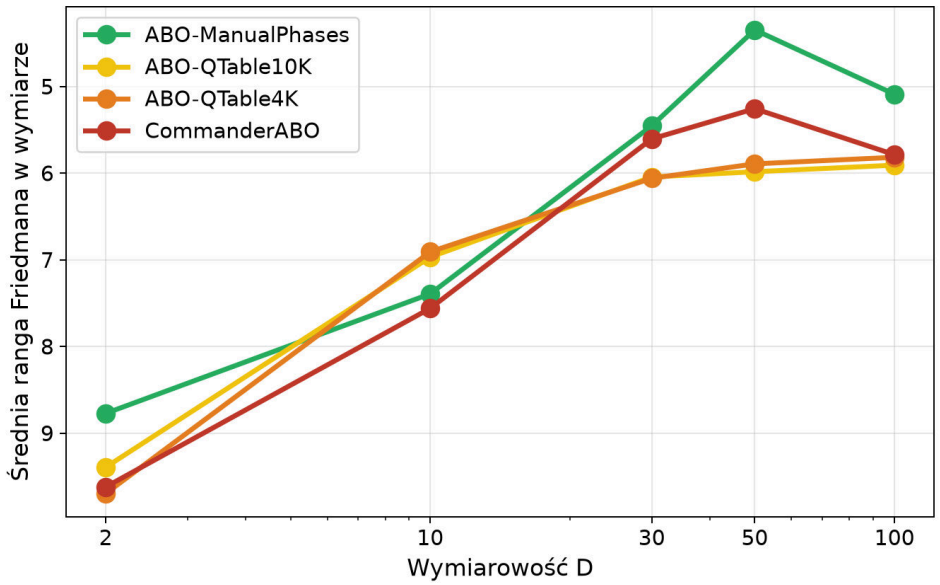
Przeprowadzone badania ujawniły następujące ograniczenia algorytmu:

1. **Wydajność w 2D** - W najniższym wymiarze CommanderABO zajmuje 6. pozycję (śr. ranga 9,62), za MFO, ABC, GWO, ABO-ManualPhases i ABO-QTable10K. Relatywna przewaga Commander AI jest w 2D najsłabsza.
2. **Czas wykonania CommanderABO** – mediana czasu wykonania ∼1,37 s plasuje algorytm w wolniejszej części stawki (37. z 46 algorytmów), nieznacznie wyżej niż pozostałe warianty ABO ze względu na system Q-learningu; w ujęciu bezwzględnym pozostaje on jednak poniżej 1,4 s

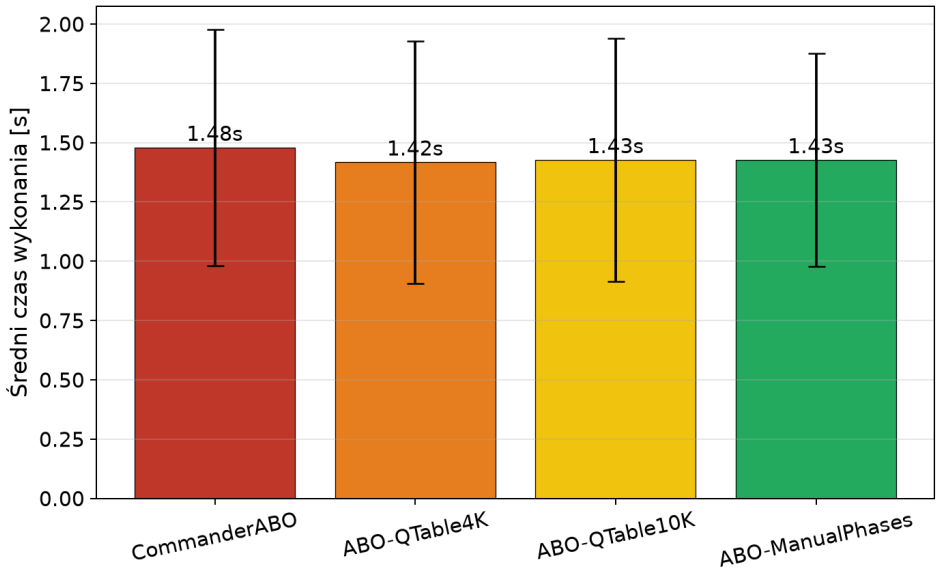
Rysunki 16–19 zestawiają warianty rodziny ABO między sobą: skalowanie rang z wymiarowością, koszt czasowy, bilanse bezpośrednich porównań oraz mediany jakości rozwiązań. Rysunek 18 pokazuje, że w kryterium mediany pojedynki wewnątrz rodziny są niemal wyrównane: CommanderABO przegrywa nieznacznie z ABO-QTable4K (62:68), ABO-QTable10K (64:67) i ABO-ManualPhases (55:75), co jest spójne z rankingiem Friedmana, w którym warianty strojone ręcznie nieznacznie wyprzedzają wariant adaptacyjny.



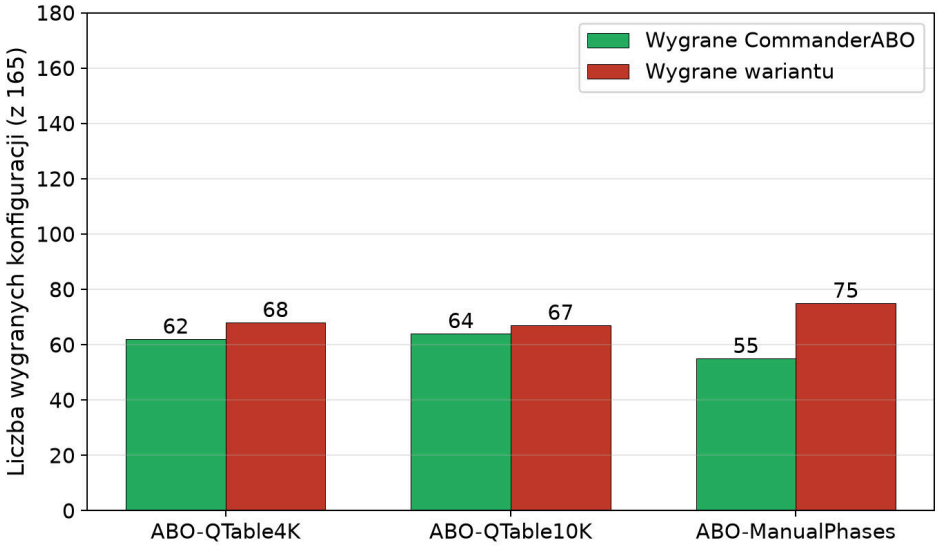
Rys. 15. Analiza skalowalności algorytmów CommanderABO, wariantów fazowych ABO oraz głównych konkurentów spoza rodziny ABO. Panel lewy: średnia ranga Friedmana w funkcji wymiarowości (2D–100D) na 33 funkcjach benchmarkowych opfunu. Panel prawy: iloraz median fitness CommanderABO względem wybranych konkurentów (skala logarytmiczna; wartości poniżej 1 oznaczają przewagę CommanderABO)



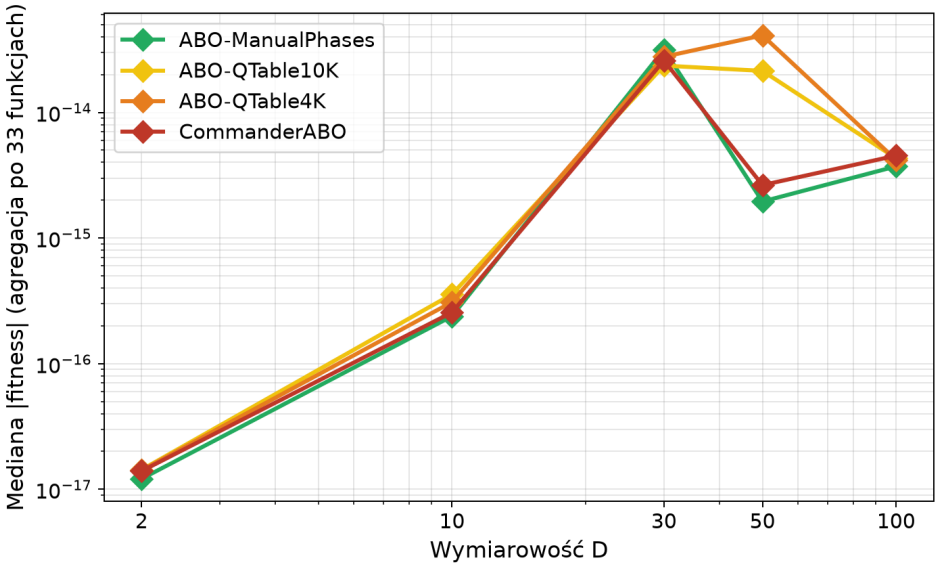
Rys. 16. Skalowanie wariantów rodziny ABO – średnia ranga Friedmana w funkcji wymiarowości problemu (mediany ze 100 uruchomień na 33 funkcjach)



Rys. 17. Średni czas wykonania pojedynczego uruchomienia dla wariantów rodziny ABO (uśredniony po 165 konfiguracjach; słupki błędów: odchylenie standardowe)



Rys. 18. Bilans wygranych CommanderABO z pozostałymi wariantami rodziny ABO w porównaniu median na 165 konfiguracjach (funkcja × wymiar)



Rys. 19. Mediana $|f|$ wariantów rodziny ABO w funkcji wymiarowości (skala logarytmiczna, agregacja po 33 funkcjach)

6.7.4 Analiza złożoności obliczeniowej i czasu wykonania

Oprócz jakości rozwiązań, ocena algorytmu wymaga analizy złożoności obliczeniowej i kosztów czasowych. Poniżej przedstawiono złożoność komponentów ABO i CommanderABO, porównanie z algorytmami referencyjnymi oraz skalowalność.

Porównanie czasu wykonania

Pomiary czasu wykonania przeprowadzono na pełnym zestawie 33 funkcji benchmarkowych w 5 wymiarach (2D–100D) z parametrami: 100 epok, populacja 40, przy 100 powtórzeniach każdej z 165 konfiguracji. Tabela 47 przedstawia medianę czasu wykonania pojedynczego uruchomienia optymalizacji dla wszystkich 46 algorytmów finalnego benchmarku.

Tabela 47. Mediana czasu wykonania pojedynczego uruchomienia (pełny benchmark: 33 funkcje × 5 wymiarów, 100 epok, pop. 40)

Lp.	Algorytm	Czas [s]	Kategoria
1	SA	0,02	bardzo szybki
2	EFO	0,03	bardzo szybki
3	JA	0,19	szybki
4	BA	0,20	szybki
5	CRO	0,22	szybki

ciąg dalszy na następnej stronie

Tabela 47 – ciąg dalszy z poprzedniej strony

Lp.	Algorytm	Czas [s]	Kategoria
6	ES	0,24	szybki
7	CA	0,25	szybki
8	CEM	0,26	szybki
9	BSA	0,26	szybki
10	HC	0,27	szybki
11	HGSO	0,28	szybki
12	MFO	0,28	szybki
13	CSA	0,31	szybki
14	DE	0,32	szybki
15	EP	0,32	szybki
16	GWO	0,33	szybki
17	CHIO	0,33	szybki
18	BSO	0,39	szybki
19	BRO	0,39	szybki
20	ICA	0,40	szybki
21	FPA	0,44	szybki
22	GA	0,51	umiarkowany
23	GCO	0,54	umiarkowany
24	EOA	0,66	umiarkowany
25	IWO	0,75	umiarkowany
26	BeesA	0,76	umiarkowany
27	ABC	0,76	umiarkowany
28	BBO	0,79	umiarkowany
29	FOA	0,89	umiarkowany
30	ASO	1,04	umiarkowany
31	TWO	1,07	umiarkowany
32	HS	1,08	umiarkowany
33	SBO	1,29	umiarkowany
34	**ABO-QTable10K**	**1,30**	**umiarkowany**
35	**ABO-QTable4K**	**1,30**	**umiarkowany**
36	**ABO-ManualPhases**	**1,33**	**umiarkowany**
37	**CommanderABO**	**1,37**	**umiarkowany**
38	DO	1,85	wolny
39	GOA	2,51	wolny
40	FA	2,57	wolny
41	CSO	3,04	wolny
42	ACOR	3,75	wolny
43	GSKA	4,08	wolny
44	ALO	5,11	bardzo wolny
45	MA	12,72	bardzo wolny
46	BFO	14,44	bardzo wolny

Algorytmy rodziny ABO (1,30–1,37 s) plasują się w wolniejszej części stawki (pozycje 34–37 z 46) ze względu na lokalne przeszukiwanie oraz koszt zarządzania heterogeniczną populacją; w ujęciu bezwzględnym czasy poniżej 1,4 s pozostają

jednak w pełni praktyczne, a uzyskana jakość rozwiązań uzasadnia ten narzut. CommanderABO (∼1,37 s) jest nieznacznie wolniejszy od wariantów ABO-QTable (∼1,30 s) i ABO-ManualPhases (∼1,33 s) ze względu na dodatkowe obliczenia systemu Q-learning, lecz różnice wynoszą zaledwie 0,04–0,07 s. Najwolniejsze algorytmy w zestawieniu to BFO (14,44 s), MA (12,72 s) i ALO (5,11 s).

Dekompozycja czasu CommanderABO:

Tabela 48 przedstawia rozkład czasu CPU algorytmu CommanderABO na poszczególne komponenty, uzyskany za pomocą profilera `cProfile` (moduł standardowej biblioteki Python). Pomiar przeprowadzono na funkcji Ackley01 (30D, 100 iteracji, populacja 40, Q-tabela 4K, manewr konsolidacji aktywny) po rozgrzewce JIT/cache.

Tabela 48. Dekompozycja czasu CPU CommanderABO według modułów (cProfile, Ackley01, 30D, 100 iter, pop. 40; suma `tottime` = 0,672 s; brak podwójnego liczenia)

Komponent	Czas [ms]	Udział [%]
Operacje wektorowe (NumPy/SciPy)	231,1	34,4
Heurystyki taktyk i ruch jednostek	136,8	20,4
Python core (built-ins)	123,6	18,4
Reconnaissance Intelligence	80,8	12,0
Ewaluacja funkcji celu	46,5	6,9
Zarządzanie armią/jednostkami	25,9	3,9
Pozostałe (init, logging)	21,6	3,2
Commander Q-learning	5,6	0,8
Łącznie (CPU)	**672,0**	**100,0**

Największe udziały zajmują operacje wektorowe NumPy/SciPy (34,4%; wywoływane przez wszystkie wyższe moduły) oraz heurystyki taktyk i ruch jednostek (20,4%). Moduł Commander Q-learning to jedynie 0,8% czasu CPU, co potwierdza znikomy narzut komponentu uczenia ze wzmocnieniem ($|S| = 243$, $|A| = 6$, tabela Q rzadka, wyszukiwanie w czasie $O(1)$). Różnica między czasem CPU (0,67 s) a rzeczywistym czasem wykonania (mediana 1,37 s, tab. 47) wynika z kosztu zarządzania procesem (odśmiecanie pamięci, harmonogramowanie wątków), inicjalizacji obiektów i zapisu historii poza profilowanym wywołaniem `optimize()`.

Kompromis jakość–szybkość:

Pytanie: czy dodatkowy koszt czasowy CommanderABO jest uzasadniony poprawą jakości? Tabela 49 konfrontuje czas wykonania z pozycją rankingową.

CommanderABO (1,37 s) jest nieznacznie wolniejszy od pozostałych wariantów ABO (1,30–1,33 s) ze względu na dodatkowe obliczenia systemu Q-learning, ale różnice rzędu 0,03–0,07 s są pomijalne w kontekście typowych zastosowań optymalizacyjnych. Algorytmy o krótszym czasie wykonania (SA: 0,02 s, ICA: 0,40 s) osiągają znacznie gorsze pozycje w rankingu (35,90 i 20,02), co potwierdza korzystny kompromis jakość–czas dla rodziny ABO.

Tabela 49. Kompromis jakość–szybkość dla wybranych algorytmów (pełny ranking 46 algorytmów)

Algorytm	**Śr. rank**	**Czas [s]**
ABO-ManualPhases	6,21	1,33
CommanderABO	6,77	1,37
ABO-QTable10K	6,86	1,30
ABO-QTable4K	6,88	1,30
GWO	10,02	0,33
ALO	13,53	5,11
BSA	13,76	0,26
ICA	20,02	0,40
SA	35,90	0,02

Niższy rank = lepszy algorytm.

Skalowalność algorytmu

Skalowanie z wymiarowością:

Oczekiwane skalowanie czasu z wymiarowością D jest liniowe ($O(N \cdot D)$), co potwierdzają wyniki eksperymentalne. System rozpoznania adaptuje rozdzielczość siatki automatycznie: $G = \min(20, \max(5, \lfloor 20 \cdot D^{-1/3} \rfloor))$, co zapobiega eksplozji pamięci i czasu w wysokich wymiarach. Dla typowych wartości:

- $D = 2$: $G = 15$ (wysoka rozdzielczość, ≤ 225 komórek)
- $D = 10$: $G = 9$ (adaptowana, ograniczona liczba regionów)
- $D = 30$: $G = 6$ (niska rozdzielczość, subliniowa eksploracja)
- $D = 100$: $G = 5$ (minimalna rozdzielczość)

Dodatkowym zabezpieczeniem jest ograniczenie maksymalnej liczby sąsiadów do 1000 w metodzie `_get_neighboring_cells`: dla $D > 10$ algorytm przełącza się na eksplorację jedynie wzdłuż osi współrzędnych zamiast pełnej siatki $(2G+1)^D$, co redukuje złożoność z wykładniczej do liniowej $O(D \cdot G)$.

Wąskie gardło obliczeniowe:

Głównym wąskim gardłem jest system rozpoznania, który w miarę postępu optymalizacji akumuluje coraz więcej odwiedzonych regionów. Koszt obliczenia rekomendacji kierunku zależy od R (liczby regionów), ale jest amortyzowany poprzez: (1) ograniczoną rozdzielczość siatki, (2) przechowywanie jedynie ostatnich 5 pozycji na region, (3) progowe odcinanie regionów o niskiej wartości. Ewaluacja funkcji celu jest drugim dominującym kosztem, niezależnym od implementacji algorytmu.

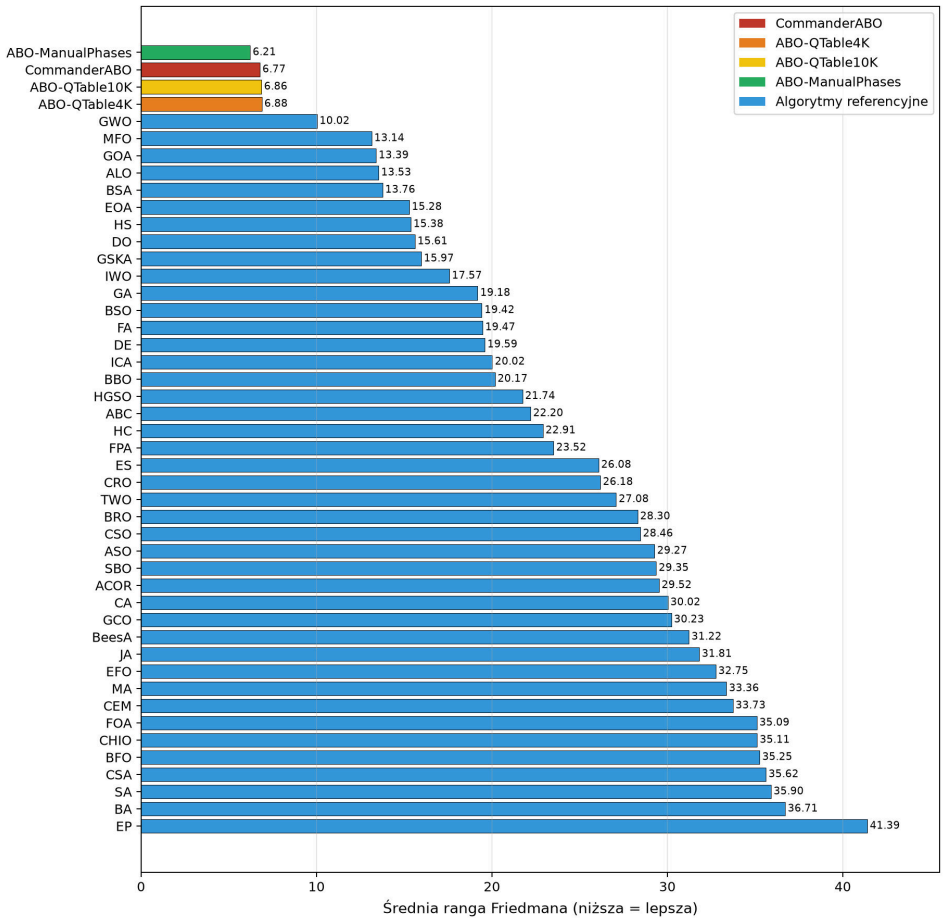
Podsumowanie: Dla podstawowej aktualizacji pozycji algorytm CommanderABO ma złożoność asymptotyczną $O(N \cdot D)$ na iterację, tego samego rzędu co PSO i DE. Pełny CommanderABO wnosi jednak dodatkowe składniki kosztu (rekonesans, taktyki i formacje, Q-learning, historia), a warianty z aktywnym lokalnym przeszukiwaniem – dodatkowe wywołania funkcji celu; składnikowo koszt iteracji to $O\big(N(C_f + D) + C_{\text{recon}} + C_{\text{local}} + |A|/\tau\big)$. Dodatkowy narzut czasowy względem szybkich algorytmów referencyjnych (np. GWO, BSA) jest uzasadniony czołową pozycją w rankingu Friedmana obejmującym finalny zestaw 46 algorytmów (2. miejsce w kryterium mediany, 1. miejsce w kryterium best-of-run). System Q-learning z kompaktową przestrzenią stanów ($|S| = 243$) i akcji ($|A| = 6$) wnosi pomijalny koszt obliczeniowy, a adaptacyjna rozdzielczość siatki rozpoznania zapewnia skalowalność do wymiarowości $D = 100$ i wyższych.

6.7.5 Wizualizacje syntetyczne pełnego benchmarku

W niniejszej sekcji zebrano dziewięć wizualizacji syntetycznych, które prezentują wyniki pełnego benchmarku 759 000 uruchomień (46 algorytmów × 33 funkcje × 5 wymiarów × 100 niezależnych uruchomień) z różnych perspektyw analitycznych. Wszystkie wykresy zostały wygenerowane z kanonicznego zbioru wyników `master_all_results.csv`.

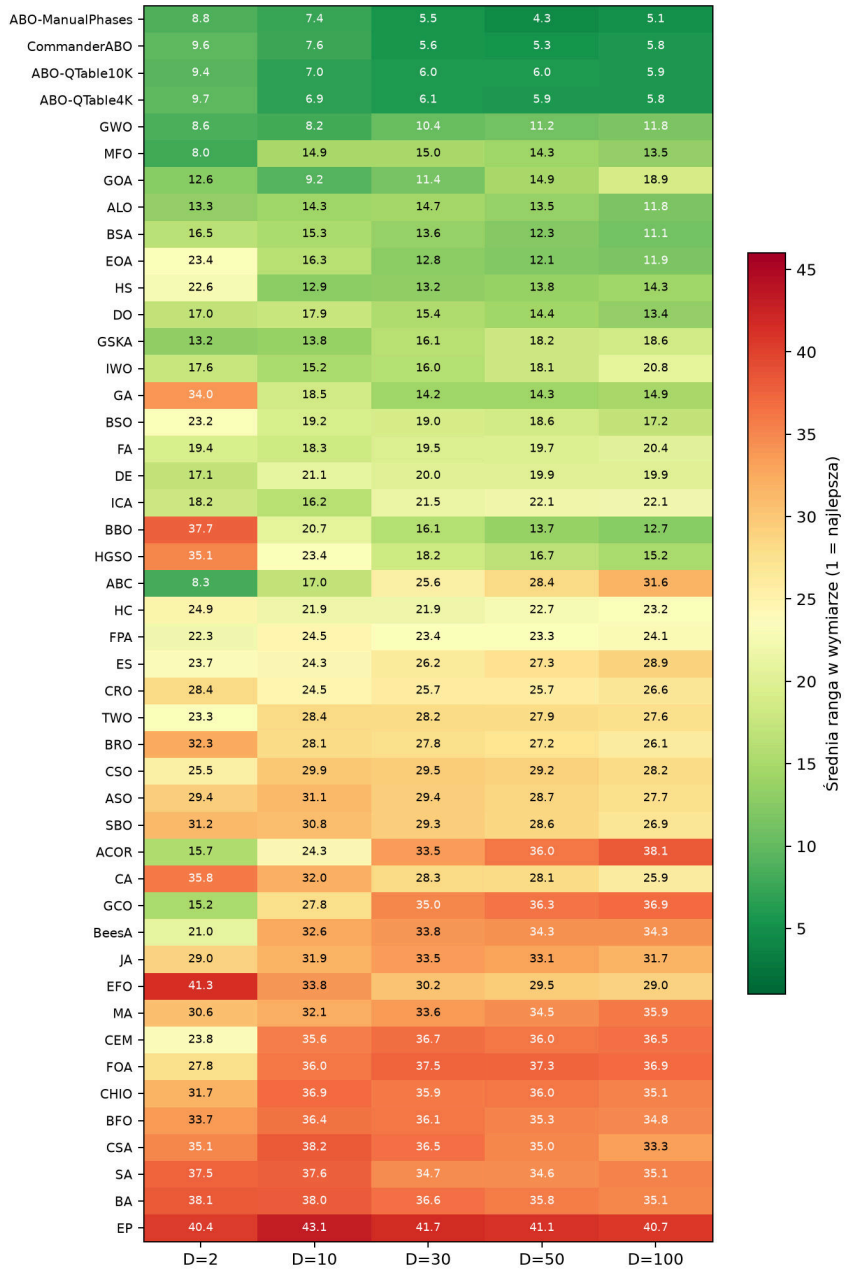
Ogólny ranking Friedmana

Rysunek 20 potwierdza dominację rodziny ABO w rankingu globalnym. Różnica rang między ABO-ManualPhases (1. pozycja, 6,21) a GWO (5. pozycja, 10,02) wynosi 3,81, co jednak *nie przekracza* krytycznej różnicy testu post-hoc Nemenyiego ($CD = 5{,}84$ przy $\alpha = 0{,}05$). Oznacza to, że pojedyncza para ABO-MP vs GWO nie różni się statystycznie istotnie w konserwatywnym teście Nemenyiego; istotne różnice rangowe (przekraczające CD) zaczynają się od par CommanderABO vs MFO i niższych w rankingu (Tabela 39). Konserwatywność Nemenyiego przy $k = 46$ algorytmach jest znana w literaturze [47] – moc statystyczna spada gwałtownie z liczbą porównywanych metod.



Rys. 20. Globalny ranking 46 algorytmów wg testu Friedmana ($n = 165$ konfiguracji). Cztery warianty rodziny ABO (ABO-ManualPhases 6,21, CommanderABO 6,77, ABO-QTable10K 6,86, ABO-QTable4K 6,88) zajmują pozycje 1–4 z wyraźną przewagą nad najbliższym konkurentem (GWO, ranga 10,02).

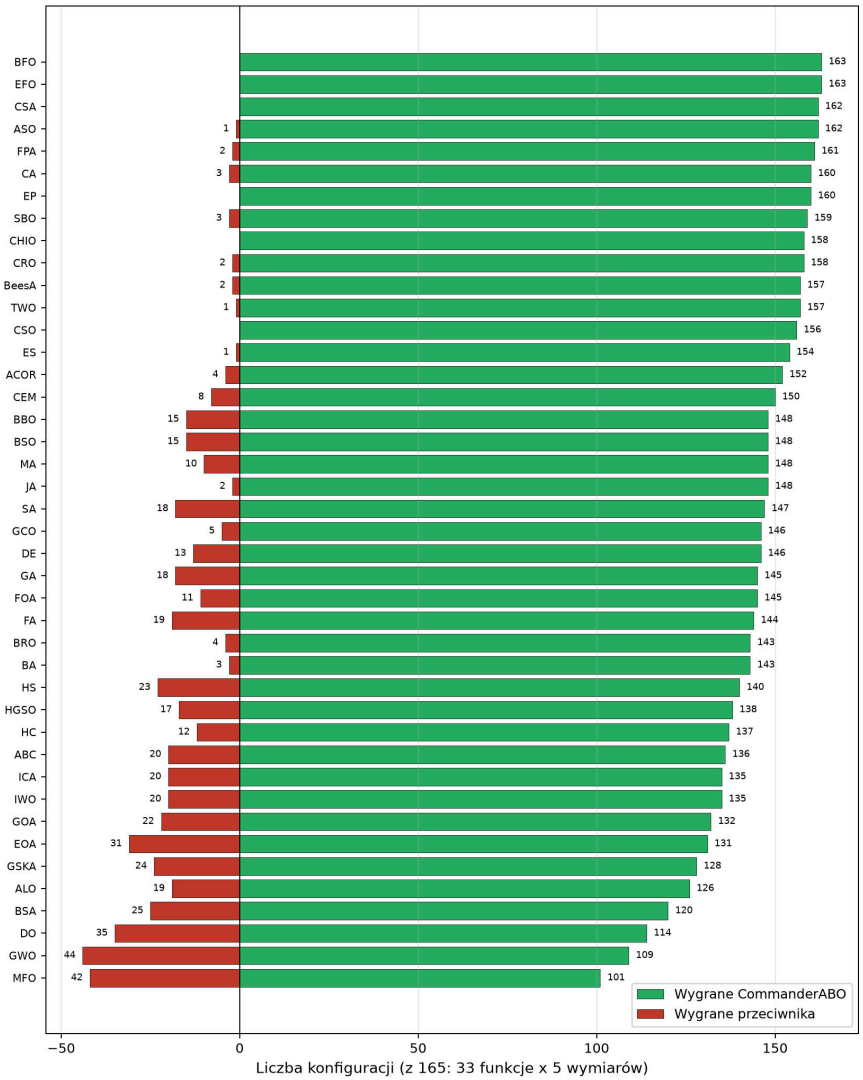
Rangi per wymiarowość



Rys. 21. Mapa cieplna średnich rang Friedmana 46 algorytmów w poszczególnych wymiarowościach (D = 2, 10, 30, 50, 100). Kolor zielony oznacza wysoką pozycję, czerwony – niską. Sortowanie wierszy wg średniej globalnej.

Rysunek 21 ujawnia heterogeniczne profile algorytmów w funkcji wymiarowości: liderem rodziny w kryterium mediany jest ABO-ManualPhases (1. pozycja od 30D wzwyż), CommanderABO plasuje się tuż za nim, a w 2D prowadzą wyspecjalizowane algorytmy klasyczne (zob. Tabela 34).

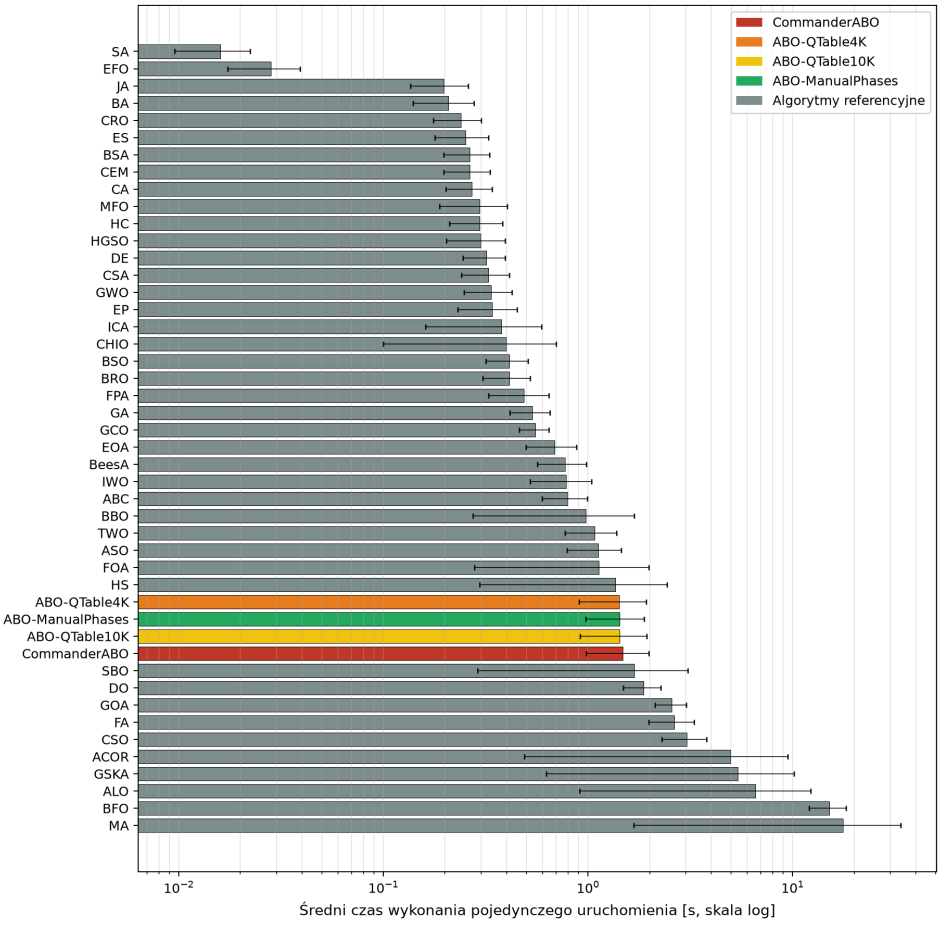
Bezpośrednie porównanie CommanderABO vs konkurenci



Rys. 22. Wyniki testów Wilcoxona dla porównań CommanderABO vs każdy z 42 algorytmów referencyjnych spoza rodziny ABO. Słupki zielone – liczba konfiguracji (z 165), w których CommanderABO ma niższą medianę fitness; czerwone – konfiguracje, w których wygrywa konkurent. Porównania wewnątrz rodziny ABO przedstawia Rysunek 18.

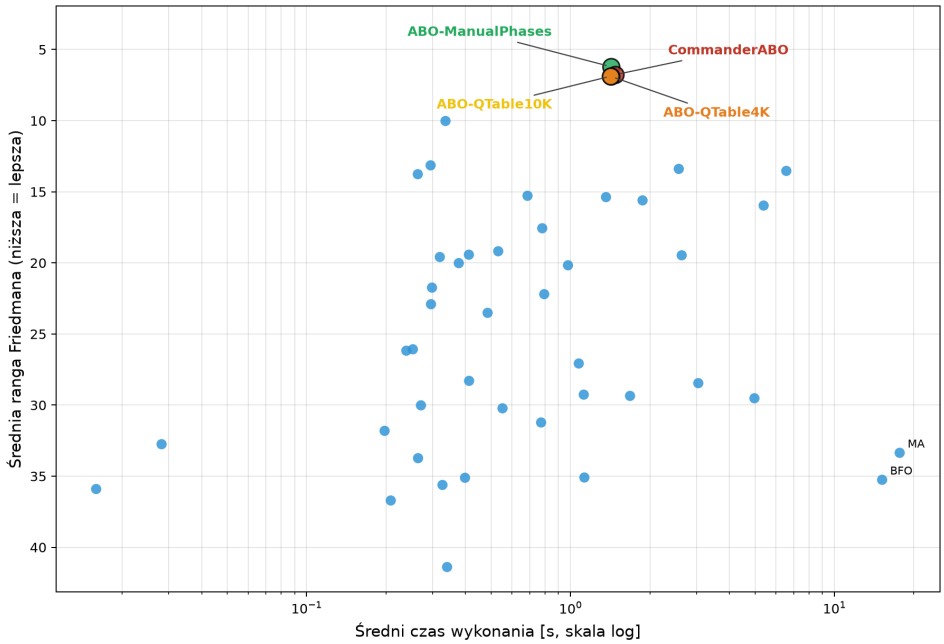
Rysunek 22 pokazuje, że CommanderABO wygrywa co najmniej 132 ze 165 konfiguracji (czyli minimum 80%) z 35 z 42 algorytmów referencyjnych; wobec 6 algorytmów (CSO, CHIO, EP, CSA, BFO, EFO) nie przegrywa w żadnej ze 165 konfiguracji (por. Rysunek 37). Najbardziej wyrównane pojedynki toczą się ze specjalistami dla niskich wymiarów (MFO – 101 wygranych CommanderABO, GWO – 109, DO – 114, BSA – 120, ALO – 126), natomiast bilanse wewnątrz rodziny ABO (Rysunek 18) są niemal zrównoważone, z lekką przewagą wariantów strojonych fazowo w kryterium mediany.

Czas wykonania



Rys. 23. Średni czas wykonania pojedynczego uruchomienia (skala logarytmiczna) dla 46 algorytmów, uśredniony po wszystkich 33 funkcjach i 5 wymiarach. Słupki błędów: odchylenie standardowe.

Kompromis szybkość vs jakość

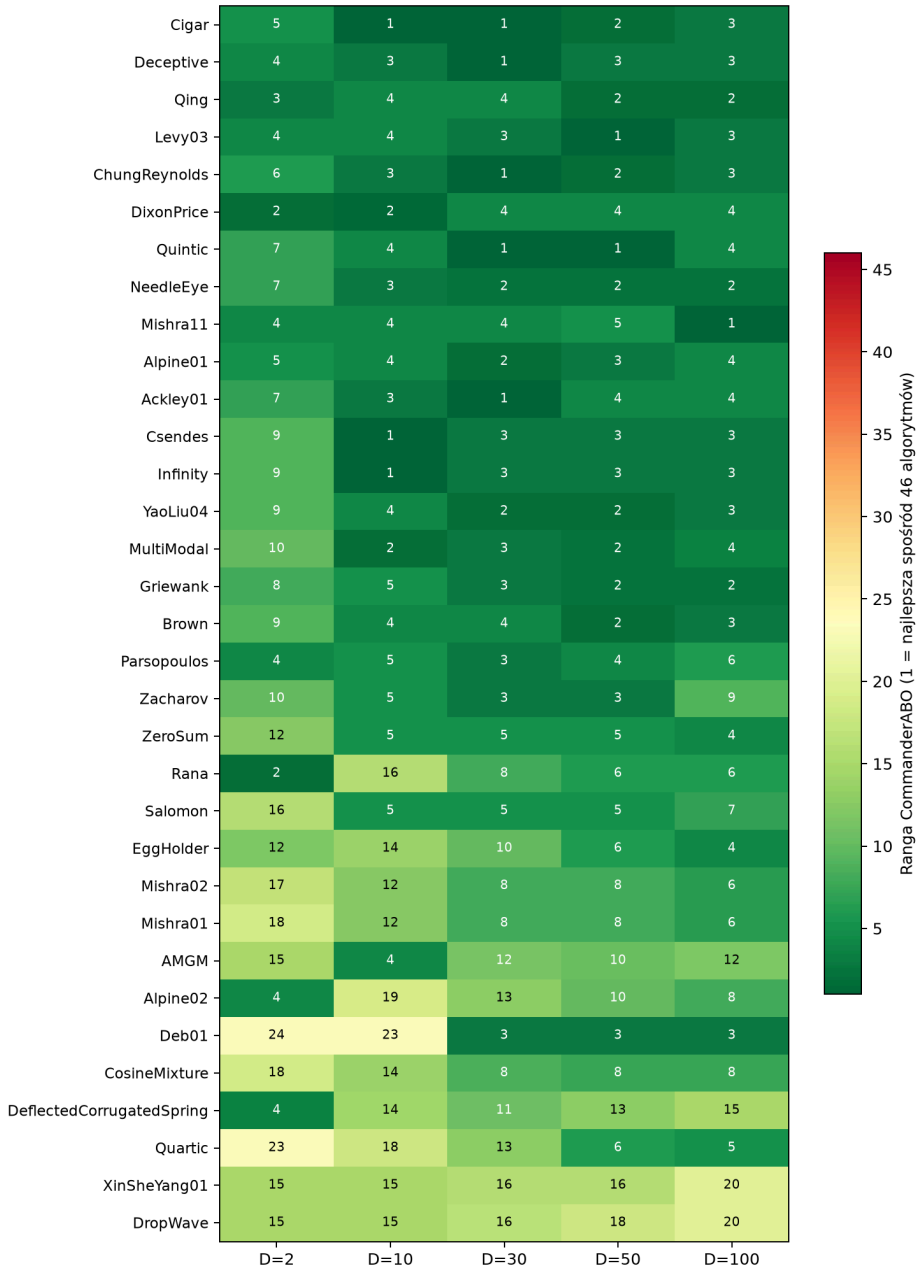


Rys. 24. Wykres rozrzutu: średnia ranga Friedmana w funkcji średniego czasu wykonania. Idealne algorytmy znajdują się w lewym górnym rogu (niski czas, niska ranga). Cztery warianty rodziny ABO wyróżniono kolorami; pozostałe – niebieskie punkty.

Rysunek 24 wizualizuje kompromis pomiędzy kosztem obliczeniowym a jakością rozwiązania. Warianty rodziny ABO znajdują się w lewym górnym sektorze wykresu – łączą najniższe rangi globalne ze stosunkowo niskim czasem wykonania (poniżej 2 s na uruchomienie).

Profil CommanderABO per (funkcja, wymiar)

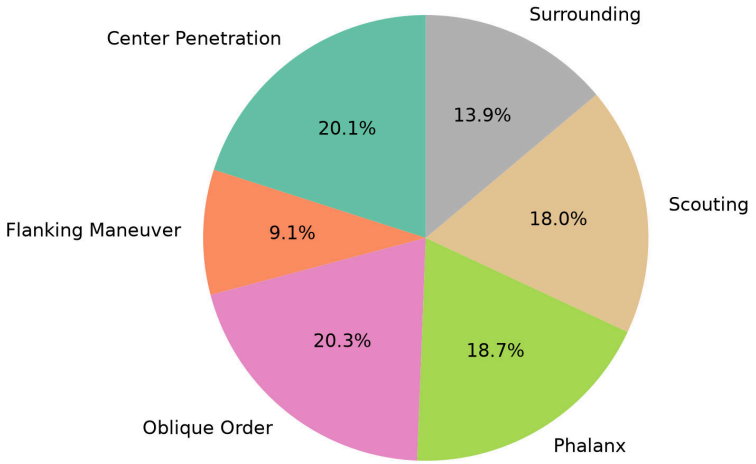
Rysunek 25 ujawnia silną zależność wydajności CommanderABO od topologii funkcji – algorytm dominuje na funkcjach unimodalnych (Cigar, ChungReynolds, Brown) oraz wybranych multimodalnych (Griewank, Levy03), podczas gdy słabsze wyniki obserwowane są dla funkcji o wąskich dolinach lub silnej nieseparowalności (Mishra11, Quintic).



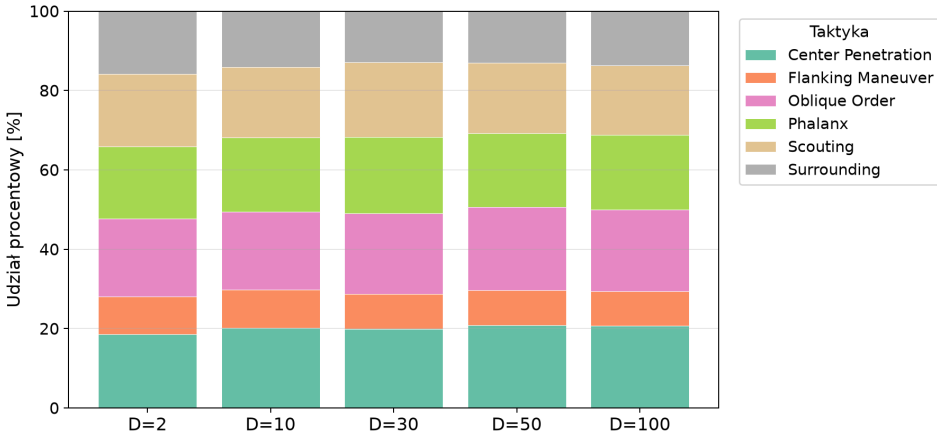
Rys. 25. Mapa cieplna rang CommanderABO dla każdej pary (funkcja, wymiar). Wiersze posortowane wg średniej rangi (najlepsze u góry). Pozwala szybko zidentyfikować klasy funkcji, w których CommanderABO osiąga przewagę oraz te wymagające dalszej optymalizacji.

Wizualizacja decyzji taktycznych Commander AI

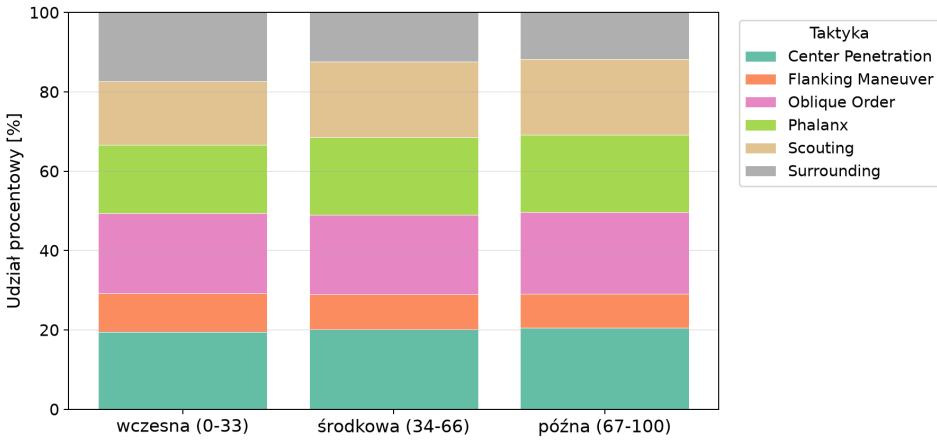
Rysunki 26–29 dokumentują zachowanie systemu Q-learning Commander AI w ujęciu udziału czasu spędzanego w poszczególnych taktykach (7 437 logowanych uruchomień po 100 iteracji; log rejestruje zmiany taktyki, więc udziały ważono czasem jej trwania). Rozkład jest zbliżony do jednostajnego i stabilny zarówno względem wymiarowości, jak i fazy optymalizacji: cztery taktyki utrzymują udział 18–21% (Oblique Order, Center Penetration, Phalanx, Scouting), a rzadziej wybierane są Surrounding (≈14%) i Flanking Maneuver (≈9%). Brak wyraźnej specjalizacji fazowej odróżnia wyuczoną politykę od ręcznego harmonogramu ABO-ManualPhases i jest spójny z obserwacją, że głównym mechanizmem Commander AI jest dobór taktyki na początku przebiegu: 67,6% uruchomień nie zawiera żadnej zmiany taktyki po pierwszej decyzji, 21,0% zawiera dokładnie jedną, a tylko 11,4% – dwie lub więcej.



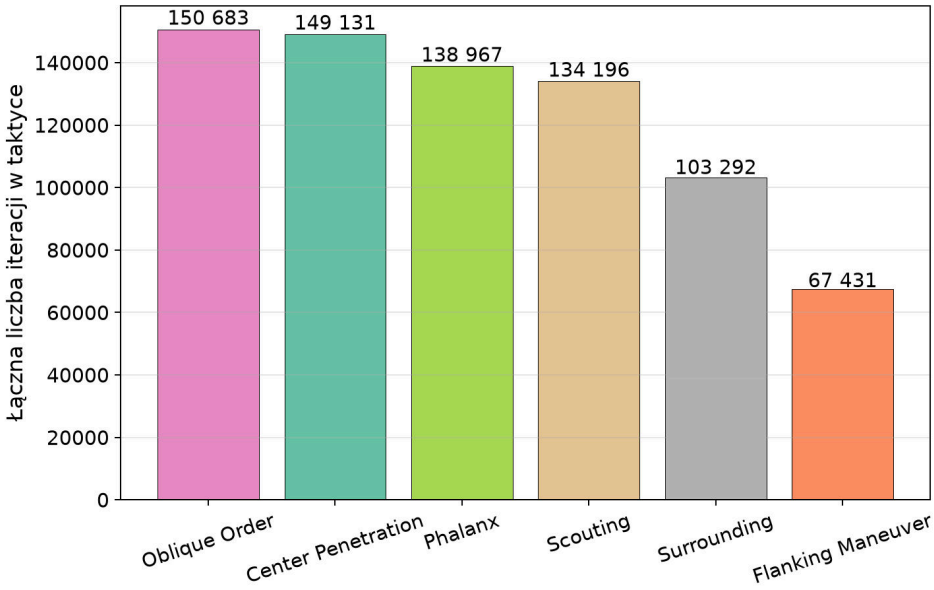
Rys. 26. Globalny udział czasu spędzanego przez Commander AI w poszczególnych taktykach (udział liczby iteracji; agregacja 7 437 logowanych uruchomień)



Rys. 27. Udział czasu taktyk Commander AI w funkcji wymiarowości problemu



Rys. 28. Udział czasu taktyk Commander AI w funkcji fazy optymalizacji (wczesna: iteracje 0–33, środkowa: 34–66, późna: 67–100)



Rys. 29. Łączny czas (liczba iteracji) spędzony przez Commander AI w poszczególnych taktykach

6.8 SZCZEGÓŁOWA ANALIZA WYNIKÓW PER FUNKCJA

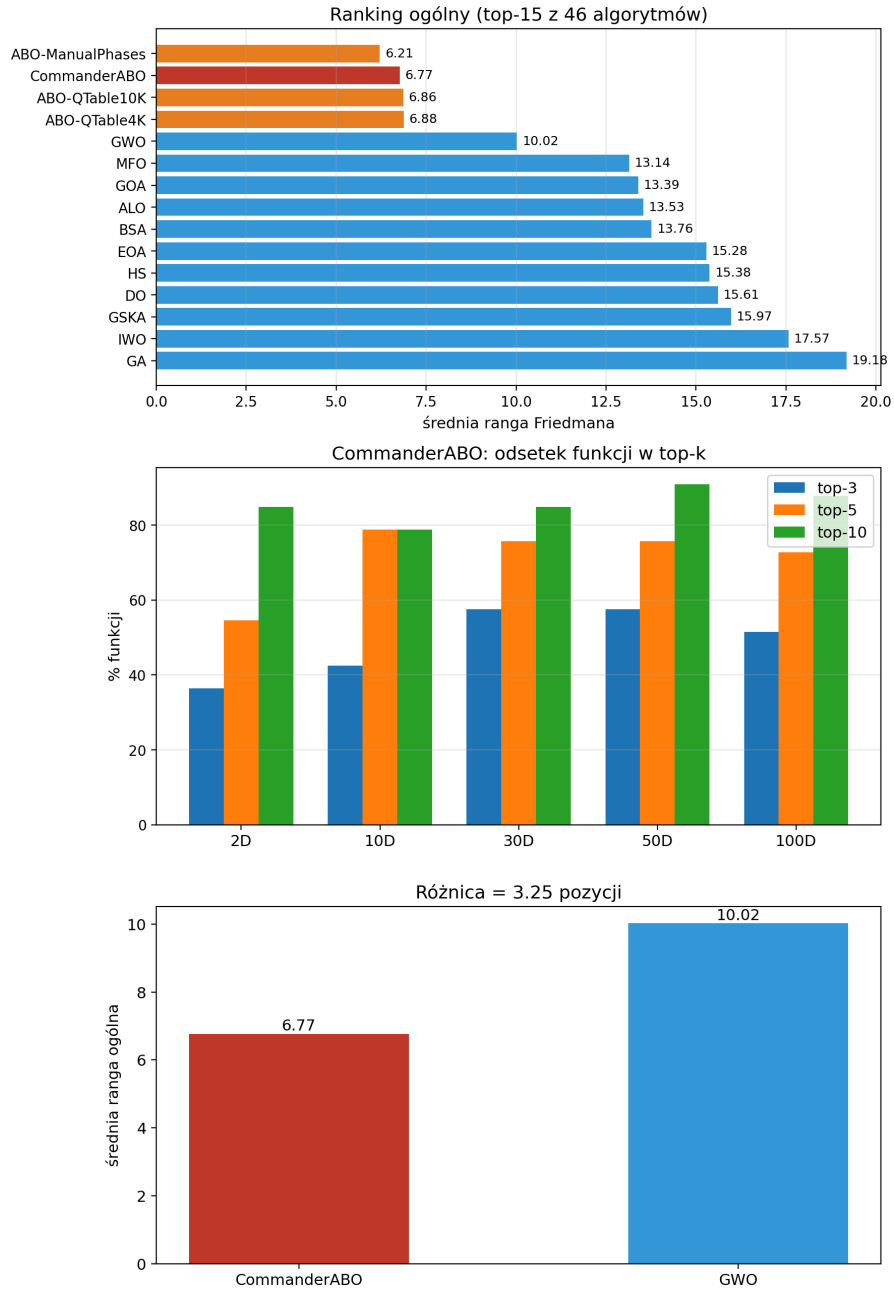
Poprzednie sekcje wykazały, że cztery warianty ABO zajmują cztery pierwsze pozycje ogólnego rankingu Friedmana, z wyraźną przewagą nad najbliższym konkurentem spoza rodziny (Tabela 33): w kryterium mediany prowadzi ABO-ManualPhases, a CommanderABO plasuje się tuż za nim, podczas gdy w kryterium best-of-run to CommanderABO osiąga najlepszą rangę w całym polu. Niniejsza sekcja uzupełnia te wyniki o **szczegółową analizę per funkcja benchmarkowa**, pokazując zachowanie CommanderABO na poziomie poszczególnych problemów testowych.

Prezentowane dane obejmują cztery komplementarne perspektywy: tabele wydajności per funkcja (Sekcja 6.8.2), mapy cieplne wydajności (Sekcja 6.8.4), testy statystyczne per funkcja (Sekcja 6.8.5), zbiorczą analizę wygranych i przegranych (Sekcja 6.8.6) oraz rozkład funkcji celu (Sekcja 6.8.7).

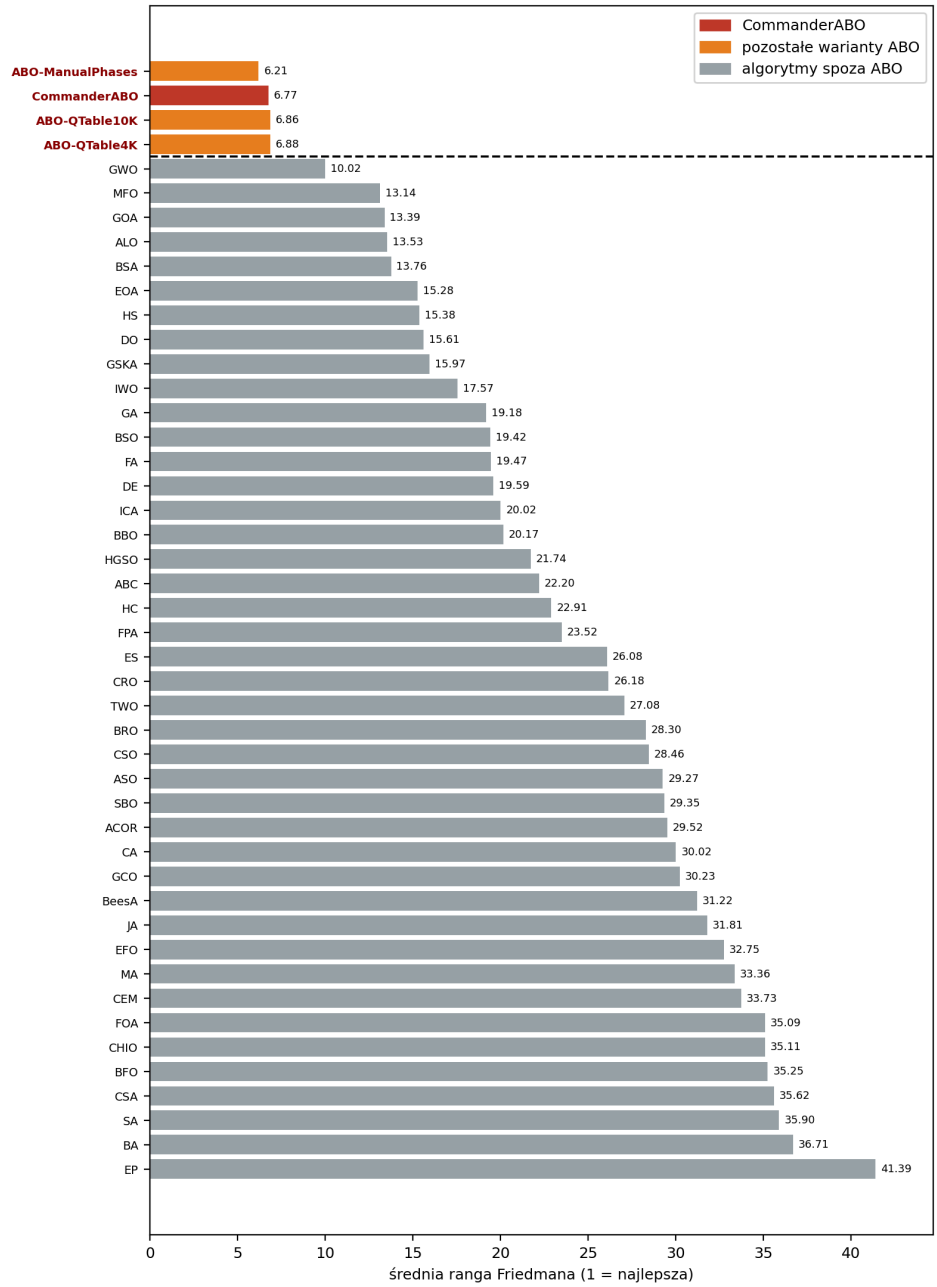
6.8.1 Podsumowanie przewagi CommanderABO

Zanim przedstawione zostaną szczegółowe wyniki per funkcja, Rysunek 30 podsumowuje trzy aspekty przewagi CommanderABO: (1) ogólny ranking Friedmana (CommanderABO: średnia ranga 6,77, 2. pozycja), (2) odsetek funkcji w top-3/top-5/top-10 w każdym wymiarze oraz (3) różnicę rankingową między rodziną ABO a pozostałymi algorytmami.

Rysunek 31 prezentuje pełny ranking 46 algorytmów: cztery warianty ABO tworzą wyraźnie odseparowaną grupę liderów, a odstęp do najlepszego algorytmu spoza rodziny (GWO) jest blisko pięciokrotnie większy niż rozrzut rang wewnątrz samej grupy ABO (3,14 wobec 0,67).



Rys. 30. Przewaga CommanderABO – podsumowanie. Górny panel: ranking ogólny 46 algorytmów (CommanderABO: ranga 6,77, 2. pozycja). Środkowy panel: odsetek funkcji, na których CommanderABO plasuje się w top-3, top-5 i top-10 w poszczególnych wymiarach. Dolny panel: różnica między CommanderABO (6,77) a najlepszym algorytmem spoza rodziny ABO (GWO, ranga 10,02) wynosi 3,25 pozycji.



Rys. 31. Ranking ogólny 46 algorytmów – cztery warianty ABO (czerwony i pomarańczowy) zajmują 4 pierwsze pozycje z wyraźną przewagą nad wszystkimi algorytmami spoza rodziny ABO (szary). Linia przerywana oddziela grupę ABO od pozostałych algorytmów.

6.8.2 Tabele wydajności algorytmów per funkcja

Tabela 50 przedstawia pozycje CommanderABO w rankingu dla każdej z 33 funkcji benchmarkowych w pięciu wymiarach (2D, 10D, 30D, 50D, 100D). Rankingi obliczono na podstawie mediany z 100 uruchomień spośród 46 algorytmów. Pogrubienie oznacza pozycję w top-3, kursywa – pozycję poza top-5. Komórki z zielonym tłem wskazują, że CommanderABO osiągnął 1. pozycję.

Tabela 50. Ranga CommanderABO na każdej funkcji benchmarkowej (spośród 46 algorytmów, single-FloatVar). Pogrubienie = top-3, kursywa = poza top-5.

Funkcja	**2D**	**10D**	**30D**	**50D**	**100D**	**Śr.**
AMGM	**1**	**1**	*10*	*10*	*11*	6,6
Ackley01	*7*	**3**	**1**	4	4	3,8
Alpine01	5	4	**2**	**3**	4	3,6
Alpine02	**3**	*19*	*13*	*10*	*8*	10,6
Brown	*9*	4	4	**2**	**3**	4,4
ChungReynolds	*6*	**3**	**1**	**2**	**3**	**3,0**
Cigar	5	**1**	**1**	**2**	**3**	**2,4**
CosineMixture	**3**	**3**	**3**	**2**	**2**	**2,6**
Csendes	*9*	**1**	**3**	**3**	**3**	3,8
Deb01	**2**	**2**	**3**	**3**	**3**	**2,6**
Deceptive	4	**3**	**1**	**3**	**3**	**2,8**
DeflectedCorrugatedSpring	**2**	*13*	*11*	*13*	*15*	10,8
DixonPrice	**2**	**1**	4	4	4	**3,0**
DropWave	*15*	*15*	*16*	*18*	*20*	16,8
EggHolder	*12*	*14*	*10*	*6*	4	9,2
Griewank	*8*	5	**3**	**1**	**1**	3,6
Infinity	*9*	**1**	**3**	**3**	**3**	3,8
Levy03	4	4	**3**	**1**	**3**	**3,0**
Mishra01	**3**	**3**	**2**	**2**	**1**	**2,2**
Mishra02	**3**	**3**	**2**	**2**	**1**	**2,2**
Mishra11	4	4	4	5	**1**	3,6
MultiModal	*10*	**2**	**3**	**1**	**1**	3,4
NeedleEye	**1**	**1**	**1**	**1**	**1**	**1,0**
Parsopoulos	4	5	**3**	4	*6*	4,4
Qing	**3**	4	4	**2**	**2**	**3,0**
Quartic	*23*	*18*	*13*	*6*	5	13,0
Quintic	*7*	4	**1**	**1**	4	3,4
Rana	**2**	*16*	*8*	*6*	*6*	7,6
Salomon	*15*	4	5	5	*7*	7,2
XinSheYang01	*15*	*15*	*16*	*16*	*20*	16,4
YaoLiu04	*9*	4	**2**	**2**	**3**	4,0
Zacharov	*10*	5	**3**	**3**	*9*	6,0
ZeroSum	**1**	5	5	5	4	4,0
Podsumowanie	12/33	14/33	19/33	19/33	17/33	81/165

Ranga spośród 46 algorytmów. Wiersz podsumowania: liczba funkcji w top-3 / łączna liczba funkcji.

Wyniki potwierdzają wysoką pozycję CommanderABO na poziomie pojedynczych funkcji. Przewaga jest szczególnie wyraźna w wymiarach średnich i wysokich: w 30D, 50D i 100D CommanderABO konsekwentnie zajmuje 2. pozycję w rankingu Friedmana (tuż za ABO-ManualPhases), a w kryterium najlepszego wyniku (best-of-run) osiąga czołową rangę w całym polu. W wymiarze 2D, gdzie algorytmy

prostsze (MFO, ABC, GWO) mają naturalną przewagę, CommanderABO utrzymuje średnią rangę 9,62 – wciąż w górnej części 46-algorytmowego rankingu. Zbiorcze wskaźniki obecności w top-k przedstawia Tabela 51.

Tabela 51 zestawia pozycję CommanderABO w rankingu 46 algorytmów z indeksem trudności każdej konfiguracji funkcja–wymiar. Dane posortowano od najtrudniejszych przypadków (indeks trudności bliski 100%) do najłatwiejszych. Na 165 konfiguracjach CommanderABO osiąga średnią rangę minimalną **5,39** (konwencja rang minimalnych dla remisów – zob. Sekcja 6.8.3; ranga Friedmana wynosi 6,77), plasując się w top-5 na **118 konfiguracjach (71,5%)**, w top-3 na **81 (49,1%)** oraz w top-10 na **141 (85,5%)**; 1. pozycję zajmuje 26 razy. Co istotne, przewaga CommanderABO narasta wraz z trudnością i wymiarowością – na najtrudniejszych konfiguracjach (trudność ≥ 85%, typowo wysokie wymiary) algorytm plasuje się w ścisłej czołówce, podczas gdy najsłabsze pozycje przypadają na najłatwiejsze przypadki 2D.

Tabela 51. Pozycja CommanderABO w rankingu 46 algorytmów (single-FloatVar) dla każdej funkcji testowej i wymiaru, posortowane według indeksu trudności. Kolory: ranga 1, top-3, top-5, top-10, >15

Funkcja	Wymiar	Trudność [%]	Ranga
Alpine02	100D	98,2	8
XinSheYang01	100D	97,9	20
Mishra01	100D	97,6	**1**
Mishra02	100D	97,6	**1**
Brown	100D	97,1	3
ChungReynolds	100D	96,7	3
Csendes	100D	96,7	3
Deb01	100D	96,7	3
Infinity	100D	96,7	3
Quintic	100D	96,7	4
Quartic	100D	96,7	5
Zacharov	100D	96,7	9
DropWave	100D	96,7	20
Parsopoulos	100D	96,2	6
Brown	50D	95,6	2
DixonPrice	100D	95,6	4
Alpine02	50D	95,6	10
XinSheYang01	50D	95,3	16
Qing	100D	94,8	2
Parsopoulos	50D	94,5	4

ciąg dalszy na następnej stronie

Tabela 51 – kontynuacja

Funkcja	Wymiar	Trudność [%]	Ranga
Quintic	50D	94,3	**1**
ChungReynolds	50D	94,3	2
Mishra01	50D	94,3	2
Qing	50D	94,3	2
Csendes	50D	94,3	3
Deb01	50D	94,3	3
Infinity	50D	94,3	3
Zacharov	50D	94,3	3
DixonPrice	50D	94,3	4
Quartic	50D	94,3	6
Mishra02	50D	94,2	2
DropWave	50D	94	18
Brown	30D	93,8	4
Levy03	50D	93,6	**1**
Alpine02	30D	93,5	13
Zacharov	30D	93,2	3
XinSheYang01	30D	93,1	16
ChungReynolds	30D	92,5	**1**
Cigar	30D	92,5	**1**
Csendes	30D	92,5	3
Deb01	30D	92,5	3
Infinity	30D	92,5	3
Levy03	30D	92,5	3
Qing	30D	92,5	4
Quartic	30D	92,5	13
DropWave	30D	92,5	16
Parsopoulos	30D	92,3	3
DixonPrice	30D	92,3	4
Quintic	30D	92,2	**1**
Mishra01	30D	92,2	2
Mishra02	30D	92,2	2
ZeroSum	50D	91,3	5
ZeroSum	100D	90,2	4
Levy03	100D	89,4	3
Zacharov	10D	89,2	5
Cigar	50D	89,1	2
ZeroSum	30D	89	5
Brown	10D	88,7	4

ciąg dalszy na następnej stronie

Tabela 51 – kontynuacja

Funkcja	Wymiar	Trudność [%]	Ranga
Cigar	10D	88,6	**1**
Csendes	10D	88,6	**1**
DixonPrice	10D	88,6	**1**
Infinity	10D	88,6	**1**
Deb01	10D	88,6	2
ChungReynolds	10D	88,6	3
Levy03	10D	88,6	4
Quintic	10D	88,6	4
XinSheYang01	10D	88,6	15
Quartic	10D	88,6	18
Qing	10D	88,5	4
Parsopoulos	10D	88,5	5
Mishra02	10D	87,7	3
Mishra01	10D	87,6	3
Alpine02	10D	87,3	19
Cigar	100D	86	3
Alpine01	10D	85,3	4
Griewank	50D	84,9	**1**
Griewank	100D	84,5	**1**
Griewank	30D	84,1	3
Csendes	2D	83,9	9
Infinity	2D	83,9	9
Cigar	2D	83,6	5
ChungReynolds	2D	83,6	6
YaoLiu04	10D	83,3	4
DeflectedCorrugatedSpring	10D	83,3	13
Quartic	2D	83,1	23
ZeroSum	10D	83	5
YaoLiu04	2D	83	9
Salomon	2D	83	15
Deb01	2D	82,9	2
DixonPrice	2D	82,9	2
Qing	2D	82,9	3
XinSheYang01	2D	82,9	15
Parsopoulos	2D	82,7	4
Ackley01	2D	82,6	7
DeflectedCorrugatedSpring	30D	82,5	11
Rana	100D	82,4	6

ciąg dalszy na następnej stronie

Tabela 51 – kontynuacja

Funkcja	**Wymiar**	**Trudność [%]**	**Ranga**
Levy03	2D	81,9	4
EggHolder	100D	81,7	4
Alpine01	30D	79	2
AMGM	100D	78,3	11
Alpine01	50D	77,5	3
DeflectedCorrugatedSpring	50D	76,8	13
Alpine01	100D	76,6	4
Griewank	2D	76,3	8
Griewank	10D	76	5
Salomon	10D	74,7	4
DeflectedCorrugatedSpring	100D	74,7	15
AMGM	50D	74,3	10
AMGM	30D	72	10
EggHolder	50D	71,8	6
Ackley01	10D	71,7	3
YaoLiu04	30D	71,3	2
Quintic	2D	70,5	7
Salomon	30D	70,1	5
Rana	50D	69,5	6
Salomon	50D	68,9	5
ZeroSum	2D	68,7	**1**
Salomon	100D	67,5	7
YaoLiu04	50D	65,8	2
EggHolder	30D	64,2	10
YaoLiu04	100D	64,1	3
AMGM	10D	64	**1**
DropWave	10D	63,5	15
Mishra11	100D	63,3	**1**
Alpine01	2D	63,2	5
MultiModal	100D	62,8	**1**
MultiModal	30D	62,7	3
Rana	30D	62,6	8
Brown	2D	62,2	9
MultiModal	50D	62,1	**1**
Mishra11	50D	61,8	5
MultiModal	10D	61,4	2
Deceptive	100D	61,1	3
Zacharov	2D	60,8	10

ciąg dalszy na następnej stronie

Tabela 51 – kontynuacja

Funkcja	Wymiar	Trudność [%]	Ranga
Mishra11	10D	60,2	4
Mishra11	30D	60,1	4
CosineMixture	100D	59,5	2
CosineMixture	10D	58,7	3
Deceptive	50D	58,1	3
CosineMixture	50D	57,7	2
Ackley01	30D	57,3	**1**
Mishra11	2D	57	4
MultiModal	2D	56,7	10
NeedleEye	2D	55,6	**1**
CosineMixture	30D	55,5	3
Ackley01	50D	55,4	4
Ackley01	100D	54,9	4
AMGM	2D	54,7	**1**
Deceptive	30D	53,6	**1**
CosineMixture	2D	53,2	3
EggHolder	10D	50,7	14
Rana	10D	48,4	16
Deceptive	10D	46,1	3
NeedleEye	10D	43,2	**1**
NeedleEye	100D	43	**1**
NeedleEye	30D	39,7	**1**
NeedleEye	50D	38,7	**1**
Deceptive	2D	37,2	4
Mishra01	2D	35,3	3
Mishra02	2D	34,3	3
DeflectedCorrugatedSpring	2D	20,1	2
EggHolder	2D	17,2	12
Rana	2D	8,7	2
DropWave	2D	7,8	15
Alpine02	2D	3,2	3

Szczegółowe tabele wydajności dla poszczególnych funkcji, zawierające medianę fitness i rangi top-10 algorytmów w każdym wymiarze, zamieszczono poniżej. W każdej tabeli CommanderABO jest wyróżniony pogrubieniem. We wszystkich 33 tabelach CommanderABO pojawia się w top-10, co potwierdza jego konsekwentnie wysoką wydajność niezależnie od typu funkcji.

6.8.3 Uwaga o konwencji rankingu

W szczegółowych tabelach per funkcja kolumna *Śr. ranga* podaje **średnią rangę Friedmana** (remisy otrzymują rangę uśrednioną; w `pandas` odpowiada to `method='average'`) – jest to standardowa konwencja testu Friedmana, dająca jednoznaczne uporządkowanie czołowej dziesiątki. Różni się ona od konwencji **rangi minimalnej** (remisy otrzymują najlepszą możliwą rangę, `method='min'`), stosowanej w Tabeli 50, w Tabeli 51 oraz przy zliczaniu obecności w top-k (81/165 w top-3 itd.). Rozbieżność ujawnia się jedynie na funkcjach „łatwych", gdzie wiele algorytmów osiąga to samo optimum numeryczne: ranga uśredniona umieszcza algorytm współ-optymalny w środku bloku remisowego (np. CommanderABO na funkcji AMGM uzyskuje 10,7 w ujęciu uśrednionym wobec 6,6 w ujęciu minimalnym), podczas gdy ranga minimalna przypisuje całemu blokowi najlepszą pozycję. Obie miary są poprawne i prowadzą do tych samych wniosków jakościowych; w rozprawie podawane są równolegle dla pełnej przejrzystości, a zliczenia top-k oraz wszystkie wnioski narracyjne opierają się konsekwentnie na ujęciu minimalnym.

Tabela 52. Wyniki dla funkcji AMGM – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
BSA	**0**	**0**	**0**	**0**	**0**	6,9
HGSO	0	0	0	0	0	6,9
MFO	0	0	0	0	0	6,9
ALO	0	$3{,}16 \times 10^{-30}$	0	0	0	8,8
BA	0	$3{,}16 \times 10^{-30}$	0	0	0	8,8
BRO	0	$3{,}16 \times 10^{-30}$	0	0	0	8,8
DO	0	$3{,}16 \times 10^{-30}$	0	0	0	8,8
FOA	0	$3{,}16 \times 10^{-30}$	0	0	0	8,8
HC	0	$3{,}16 \times 10^{-30}$	0	0	0	8,8
CommanderABO	0	0	$1{,}26 \times 10^{-29}$	$7{,}89 \times 10^{-29}$	$4{,}17 \times 10^{-28}$	10,7

Tabela 53. Wyniki dla funkcji Ackley01 – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-ManualPhases	$1{,}50 \times 10^{-8}$	$1{,}83 \times 10^{-8}$	$1{,}21 \times 10^{-8}$	$1{,}13 \times 10^{-8}$	$1{,}14 \times 10^{-8}$	2,8
ABO-QTable4K	$1{,}81 \times 10^{-8}$	$1{,}60 \times 10^{-8}$	$1{,}29 \times 10^{-8}$	$1{,}15 \times 10^{-8}$	$1{,}14 \times 10^{-8}$	3,6
ABO-QTable10K	$1{,}81 \times 10^{-8}$	$1{,}67 \times 10^{-8}$	$1{,}29 \times 10^{-8}$	$1{,}16 \times 10^{-8}$	$1{,}14 \times 10^{-8}$	3,8
CommanderABO	$1{,}66 \times 10^{-8}$	$1{,}73 \times 10^{-8}$	$1{,}18 \times 10^{-8}$	$1{,}20 \times 10^{-8}$	$1{,}23 \times 10^{-8}$	3,8
GWO	$4{,}44 \times 10^{-16}$	$6{,}77 \times 10^{-7}$	0,007165	0,08176	2,544	4,8
MFO	$4{,}00 \times 10^{-15}$	1,836	2,408	2,435	2,54	6,6
EOA	0,06913	0,0992	0,02529	0,008592	$9{,}79 \times 10^{-6}$	9,0
DO	$4{,}75 \times 10^{-10}$	3,272	7,385	8,554	9,441	10,4
BSO	0,01209	0,4453	5,612	9,181	12,37	11,8
HS	0,004607	2,451	7,202	10,94	14,94	13,6

Tabela 54. Wyniki dla funkcji Alpine01 – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-ManualPhases	$6{,}98 \times 10^{-10}$	$5{,}84 \times 10^{-8}$	$1{,}17 \times 10^{-7}$	$2{,}23 \times 10^{-7}$	$4{,}52 \times 10^{-7}$	1,2
ABO-QTable10K	$7{,}63 \times 10^{-10}$	$7{,}62 \times 10^{-8}$	$2{,}38 \times 10^{-7}$	$3{,}82 \times 10^{-7}$	$5{,}95 \times 10^{-7}$	2,9
ABO-QTable4K	$7{,}63 \times 10^{-10}$	$7{,}62 \times 10^{-8}$	$2{,}65 \times 10^{-7}$	$3{,}31 \times 10^{-7}$	$6{,}15 \times 10^{-7}$	3,1
CommanderABO	$9{,}75 \times 10^{-10}$	$1{,}35 \times 10^{-7}$	$2{,}01 \times 10^{-7}$	$3{,}55 \times 10^{-7}$	$7{,}41 \times 10^{-7}$	3,6
GWO	$2{,}11 \times 10^{-6}$	0,001196	0,02082	1,628	12,65	7,2
EOA	0,001028	0,02219	0,005998	0,002737	$3{,}73 \times 10^{-6}$	9,4
HS	$3{,}63 \times 10^{-5}$	0,03337	1,901	7,797	37,62	10,2
IWO	$4{,}01 \times 10^{-5}$	0,005453	1,345	11,07	61,28	10,8
ICA	$2{,}33 \times 10^{-9}$	0,004974	6,31	22,98	83,34	11,6
GOA	$4{,}59 \times 10^{-6}$	0,01341	2,566	12,91	65,22	11,8

Tabela 55. Wyniki dla funkcji Alpine02 – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
HS	-6,129	$-2{,}33 \times 10^{4}$	$-6{,}16 \times 10^{12}$	$-1{,}20 \times 10^{20}$	$-2{,}01 \times 10^{35}$	5,4
BSA	-6,129	-6087	$-2{,}22 \times 10^{12}$	$-5{,}05 \times 10^{20}$	$-1{,}44 \times 10^{41}$	6,0
ABO-ManualPhases	-6,13	-4029	$-8{,}57 \times 10^{9}$	$-2{,}29 \times 10^{16}$	$-4{,}44 \times 10^{31}$	7,4
GOA	-6,13	$-1{,}10 \times 10^{4}$	$-7{,}70 \times 10^{9}$	$-3{,}13 \times 10^{15}$	$-9{,}34 \times 10^{26}$	8,8
MFO	**-6,13**	-9830	$-3{,}99 \times 10^{9}$	$-3{,}36 \times 10^{14}$	$-1{,}42 \times 10^{26}$	8,9
ABO-QTable4K	-6,13	-4029	$-6{,}49 \times 10^{9}$	$-1{,}11 \times 10^{16}$	$-1{,}57 \times 10^{31}$	9,3
BBO	-6,09	$-1{,}97 \times 10^{4}$	$-2{,}09 \times 10^{12}$	$-8{,}82 \times 10^{19}$	$-4{,}04 \times 10^{36}$	9,4
DE	-6,129	-8902	$-3{,}03 \times 10^{10}$	$-4{,}70 \times 10^{15}$	$-1{,}67 \times 10^{26}$	9,4
ABO-QTable10K	-6,13	-4029	$-5{,}16 \times 10^{9}$	$-1{,}11 \times 10^{16}$	$-1{,}45 \times 10^{31}$	9,7
GA	-6,115	$-1{,}38 \times 10^{4}$	$-1{,}61 \times 10^{12}$	$-3{,}72 \times 10^{18}$	$-1{,}83 \times 10^{31}$	10,0

Tabela 56. Wyniki dla funkcji Brown – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-ManualPhases	$1{,}98 \times 10^{-17}$	$2{,}77 \times 10^{-16}$	$8{,}21 \times 10^{-16}$	$1{,}48 \times 10^{-15}$	$3{,}19 \times 10^{-15}$	2,4
ABO-QTable4K	$2{,}29 \times 10^{-17}$	$2{,}93 \times 10^{-16}$	$8{,}40 \times 10^{-16}$	$1{,}65 \times 10^{-15}$	$3{,}56 \times 10^{-15}$	4,1
CommanderABO	$2{,}27 \times 10^{-17}$	$3{,}12 \times 10^{-16}$	$8{,}55 \times 10^{-16}$	$1{,}59 \times 10^{-15}$	$3{,}59 \times 10^{-15}$	4,4
GWO	$1{,}14 \times 10^{-67}$	$3{,}58 \times 10^{-15}$	$1{,}78 \times 10^{-6}$	$5{,}87 \times 10^{-4}$	0,1452	4,4
ABO-QTable10K	$2{,}29 \times 10^{-17}$	$2{,}93 \times 10^{-16}$	$8{,}40 \times 10^{-16}$	$1{,}71 \times 10^{-15}$	$3{,}60 \times 10^{-15}$	4,7
ALO	$1{,}69 \times 10^{-14}$	$2{,}18 \times 10^{-8}$	0,6771	3,373	14,33	8,4
MFO	$1{,}66 \times 10^{-39}$	0,03222	2,338	6,444	17,78	10,0
GSKA	$1{,}35 \times 10^{-18}$	$1{,}99 \times 10^{-4}$	0,6826	7,94	411,5	10,4
IWO	$1{,}04 \times 10^{-7}$	$7{,}10 \times 10^{-4}$	0,1444	2,112	77,77	11,0
HGSO	$6{,}63 \times 10^{-4}$	0,01973	0,07144	0,1319	0,2973	12,8

Tabela 57. Wyniki dla funkcji ChungReynolds – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-QTable10K	$5{,}87 \times 10^{-33}$	$1{,}18 \times 10^{-32}$	$3{,}56 \times 10^{-31}$	$1{,}01 \times 10^{-30}$	$1{,}53 \times 10^{-30}$	3,0
CommanderABO	$2{,}45 \times 10^{-33}$	$1{,}50 \times 10^{-32}$	$3{,}04 \times 10^{-31}$	$7{,}34 \times 10^{-31}$	$2{,}01 \times 10^{-30}$	3,0
ABO-ManualPhases	$1{,}10 \times 10^{-33}$	$1{,}85 \times 10^{-32}$	$5{,}71 \times 10^{-31}$	$6{,}72 \times 10^{-31}$	$2{,}59 \times 10^{-30}$	3,6
ABO-QTable4K	$5{,}87 \times 10^{-33}$	$1{,}37 \times 10^{-32}$	$3{,}56 \times 10^{-31}$	$1{,}07 \times 10^{-30}$	$1{,}46 \times 10^{-30}$	3,6
GWO	$1{,}52 \times 10^{-130}$	$5{,}35 \times 10^{-24}$	$4{,}11 \times 10^{-7}$	0,01585	259,7	4,6
EOA	$1{,}59 \times 10^{-5}$	$7{,}90 \times 10^{-4}$	$1{,}01 \times 10^{-4}$	$2{,}25 \times 10^{-6}$	$6{,}96 \times 10^{-18}$	8,6
DO	$5{,}51 \times 10^{-39}$	644	$4{,}86 \times 10^{5}$	$2{,}53 \times 10^{6}$	$2{,}09 \times 10^{7}$	9,8
GSKA	$3{,}01 \times 10^{-29}$	0,0309	$1{,}00 \times 10^{5}$	$6{,}04 \times 10^{6}$	$2{,}84 \times 10^{8}$	10,2
GOA	$1{,}03 \times 10^{-14}$	0,005303	923	$1{,}54 \times 10^{6}$	$3{,}34 \times 10^{9}$	11,4
BSA	$9{,}56 \times 10^{-10}$	606,9	$1{,}08 \times 10^{6}$	$6{,}49 \times 10^{6}$	$5{,}60 \times 10^{7}$	13,2

Tabela 58. Wyniki dla funkcji Cigar – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
CommanderABO	$3{,}39 \times 10^{-11}$	$1{,}78 \times 10^{-10}$	$5{,}05 \times 10^{-10}$	$8{,}49 \times 10^{-10}$	$1{,}37 \times 10^{-9}$	2,4
ABO-QTable10K	$7{,}96 \times 10^{-11}$	$1{,}81 \times 10^{-10}$	$6{,}12 \times 10^{-10}$	$1{,}01 \times 10^{-9}$	$1{,}18 \times 10^{-9}$	3,6
ABO-ManualPhases	$4{,}55 \times 10^{-11}$	$2{,}16 \times 10^{-10}$	$7{,}76 \times 10^{-10}$	$7{,}87 \times 10^{-10}$	$1{,}56 \times 10^{-9}$	3,8
ABO-QTable4K	$7{,}96 \times 10^{-11}$	$1{,}81 \times 10^{-10}$	$6{,}12 \times 10^{-10}$	$1{,}01 \times 10^{-9}$	$1{,}18 \times 10^{-9}$	3,8
GWO	$3{,}18 \times 10^{-61}$	$8{,}05 \times 10^{-7}$	534,8	$1{,}02 \times 10^{5}$	$1{,}61 \times 10^{7}$	4,6
EOA	2,962	$1{,}54 \times 10^{4}$	5560	1582	0,0012	7,8
DO	$1{,}44 \times 10^{-16}$	$9{,}00 \times 10^{6}$	$6{,}51 \times 10^{8}$	$1{,}54 \times 10^{9}$	$4{,}30 \times 10^{9}$	9,6
GSKA	$9{,}75 \times 10^{-11}$	$6{,}24 \times 10^{4}$	$2{,}96 \times 10^{8}$	$2{,}21 \times 10^{9}$	$1{,}73 \times 10^{10}$	10,6
GOA	$2{,}03 \times 10^{-6}$	$2{,}03 \times 10^{4}$	$2{,}22 \times 10^{7}$	$1{,}21 \times 10^{9}$	$5{,}85 \times 10^{10}$	11,0
HS	13,93	$9{,}36 \times 10^{4}$	$5{,}08 \times 10^{8}$	$3{,}38 \times 10^{9}$	$2{,}38 \times 10^{10}$	13,4

Tabela 59. Wyniki dla funkcji CosineMixture – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
SA	**-7,26**	**-23,83**	**-59,15**	**-87,14**	**-160,6**	1,0
FOA	-3,38	-10,95	-27,23	-45	-90	4,2
ABO-ManualPhases	-1,8	-9	-27	-45	-90	11,2
ABO-QTable10K	-1,8	-9	-27	-45	-90	11,2
ABO-QTable4K	-1,8	-9	-27	-45	-90	11,2
ALO	-1,8	-9	-27	-45	-90	11,2
BA	-1,8	-9	-27	-45	-90	11,2
BRO	-1,8	-9	-27	-45	-90	11,2
BSA	-1,8	-9	-27	-45	-90	11,2
CommanderABO	-1,8	-9	-27	-45	-90	11,2

Tabela 60. Wyniki dla funkcji Csendes – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-QTable10K	$6{,}77 \times 10^{-51}$	$2{,}69 \times 10^{-44}$	$2{,}69 \times 10^{-49}$	$6{,}93 \times 10^{-49}$	$3{,}33 \times 10^{-48}$	2,7
ABO-QTable4K	$6{,}77 \times 10^{-51}$	$2{,}69 \times 10^{-44}$	$2{,}61 \times 10^{-49}$	$7{,}47 \times 10^{-49}$	$3{,}33 \times 10^{-48}$	2,7
CommanderABO	$9{,}81 \times 10^{-51}$	$1{,}40 \times 10^{-44}$	$5{,}25 \times 10^{-49}$	$1{,}21 \times 10^{-48}$	$4{,}49 \times 10^{-48}$	3,8
ABO-ManualPhases	$9{,}51 \times 10^{-51}$	$6{,}02 \times 10^{-44}$	$3{,}36 \times 10^{-48}$	$3{,}55 \times 10^{-48}$	$8{,}36 \times 10^{-48}$	4,8
GWO	$3{,}50 \times 10^{-196}$	$4{,}99 \times 10^{-37}$	$1{,}43 \times 10^{-16}$	$1{,}65 \times 10^{-11}$	$3{,}22 \times 10^{-7}$	5,0
EOA	$6{,}47 \times 10^{-20}$	$5{,}33 \times 10^{-18}$	$1{,}85 \times 10^{-20}$	$6{,}13 \times 10^{-23}$	$2{,}36 \times 10^{-41}$	8,0
DO	$4{,}84 \times 10^{-69}$	$1{,}12 \times 10^{-9}$	$6{,}13 \times 10^{-6}$	$3{,}42 \times 10^{-5}$	$1{,}72 \times 10^{-4}$	8,6
BSA	$4{,}25 \times 10^{-32}$	$4{,}51 \times 10^{-12}$	$1{,}31 \times 10^{-5}$	$1{,}39 \times 10^{-4}$	$8{,}11 \times 10^{-4}$	10,4
GSKA	$4{,}09 \times 10^{-54}$	$2{,}68 \times 10^{-13}$	$5{,}15 \times 10^{-5}$	0,003808	0,121	11,6
HGSO	$9{,}47 \times 10^{-13}$	$1{,}09 \times 10^{-10}$	$5{,}97 \times 10^{-10}$	$8{,}45 \times 10^{-10}$	$1{,}60 \times 10^{-9}$	12,8

Tabela 61. Wyniki dla funkcji Deb01 – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
SA	**-0,962**	**-0,6085**	**-0,48**	**-0,4414**	**-0,402**	1,0
ABO-ManualPhases	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	-0,4248	-0,3893	-0,3557	10,5
CommanderABO	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	-0,1148	-0,1778	-0,02547	11,1
ABO-QTable10K	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	-0,0666	-0,03994	-0,01996	12,0
ABO-QTable4K	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	-0,0666	-0,03994	-0,01996	12,0
ABC	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	24,6
ACOR	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	24,6
ALO	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	24,6
ASO	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	24,6
BA	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	$-5{,}27 \times 10^{-92}$	24,6

Tabela 62. Wyniki dla funkcji Deceptive – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-QTable4K	**-1**	**-0,9604**	-0,9735	**-0,9841**	**-0,9841**	1,8
ABO-QTable10K	-1	-0,9604	-0,9867	-0,9841	-0,9841	2,0
CommanderABO	-1	-0,9604	**-0,9867**	-0,9841	-0,9841	2,8
ABO-ManualPhases	-1	-0,7744	-0,9735	-0,9761	-0,9801	5,0
HS	-0,9702	-0,9391	-0,8459	-0,7716	-0,6753	7,0
GA	-0,9376	-0,8907	-0,8197	-0,7417	-0,653	8,6
BBO	-0,8837	-0,84	-0,7964	-0,7614	-0,6852	9,8
ICA	-1	-0,7744	-0,6835	-0,5818	-0,4529	12,8
MFO	-0,81	-0,6724	-0,6507	-0,6464	-0,64	14,5
BSA	-0,9833	-0,6405	-0,64	-0,64	-0,64	15,9

Tabela 63. Wyniki dla funkcji DeflectedCorrugatedSpring – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
EOA	-0,9983	-0,8433	**-0,8433**	**-0,8433**	**-0,8433**	4,0
BSA	-0,9999	-0,8433	0,4099	1,506	2,916	5,0
GSKA	-0,9974	-0,8433	0,04547	1,507	6,652	6,6
BSO	-0,9982	-0,8433	0,4099	2,916	7,697	7,4
GWO	-1	-0,8433	0,4099	4,639	18,42	8,0
FA	-1	**-0,8433**	1,506	4,639	17,96	8,2
MFO	**-1**	-0,3734	2,916	6,676	14,66	9,8
HS	-0,8433	-0,8431	0,4099	1,507	6,683	10,8
IWO	-0,9956	-0,8314	1,506	4,639	17,95	11,0
CommanderABO	-1	-0,3734	1,507	8,144	25,47	11,4

Tabela 64. Wyniki dla funkcji DixonPrice – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-QTable4K	$1{,}35 \times 10^{-15}$	0,6667	$1{,}13 \times 10^{-4}$	**0,6667**	3,534	2,2
ABO-QTable10K	$1{,}35 \times 10^{-15}$	0,6667	0,4832	0,6667	3,689	2,8
CommanderABO	$1{,}28 \times 10^{-15}$	**0,6667**	0,6667	0,667	5,318	3,1
ABO-ManualPhases	$1{,}38 \times 10^{-15}$	0,6667	0,6667	0,6667	8,253	3,5
GWO	$4{,}15 \times 10^{-7}$	0,6668	0,6723	1,172	64,87	6,6
IWO	$3{,}97 \times 10^{-7}$	0,6682	14,25	979,8	$5{,}45 \times 10^{5}$	9,2
EOA	0,006805	0,7639	0,8006	0,8885	**0,9995**	9,4
GOA	$7{,}32 \times 10^{-9}$	0,6966	53,78	8024	$2{,}16 \times 10^{6}$	11,8
BSO	$1{,}03 \times 10^{-4}$	1,135	114,8	3922	$1{,}04 \times 10^{5}$	12,0
GSKA	$5{,}71 \times 10^{-6}$	0,7485	355,1	$1{,}40 \times 10^{4}$	$4{,}56 \times 10^{5}$	12,2

Tabela 65. Wyniki dla funkcji DropWave – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
GWO	**-1**	**-0,9362**	-0,6195	-0,3691	-0,1274	2,0
DO	-1	-0,7858	-0,3691	-0,2297	-0,1528	3,2
EOA	-0,9969	-0,9362	**-0,9362**	**-0,9362**	**-0,7858**	3,2
BSA	-0,9994	-0,7857	-0,2888	-0,1858	-0,09205	5,8
FA	-0,9999	-0,7858	-0,2276	-0,06921	-0,0194	7,0
BSO	-0,9921	-0,7857	-0,1858	-0,07951	-0,03397	9,8
ABO-ManualPhases	-0,9994	-0,7821	-0,1273	-0,04414	-0,01496	11,2
HS	-0,9362	-0,7858	-0,2776	-0,09184	-0,01905	11,4
HC	-0,9846	-0,7854	-0,2295	-0,06092	-0,01246	13,0
IWO	-0,9979	-0,7858	-0,1528	-0,03203	-0,008605	13,2

Tabela 66. Wyniki dla funkcji EggHolder – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
MFO	**-959,6**	**-7067**	$-1{,}88 \times 10^{4}$	$-2{,}79 \times 10^{4}$	$-5{,}03 \times 10^{4}$	2,8
ABO-ManualPhases	-959,6	-4738	$-1{,}53 \times 10^{4}$	$-3{,}12 \times 10^{4}$	$-6{,}80 \times 10^{4}$	5,6
ABO-QTable10K	-959,6	-5020	$-1{,}54 \times 10^{4}$	$-3{,}04 \times 10^{4}$	$-6{,}76 \times 10^{4}$	6,4
ALO	-935,3	-5918	$-1{,}80 \times 10^{4}$	$-3{,}06 \times 10^{4}$	$-5{,}66 \times 10^{4}$	6,4
ABO-QTable4K	-959,6	-5133	$-1{,}54 \times 10^{4}$	$-3{,}03 \times 10^{4}$	$-6{,}74 \times 10^{4}$	6,6
GOA	-959,6	-6112	$-1{,}59 \times 10^{4}$	$-2{,}28 \times 10^{4}$	$-3{,}74 \times 10^{4}$	6,6
CommanderABO	-959,6	-4718	$-1{,}45 \times 10^{4}$	$-2{,}70 \times 10^{4}$	$-6{,}70 \times 10^{4}$	9,2
HS	-928,1	-6757	$-1{,}80 \times 10^{4}$	$-2{,}50 \times 10^{4}$	$-3{,}30 \times 10^{4}$	9,4
GA	-883,1	-6150	$-1{,}77 \times 10^{4}$	$-2{,}67 \times 10^{4}$	$-4{,}50 \times 10^{4}$	11,0
BRO	-952,3	-5401	$-1{,}34 \times 10^{4}$	$-2{,}05 \times 10^{4}$	$-3{,}81 \times 10^{4}$	11,2

Tabela 67. Wyniki dla funkcji Griewank – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-ManualPhases	0,007396	0,09104	0,007396	**0**	**0**	3,1
ABO-QTable10K	0,007396	0,107	0,007396	0	0	3,5
GWO	0,007396	**0,05822**	$6{,}75 \times 10^{-5}$	0,005209	0,2548	4,0
CommanderABO	0,007718	0,1095	0,007396	0	0	4,2
ABO-QTable4K	0,007402	0,1156	0,008627	0	0	4,8
EOA	0,01117	0,2545	$4{,}41 \times 10^{-4}$	$3{,}59 \times 10^{-5}$	$4{,}38 \times 10^{-11}$	7,8
GOA	0,007396	0,1773	0,721	1,31	15,44	9,2
GSKA	**0,002899**	0,6903	1,08	1,615	5,217	9,8
HS	0,009869	0,07045	1,151	1,958	7,144	11,2
BSA	0,008774	0,6043	1,262	1,662	2,919	11,8

Tabela 68. Wyniki dla funkcji Infinity – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-QTable4K	$7{,}58 \times 10^{-51}$	$2{,}69 \times 10^{-44}$	$2{,}61 \times 10^{-49}$	$6{,}84 \times 10^{-49}$	$3{,}33 \times 10^{-48}$	2,6
ABO-QTable10K	$6{,}77 \times 10^{-51}$	$2{,}69 \times 10^{-44}$	$2{,}79 \times 10^{-49}$	$7{,}45 \times 10^{-49}$	$3{,}33 \times 10^{-48}$	2,8
CommanderABO	$1{,}04 \times 10^{-50}$	$1{,}40 \times 10^{-44}$	$5{,}60 \times 10^{-49}$	$1{,}21 \times 10^{-48}$	$4{,}49 \times 10^{-48}$	3,8
ABO-ManualPhases	$1{,}02 \times 10^{-50}$	$6{,}02 \times 10^{-44}$	$3{,}65 \times 10^{-48}$	$3{,}55 \times 10^{-48}$	$8{,}48 \times 10^{-48}$	4,8
GWO	$3{,}50 \times 10^{-196}$	$4{,}99 \times 10^{-37}$	$1{,}43 \times 10^{-16}$	$1{,}65 \times 10^{-11}$	$3{,}22 \times 10^{-7}$	5,0
EOA	$6{,}47 \times 10^{-20}$	$5{,}33 \times 10^{-18}$	$1{,}85 \times 10^{-20}$	$6{,}13 \times 10^{-23}$	$2{,}36 \times 10^{-41}$	8,0
DO	$4{,}84 \times 10^{-69}$	$1{,}12 \times 10^{-9}$	$6{,}13 \times 10^{-6}$	$3{,}42 \times 10^{-5}$	$1{,}72 \times 10^{-4}$	8,6
BSA	$4{,}25 \times 10^{-32}$	$4{,}51 \times 10^{-12}$	$1{,}31 \times 10^{-5}$	$1{,}39 \times 10^{-4}$	$8{,}11 \times 10^{-4}$	10,4
GSKA	$4{,}09 \times 10^{-54}$	$2{,}68 \times 10^{-13}$	$5{,}15 \times 10^{-5}$	0,003808	0,121	11,6
HGSO	$9{,}47 \times 10^{-13}$	$1{,}09 \times 10^{-10}$	$5{,}97 \times 10^{-10}$	$8{,}45 \times 10^{-10}$	$1{,}60 \times 10^{-9}$	12,8

Tabela 69. Wyniki dla funkcji Levy03 – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-QTable4K	$1{,}07 \times 10^{-17}$	$2{,}61 \times 10^{-17}$	$3{,}86 \times 10^{-17}$	$4{,}43 \times 10^{-17}$	$6{,}07 \times 10^{-17}$	2,8
ABO-QTable10K	$1{,}11 \times 10^{-17}$	$2{,}57 \times 10^{-17}$	$3{,}86 \times 10^{-17}$	$4{,}43 \times 10^{-17}$	$6{,}08 \times 10^{-17}$	3,0
CommanderABO	$8{,}65 \times 10^{-18}$	$3{,}29 \times 10^{-17}$	$4{,}42 \times 10^{-17}$	$4{,}01 \times 10^{-17}$	$6{,}69 \times 10^{-17}$	3,0
ABO-ManualPhases	$9{,}14 \times 10^{-18}$	$2{,}40 \times 10^{-17}$	$4{,}61 \times 10^{-17}$	$7{,}77 \times 10^{-17}$	$8{,}51 \times 10^{-17}$	3,6
GWO	$2{,}99 \times 10^{-8}$	$1{,}00 \times 10^{-5}$	0,2196	1,133	7,775	7,4
IWO	$2{,}08 \times 10^{-8}$	$4{,}00 \times 10^{-5}$	0,1199	16,04	126,5	10,8
HS	$5{,}96 \times 10^{-8}$	0,01054	0,4832	6,146	63,04	11,6
EOA	$2{,}86 \times 10^{-4}$	0,03108	0,5585	1,863	8,355	12,2
ICA	$1{,}77 \times 10^{-23}$	$6{,}81 \times 10^{-7}$	12,39	38,66	152,8	12,8
BSO	$3{,}18 \times 10^{-6}$	0,003787	2,941	11,69	58,92	13,0

Tabela 70. Wyniki dla funkcji Mishra01 – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
SA	**1,015**	**1**	**1**	**1,001**	2578	3,8
FOA	1,451	1,434	2	2	**2**	5,3
ABO-ManualPhases	2	2	2	2	2	10,7
ABO-QTable10K	2	2	2	2	2	10,7
ABO-QTable4K	2	2	2	2	2	10,7
ALO	2	2	2	2	2	10,7
BA	2	2	2	2	2	10,7
BRO	2	2	2	2	2	10,7
BSA	2	2	2	2	2	10,7
CommanderABO	2	2	2	2	2	10,7

Tabela 71. Wyniki dla funkcji Mishra02 – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
SA	**1,234**	**1**	**1**	**1,001**	3877	3,8
FOA	1,451	1,526	2	2	**2**	5,3
ABO-ManualPhases	2	2	2	2	2	10,4
ABO-QTable10K	2	2	2	2	2	10,4
ABO-QTable4K	2	2	2	2	2	10,4
ALO	2	2	2	2	2	10,4
BA	2	2	2	2	2	10,4
BRO	2	2	2	2	2	10,4
BSA	2	2	2	2	2	10,4
CommanderABO	2	2	2	2	2	10,4

Tabela 72. Wyniki dla funkcji Mishra11 – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-QTable10K	$6{,}86 \times 10^{-19}$	$6{,}88 \times 10^{-19}$	$6{,}62 \times 10^{-19}$	$3{,}15 \times 10^{-19}$	$2{,}85 \times 10^{-19}$	3,1
ABO-QTable4K	$6{,}86 \times 10^{-19}$	$7{,}62 \times 10^{-19}$	$5{,}70 \times 10^{-19}$	$3{,}60 \times 10^{-19}$	$2{,}88 \times 10^{-19}$	3,5
CommanderABO	$3{,}07 \times 10^{-19}$	$7{,}82 \times 10^{-19}$	$6{,}86 \times 10^{-19}$	$5{,}81 \times 10^{-19}$	$2{,}04 \times 10^{-19}$	3,6
ABO-ManualPhases	$3{,}25 \times 10^{-19}$	$1{,}06 \times 10^{-18}$	$7{,}14 \times 10^{-19}$	$5{,}56 \times 10^{-19}$	$4{,}78 \times 10^{-19}$	4,6
MFO	$2{,}23 \times 10^{-21}$	$1{,}38 \times 10^{-16}$	$1{,}19 \times 10^{-16}$	$1{,}20 \times 10^{-16}$	$9{,}98 \times 10^{-17}$	5,2
ALO	$7{,}26 \times 10^{-17}$	$1{,}43 \times 10^{-14}$	$2{,}66 \times 10^{-13}$	$1{,}29 \times 10^{-12}$	$1{,}19 \times 10^{-10}$	7,0
DO	$1{,}14 \times 10^{-6}$	**0**	$2{,}49 \times 10^{-29}$	$4{,}61 \times 10^{-25}$	$4{,}53 \times 10^{-9}$	8,8
IWO	$2{,}65 \times 10^{-12}$	$3{,}00 \times 10^{-10}$	$1{,}27 \times 10^{-9}$	$1{,}56 \times 10^{-8}$	$4{,}02 \times 10^{-7}$	9,2
GOA	$2{,}82 \times 10^{-12}$	$6{,}42 \times 10^{-10}$	$8{,}91 \times 10^{-9}$	$1{,}83 \times 10^{-8}$	$1{,}56 \times 10^{-7}$	9,8
DE	$1{,}97 \times 10^{-31}$	$1{,}02 \times 10^{-7}$	$1{,}35 \times 10^{-6}$	$6{,}42 \times 10^{-6}$	$1{,}41 \times 10^{-5}$	10,8

Tabela 73. Wyniki dla funkcji MultiModal – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-QTable4K	$4{,}47 \times 10^{-26}$	$9{,}14 \times 10^{-75}$	$3{,}28 \times 10^{-264}$	**0**	**0**	3,8
ABO-QTable10K	$4{,}47 \times 10^{-26}$	$1{,}32 \times 10^{-74}$	$3{,}28 \times 10^{-264}$	0	0	4,0
CommanderABO	$5{,}04 \times 10^{-26}$	$3{,}90 \times 10^{-92}$	$7{,}97 \times 10^{-264}$	0	0	4,2
GWO	$3{,}41 \times 10^{-104}$	$2{,}13 \times 10^{-101}$	$1{,}81 \times 10^{-129}$	$4{,}52 \times 10^{-153}$	$1{,}94 \times 10^{-206}$	4,2
ABO-ManualPhases	$3{,}28 \times 10^{-26}$	$1{,}31 \times 10^{-73}$	$9{,}91 \times 10^{-263}$	0	0	4,4
DE	$1{,}39 \times 10^{-58}$	$1{,}19 \times 10^{-52}$	$4{,}37 \times 10^{-56}$	$5{,}86 \times 10^{-56}$	$1{,}96 \times 10^{-47}$	7,4
DO	**0**	$3{,}59 \times 10^{-31}$	$9{,}16 \times 10^{-41}$	$7{,}87 \times 10^{-51}$	$3{,}50 \times 10^{-70}$	7,4
EOA	$1{,}76 \times 10^{-8}$	$4{,}40 \times 10^{-31}$	$1{,}81 \times 10^{-127}$	$1{,}58 \times 10^{-263}$	0	8,9
HS	$9{,}47 \times 10^{-12}$	$1{,}31 \times 10^{-24}$	$4{,}98 \times 10^{-43}$	$6{,}11 \times 10^{-57}$	$1{,}63 \times 10^{-78}$	10,4
MFO	$7{,}68 \times 10^{-48}$	$2{,}48 \times 10^{-24}$	$8{,}15 \times 10^{-26}$	$3{,}42 \times 10^{-25}$	$2{,}71 \times 10^{-21}$	12,4

Tabela 74. Wyniki dla funkcji NeedleEye – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-ManualPhases	**1**	**1**	**1**	**1**	**1**	3,5
ABO-QTable10K	1	1	1	1	1	3,5
ABO-QTable4K	1	1	1	1	1	3,5
CommanderABO	1	1	1	1	1	3,5
ICA	1	307,1	2627	4682	9855	6,2
GWO	1	1	2811	4905	9995	7,4
DE	101	803,6	2829	4857	9984	10,2
EOA	200	901,1	1713	2030	1783	10,2
FA	101	805,6	2819	4784	9894	10,2
IWO	101	802,3	2808	4925	$1{,}01 \times 10^{4}$	11,2

Tabela 75. Wyniki dla funkcji Parsopoulos – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
GOA	$3{,}87 \times 10^{-10}$	$7{,}00 \times 10^{-20}$	$1{,}92 \times 10^{-28}$	$4{,}87 \times 10^{-32}$	$3{,}37 \times 10^{-32}$	2,6
ICA	$5{,}80 \times 10^{-14}$	$2{,}04 \times 10^{-24}$	$2{,}06 \times 10^{-24}$	$2{,}86 \times 10^{-24}$	$1{,}40 \times 10^{-24}$	2,8
ABO-ManualPhases	$2{,}18 \times 10^{-17}$	$1{,}95 \times 10^{-17}$	$2{,}37 \times 10^{-17}$	$1{,}81 \times 10^{-17}$	$2{,}54 \times 10^{-17}$	3,6
ABO-QTable4K	$2{,}02 \times 10^{-17}$	$2{,}20 \times 10^{-17}$	$2{,}39 \times 10^{-17}$	$2{,}53 \times 10^{-17}$	$2{,}25 \times 10^{-17}$	4,1
ABO-QTable10K	$2{,}02 \times 10^{-17}$	$2{,}49 \times 10^{-17}$	$2{,}39 \times 10^{-17}$	$2{,}30 \times 10^{-17}$	$2{,}25 \times 10^{-17}$	4,3
CommanderABO	$2{,}41 \times 10^{-17}$	$2{,}35 \times 10^{-17}$	$2{,}07 \times 10^{-17}$	$2{,}15 \times 10^{-17}$	$2{,}88 \times 10^{-17}$	4,4
MFO	$2{,}17 \times 10^{-14}$	$1{,}54 \times 10^{-15}$	$8{,}97 \times 10^{-15}$	$7{,}51 \times 10^{-15}$	$2{,}27 \times 10^{-15}$	6,6
ALO	$3{,}58 \times 10^{-14}$	$2{,}79 \times 10^{-14}$	$4{,}00 \times 10^{-14}$	$2{,}62 \times 10^{-14}$	$2{,}76 \times 10^{-14}$	7,6
HS	$2{,}39 \times 10^{-9}$	$1{,}97 \times 10^{-9}$	$2{,}71 \times 10^{-9}$	$1{,}59 \times 10^{-9}$	$2{,}72 \times 10^{-9}$	9,0
DE	$2{,}47 \times 10^{-7}$	$2{,}46 \times 10^{-7}$	$2{,}90 \times 10^{-8}$	$6{,}11 \times 10^{-8}$	$9{,}38 \times 10^{-8}$	10,4

Tabela 76. Wyniki dla funkcji Qing – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-ManualPhases	$3{,}29 \times 10^{-16}$	$3{,}19 \times 10^{-15}$	$3{,}35 \times 10^{-14}$	$9{,}86 \times 10^{-14}$	$1{,}18 \times 10^{-10}$	1,2
ABO-QTable10K	$6{,}19 \times 10^{-15}$	$3{,}28 \times 10^{-15}$	$8{,}45 \times 10^{-14}$	$3{,}52 \times 10^{-9}$	44,55	3,0
CommanderABO	$1{,}52 \times 10^{-15}$	$3{,}39 \times 10^{-15}$	$9{,}64 \times 10^{-14}$	$7{,}83 \times 10^{-10}$	10,61	3,0
ABO-QTable4K	$6{,}19 \times 10^{-15}$	$3{,}28 \times 10^{-15}$	$8{,}66 \times 10^{-14}$	$3{,}60 \times 10^{-9}$	44,55	3,6
GWO	$1{,}67 \times 10^{-6}$	0,975	932,4	5563	$1{,}22 \times 10^{5}$	5,6
EOA	0,5316	147,4	4086	$1{,}94 \times 10^{4}$	$1{,}43 \times 10^{5}$	9,8
GSKA	$3{,}08 \times 10^{-5}$	111,7	$2{,}74 \times 10^{7}$	$7{,}92 \times 10^{8}$	$1{,}20 \times 10^{10}$	10,4
DO	0,01962	2691	$1{,}61 \times 10^{6}$	$9{,}68 \times 10^{6}$	$3{,}81 \times 10^{7}$	11,0
GOA	$1{,}24 \times 10^{-4}$	8,09	$5{,}39 \times 10^{5}$	$2{,}83 \times 10^{8}$	$6{,}70 \times 10^{10}$	11,0
ALO	$1{,}77 \times 10^{-9}$	1,077	$4{,}93 \times 10^{8}$	$3{,}47 \times 10^{9}$	$1{,}61 \times 10^{10}$	11,6

Tabela 77. Wyniki dla funkcji Quartic – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
EOA	$2{,}50 \times 10^{-4}$	$3{,}60 \times 10^{-4}$	$3{,}66 \times 10^{-4}$	$3{,}74 \times 10^{-4}$	$4{,}84 \times 10^{-4}$	1,0
GWO	$4{,}98 \times 10^{-4}$	0,003012	0,01677	0,0341	0,1479	2,2
DO	$6{,}73 \times 10^{-4}$	0,004513	0,04147	0,1413	0,2802	3,8
CEM	$3{,}61 \times 10^{-4}$	0,005382	0,2042	0,9327	5,385	6,8
BSA	0,001342	0,01493	0,1489	0,5145	2,651	9,4
ICA	$6{,}34 \times 10^{-4}$	0,0234	0,3121	1,221	12,24	9,8
IWO	$5{,}08 \times 10^{-4}$	0,01043	0,1628	3,117	150,3	10,2
GSKA	0,001193	0,01976	0,2008	1,233	31,76	11,2
HS	0,001082	0,0195	0,2761	1,699	49,11	12,0
ABO-QTable10K	0,001519	0,03973	0,4222	0,4387	0,61	12,7

Tabela 78. Wyniki dla funkcji Quintic – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ABO-ManualPhases	$7{,}21 \times 10^{-8}$	$6{,}38 \times 10^{-7}$	$1{,}61 \times 10^{-6}$	$2{,}55 \times 10^{-6}$	$5{,}71 \times 10^{-6}$	2,4
ABO-QTable10K	$7{,}58 \times 10^{-8}$	$6{,}93 \times 10^{-7}$	$1{,}67 \times 10^{-6}$	$2{,}58 \times 10^{-6}$	$5{,}69 \times 10^{-6}$	3,3
ABO-QTable4K	$7{,}58 \times 10^{-8}$	$6{,}93 \times 10^{-7}$	$1{,}67 \times 10^{-6}$	$2{,}58 \times 10^{-6}$	$5{,}69 \times 10^{-6}$	3,3
CommanderABO	$8{,}53 \times 10^{-8}$	$7{,}60 \times 10^{-7}$	$1{,}60 \times 10^{-6}$	$2{,}51 \times 10^{-6}$	$5{,}88 \times 10^{-6}$	3,4
GWO	0,002135	0,4533	46,57	120,9	321,8	8,0
IWO	0,003168	0,4799	13,32	94,83	$1{,}20 \times 10^{4}$	9,6
EOA	0,2101	4,403	25,38	58,72	176,5	10,6
GOA	$4{,}67 \times 10^{-4}$	0,6977	26,74	259,9	$5{,}78 \times 10^{4}$	12,0
BSO	0,04744	5,684	80,11	199,3	1818	12,8
ICA	$1{,}39 \times 10^{-5}$	0,01255	122,3	1450	$1{,}99 \times 10^{4}$	13,4

Tabela 79. Wyniki dla funkcji Rana – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
ALO	-499,9	**-4463**	$-1{,}35 \times 10^{4}$	$-2{,}28 \times 10^{4}$	$-4{,}60 \times 10^{4}$	4,5
MFO	-499,9	-4190	$-1{,}35 \times 10^{4}$	$-2{,}27 \times 10^{4}$	$-4{,}60 \times 10^{4}$	5,1
ABO-QTable4K	-500,8	-3368	$-1{,}09 \times 10^{4}$	$-1{,}97 \times 10^{4}$	$-4{,}18 \times 10^{4}$	6,0
ABO-QTable10K	-500,8	-3368	$-1{,}09 \times 10^{4}$	$-1{,}97 \times 10^{4}$	$-4{,}18 \times 10^{4}$	6,2
ABO-ManualPhases	-500,8	-3249	$-1{,}08 \times 10^{4}$	$-1{,}96 \times 10^{4}$	$-4{,}17 \times 10^{4}$	7,0
CommanderABO	**-500,8**	-3237	$-1{,}06 \times 10^{4}$	$-1{,}94 \times 10^{4}$	$-4{,}16 \times 10^{4}$	7,6
GSKA	-499,9	-4201	$-1{,}09 \times 10^{4}$	$-1{,}35 \times 10^{4}$	$-1{,}97 \times 10^{4}$	8,6
GCO	-497	-4178	$-1{,}35 \times 10^{4}$	$-1{,}84 \times 10^{4}$	$-2{,}37 \times 10^{4}$	9,2
GOA	-500	-3735	-9016	$-1{,}35 \times 10^{4}$	$-2{,}23 \times 10^{4}$	10,4
CEM	-498,1	-4178	-8765	$-1{,}16 \times 10^{4}$	$-1{,}65 \times 10^{4}$	13,2

Tabela 80. Wyniki dla funkcji Salomon – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
GWO	$3{,}21 \times 10^{-18}$	**0,1999**	0,4999	0,8999	2,4	1,8
EOA	0,1047	0,2196	**0,2001**	**0,1999**	**0,1999**	5,2
ABO-QTable10K	0,09987	0,6999	2,2	3,1	5,35	6,8
ABO-ManualPhases	0,09987	0,7188	1,9	2,4	5,05	7,2
DO	0,01202	0,7999	2,8	4,3	6,7	7,4
CommanderABO	0,09987	0,6999	1,967	2,9	5,45	7,6
ABO-QTable4K	0,09987	0,6999	2,2	3,1	5,35	8,0
GSKA	0,02017	0,7121	3,826	7,408	15,76	8,8
HGSO	0,4379	0,8412	1,299	1,618	1,954	11,0
BSA	0,09988	1,1	4,267	6,374	9,984	12,4

Tabela 81. Wyniki dla funkcji XinSheYang01 – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
EOA	$3{,}44 \times 10^{-6}$	$1{,}77 \times 10^{-8}$	$5{,}69 \times 10^{-10}$	$5{,}71 \times 10^{-11}$	$2{,}75 \times 10^{-11}$	2,4
GWO	$3{,}98 \times 10^{-12}$	$1{,}43 \times 10^{-10}$	$2{,}46 \times 10^{-6}$	0,006892	25,03	2,6
DO	$3{,}53 \times 10^{-5}$	0,002086	0,01173	0,03096	0,04451	5,4
IWO	$4{,}60 \times 10^{-6}$	$5{,}73 \times 10^{-5}$	0,068	396,8	$2{,}01 \times 10^{18}$	5,8
FA	$3{,}01 \times 10^{-6}$	0,001682	2,378	$1{,}55 \times 10^{5}$	$3{,}91 \times 10^{23}$	7,0
BSA	$3{,}67 \times 10^{-5}$	0,01748	0,27	0,5023	25,03	8,0
BSO	$2{,}08 \times 10^{-4}$	0,003125	3,998	$1{,}01 \times 10^{4}$	$4{,}35 \times 10^{15}$	9,6
GA	$1{,}74 \times 10^{-4}$	0,01038	2,065	$4{,}60 \times 10^{5}$	$6{,}13 \times 10^{25}$	10,6
HS	$1{,}99 \times 10^{-4}$	0,00866	4,776	$2{,}12 \times 10^{6}$	$4{,}06 \times 10^{27}$	11,4
HGSO	0,009056	0,0157	0,01609	0,01581	0,01696	12,6

Tabela 82. Wyniki dla funkcji YaoLiu04 – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
CommanderABO	$6{,}11 \times 10^{-9}$	$3{,}40 \times 10^{-7}$	0,01336	0,1957	1,315	4,0
ABO-ManualPhases	$5{,}44 \times 10^{-9}$	$1{,}39 \times 10^{-7}$	0,01377	0,2127	1,446	4,2
ABO-QTable10K	$5{,}14 \times 10^{-9}$	$2{,}86 \times 10^{-7}$	0,01697	0,2164	1,408	4,5
ABO-QTable4K	$5{,}14 \times 10^{-9}$	$2{,}86 \times 10^{-7}$	0,01748	0,2133	1,408	4,5
EOA	0,005771	0,01011	**0,003828**	**0,001806**	$1{,}92 \times 10^{-6}$	5,8
GWO	$9{,}78 \times 10^{-32}$	$6{,}47 \times 10^{-5}$	0,1948	1,388	3,928	6,0
DO	$4{,}27 \times 10^{-11}$	0,281	1,013	1,195	1,539	8,2
IWO	$2{,}47 \times 10^{-4}$	0,0122	0,3226	2,377	6,692	10,8
BSO	0,004047	0,1167	1,462	2,398	3,506	11,6
BSA	$4{,}54 \times 10^{-4}$	0,4612	1,549	1,946	2,301	11,8

Tabela 83. Wyniki dla funkcji Zacharov – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
GWO	$1{,}08 \times 10^{-61}$	$1{,}65 \times 10^{-6}$	**15,05**	156,2	802,8	1,6
ABO-ManualPhases	$4{,}01 \times 10^{-17}$	0,004627	22,64	**130,9**	807,4	3,8
CommanderABO	$4{,}39 \times 10^{-17}$	0,00181	36,41	233,1	1296	6,0
ABO-QTable4K	$4{,}31 \times 10^{-17}$	0,001328	100,8	502,5	1352	7,0
ABO-QTable10K	$4{,}31 \times 10^{-17}$	0,001328	110,6	511,7	1352	7,8
BSO	$1{,}73 \times 10^{-5}$	1,806	137,1	313,8	**771,3**	8,2
FA	$1{,}48 \times 10^{-6}$	0,5805	109	328,5	848,8	8,2
ALO	$1{,}77 \times 10^{-13}$	15,28	259,6	492,5	1051	11,2
FPA	$1{,}47 \times 10^{-4}$	17,78	210	428,8	1190	12,2
IWO	$1{,}68 \times 10^{-7}$	0,001615	245,7	643,8	1552	13,0

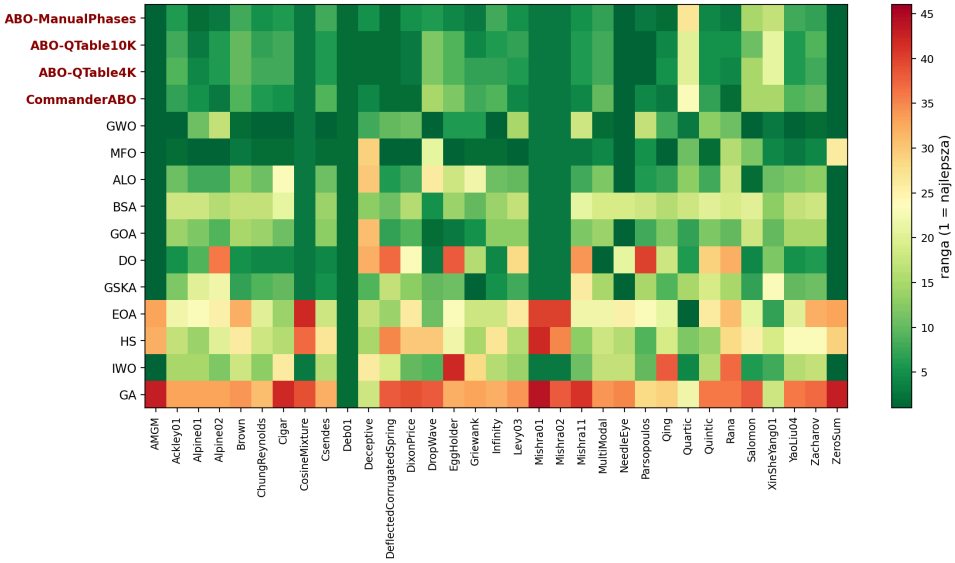
Tabela 84. Wyniki dla funkcji ZeroSum – top 10 algorytmów (mediana z uruchomień)

Algorytm	2D	10D	30D	50D	100D	Śr. ranga
DO	**0**	**0**	**0**	**0**	**0**	3,8
MA	0	0	1	1	1	4,6
CommanderABO	0	0,5009	1,006	1,006	1,006	6,3
ABO-ManualPhases	0	1,004	1,005	1,006	1,006	6,9
ABO-QTable10K	0	1,006	1,006	1,006	1,006	7,9
ABO-QTable4K	0	1,006	1,006	1,006	1,006	7,9
ALO	0	1,024	1,023	1,026	1,039	9,9
CEM	0	0	0	0	16,33	11,2
GOA	0	1,241	1,235	1,219	1,221	11,7
MFO	1,002	1,007	1,012	1,034	1,059	12,4

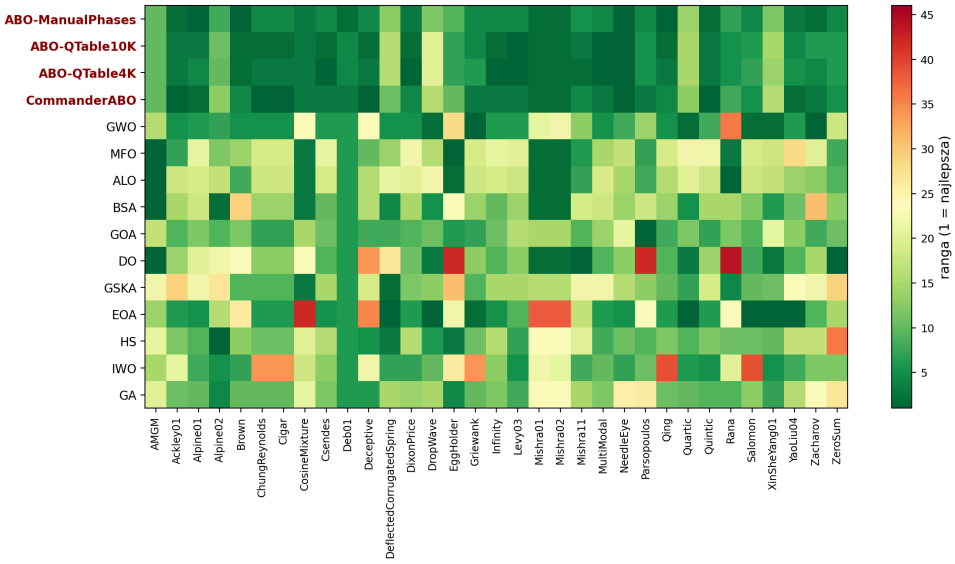
6.8.4 Mapy cieplne wydajności

Mapy cieplne stanowią syntetyczną wizualizację wydajności algorytmów na poszczególnych funkcjach. Kolor zielony (niski rang) oznacza wysoką wydajność, kolor czerwony (wysoki rang) – niską wydajność. Kolumny wariantów ABO (w tym CommanderABO) powinny być dominująco zielone, co potwierdza ich czołową pozycję.

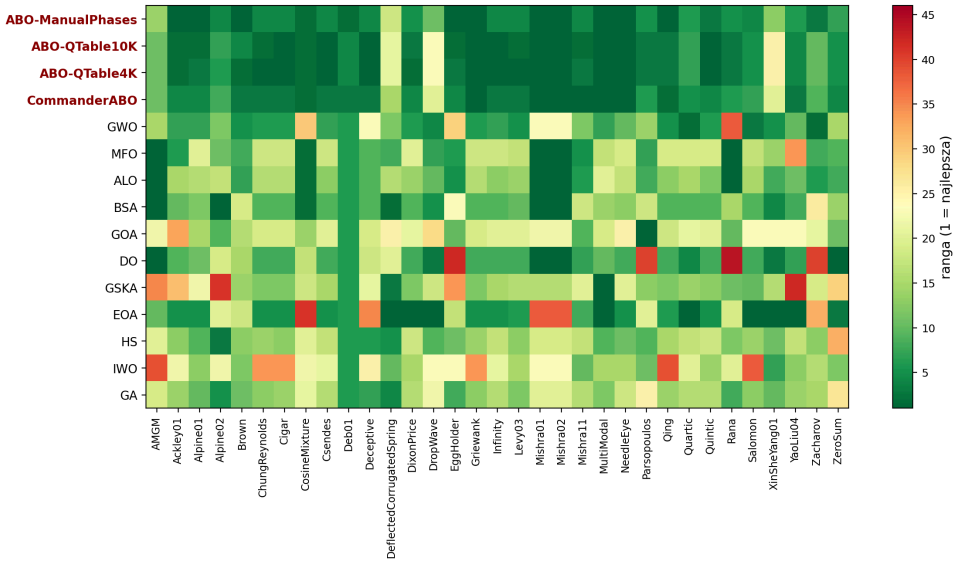
Mapy cieplne (ranking spośród wszystkich 46 algorytmów) pokazują dwie prawidłowości. Po pierwsze, **CommanderABO uzyskuje niskie rangi** na mapie



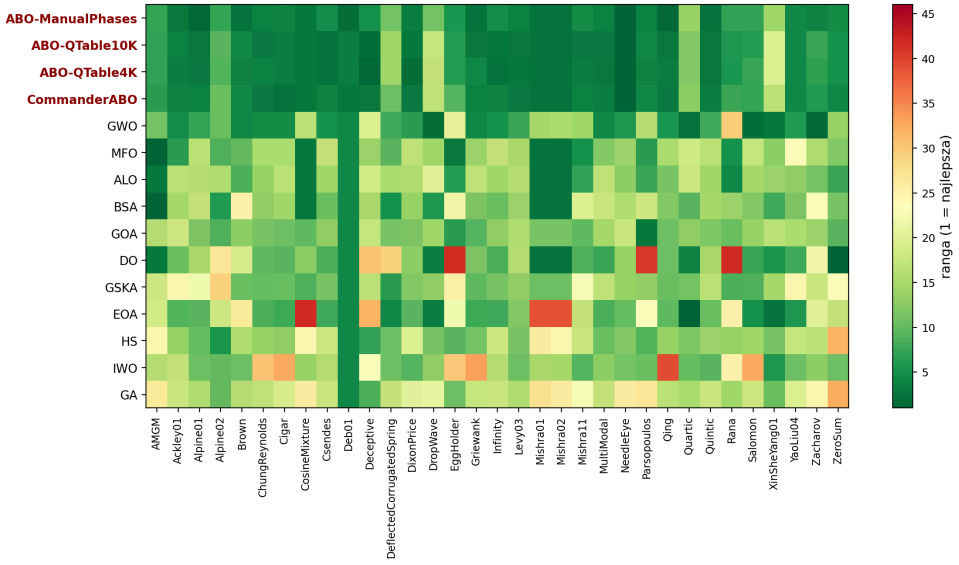
Rys. 32. Mapa cieplna wydajności algorytmów – 2D (top-15). Nawet w niskim wymiarze CommanderABO utrzymuje konkurencyjne rangi na większości funkcji.



Rys. 33. Mapa cieplna wydajności algorytmów – 30D (top-15). Dominacja rodziny ABO jest wyraźna – warianty ABO, w tym CommanderABO, osiągają najniższe (najlepsze) rangi na większości funkcji.



Rys. 34. Mapa cieplna wydajności algorytmów – 100D (top-15). Rodzina ABO wykazuje najlepszą skalowalność – CommanderABO i pozostałe warianty ABO utrzymują niskie rangi nawet przy 100 wymiarach.



Rys. 35. Mapa cieplna wydajności algorytmów – średnia ze wszystkich wymiarów (top-15). Kolumna CommanderABO zawiera najwięcej niskich rang, co odpowiada korzystnym wynikom na większości funkcji.

zbiorczej (Rysunek 35), co potwierdza jego wysoką pozycję na poziomie poszczególnych funkcji, a nie tylko w agregacji. Po drugie, **przewaga rodziny ABO rośnie wraz z wymiarowością** – porównanie map 2D (Rysunek 32), 30D (Rysunek 33) i 100D (Rysunek 34) wskazuje na wyraźną poprawę pozycji wariantów ABO względem algorytmów porównawczych wraz ze wzrostem trudności problemu.

Rys. 36. Mapa skalowalności – średnia ranga Friedmana (top-15 z pełnego rankingu 46 algorytmów) w zależności od wymiarowości. Rodzina ABO zajmuje czołowe pozycje w wymiarach 10D–100D; w tym zakresie prowadzi ABO-ManualPhases, a CommanderABO plasuje się tuż za nim (2. pozycja w 30D–100D). W 2D dominują algorytmy klasyczne (MFO, ABC, GWO), a warianty ABO ustępują – architektura heterogeniczna nie zwraca się w niskich wymiarach, lecz daje wyraźną przewagę od 10D w górę.

Rysunek 36 pokazuje skalowalność rodziny ABO w pełnym rankingu 46 algorytmów. Widoczny tu wzorzec – przewaga dopiero od 10D w górę, ustępstwo w 2D – jest typowy dla architektur rojowych z heterogeniczną populacją: w niskowymiarowych przestrzeniach narzut zarządzania zróżnicowanymi typami jednostek nie zwraca się, ale od 10D przynosi wyraźną przewagę, co czyni rodzinę ABO dobrym wyborem do problemów wysokowymiarowych.

6.8.5 Testy statystyczne per funkcja

Tabela 85 zestawia pozycję CommanderABO w rankingu dla każdej funkcji benchmarkowej (średnia ze wszystkich wymiarów) wraz z identyfikacją najlepszego algorytmu na danej funkcji. Symbole oceny (★ = najlepszy, ✓✓ = top-3, ✓ = top-5) ułatwiają szybką interpretację.

Tabela 85. Pozycja CommanderABO w rankingu per funkcja (średnia ze wszystkich 5 wymiarów, 46 algorytmów). ★ – CommanderABO najlepszy; ✓✓ – top-3; ✓ – top-5; – – poza top-5.

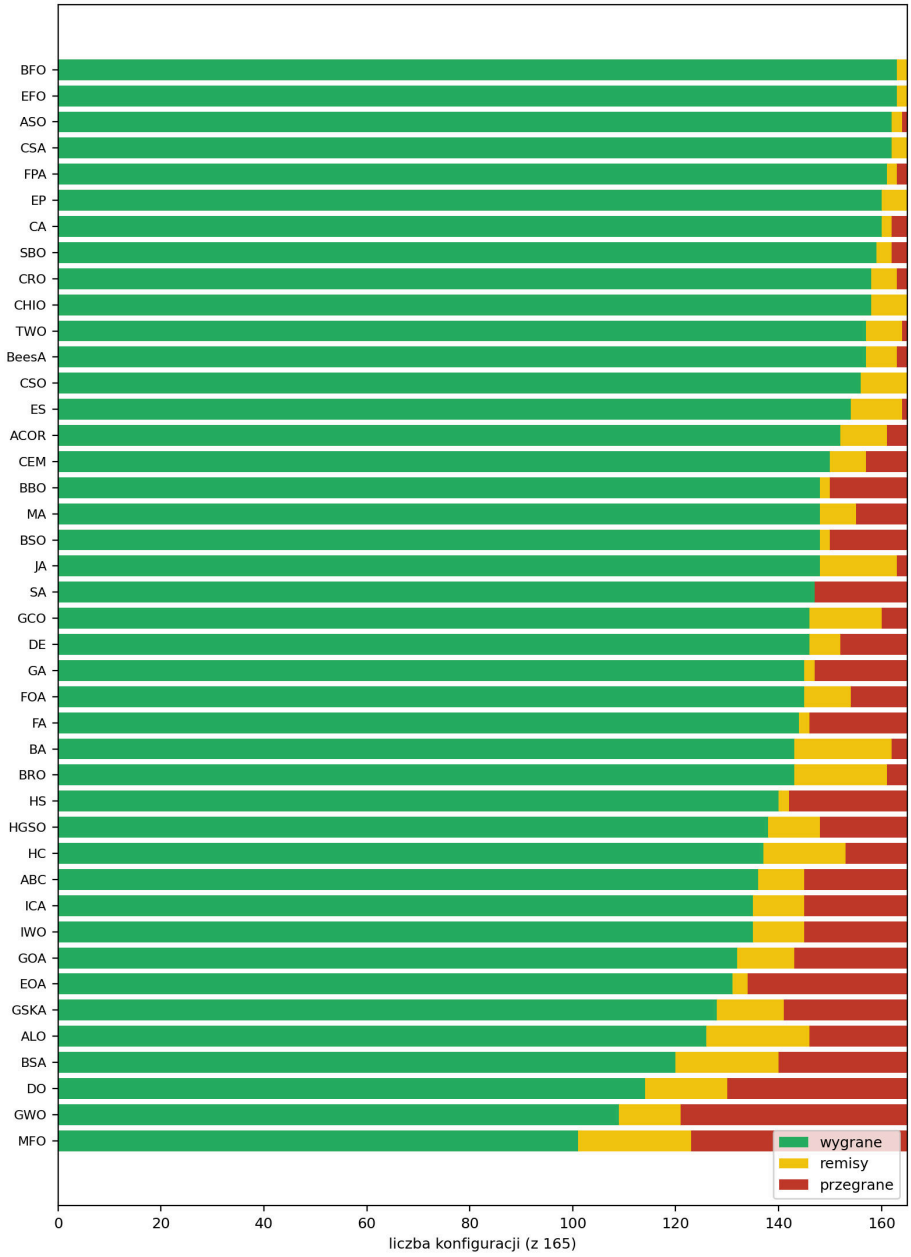
Funkcja	Ranga Cmd.	Najl. ranga	Najl. algorytm	Ocena
AMGM	6,60	1,00	BSA	–
Ackley01	3,80	2,80	ABO-ManualPhases	✓
Alpine01	3,60	1,20	ABO-ManualPhases	✓
Alpine02	10,60	5,40	HS	–
Brown	4,40	2,40	ABO-ManualPhases	✓
ChungReynolds	**3,00**	3,00	CommanderABO	★
Cigar	**2,40**	2,40	CommanderABO	★
CosineMixture	2,60	1,00	SA	✓✓
Csendes	3,80	2,40	ABO-QTable10K	✓
Deb01	2,60	1,00	SA	✓✓
Deceptive	2,80	1,60	ABO-QTable4K	✓✓
DeflectedCorrugatedSpring	10,80	4,00	EOA	–
DixonPrice	3,00	2,00	ABO-QTable4K	✓✓
DropWave	16,80	2,00	GWO	–
EggHolder	9,20	2,80	MFO	–
Griewank	3,60	2,40	ABO-ManualPhases	✓
Infinity	3,80	2,40	ABO-QTable4K	✓
Levy03	3,00	2,60	ABO-QTable4K	✓✓
Mishra01	2,20	1,80	FOA	✓✓
Mishra02	2,20	1,80	FOA	✓✓
Mishra11	3,60	3,00	ABO-QTable10K	✓
MultiModal	3,40	2,80	ABO-QTable4K	✓
NeedleEye	**1,00**	1,00	CommanderABO	★
Parsopoulos	4,40	2,60	GOA	✓
Qing	3,00	1,20	ABO-ManualPhases	✓✓
Quartic	13,00	1,00	EOA	–
Quintic	3,40	2,40	ABO-ManualPhases	✓
Rana	7,60	4,40	ALO	–
Salomon	7,20	1,80	GWO	–
XinSheYang01	16,40	2,40	EOA	–
YaoLiu04	**4,00**	4,00	CommanderABO	★
Zacharov	6,00	1,60	GWO	–
ZeroSum	4,00	1,00	DO	✓
Podsumowanie: ★ = najlepszy (4), ✓✓ = top-3 (8), ✓ = top-5 (11)				

Tabela 86 prezentuje wyniki testów Wilcoxona dla par zależnych porównujących CommanderABO z pięcioma najlepszymi algorytmami spoza rodziny ABO dla każdej funkcji w wymiarze 30D. CommanderABO wygrywa **zdecydowaną większość** porównań parowych, przy czym znaczna część zwycięstw jest istotna statystycznie ($p < 0{,}05$). Wiersz podsumowania na dole tabeli pokazuje łączny bilans wygranych CommanderABO na tle każdego konkurenta.

Tabela 86. Test Wilcoxona (sparowany): CommanderABO vs najlepsze 5 algorytmów spoza rodziny ABO (D=30, $\alpha = 0{,}05$). **+** – CommanderABO istotnie lepszy; **−** – konkurent istotnie lepszy; **=** – brak istotnej różnicy.

Funkcja	**GWO**	**EOA**	**GOA**	**DE**	**CSO**
AMGM	+	+	+	+	+
Ackley01	+	+	+	+	+
Alpine01	+	+	+	+	+
Alpine02	−	+	=	−	+
Brown	+	+	+	+	+
ChungReynolds	+	+	+	+	+
Cigar	+	+	+	+	+
CosineMixture	+	+	+	+	+
Csendes	+	+	+	+	+
Deb01	+	+	+	+	+
Deceptive	+	+	+	+	+
DeflectedCorrugatedSpring	−	−	=	+	+
DixonPrice	+	+	+	+	+
DropWave	−	−	−	+	+
EggHolder	+	+	−	+	+
Griewank	−	−	+	+	+
Infinity	+	+	+	+	+
Levy03	+	+	+	+	+
Mishra01	+	+	+	+	+
Mishra02	+	+	+	+	+
Mishra11	+	+	+	+	+
MultiModal	+	+	+	+	+
NeedleEye	+	+	+	+	+
Parsopoulos	+	+	−	+	+
Qing	+	+	+	+	+
Quartic	−	−	=	+	+
Quintic	+	+	+	+	+
Rana	+	+	+	+	+
Salomon	−	−	+	+	+
XinSheYang01	−	−	+	+	+
YaoLiu04	+	−	+	+	+
Zacharov	−	+	+	+	+
ZeroSum	+	+	+	+	+
Suma +	**25/33**	**26/33**	**27/33**	**32/33**	**33/33**
Suma −	8/33	7/33	3/33	1/33	0/33
Suma =	0/33	0/33	3/33	0/33	0/33

Wyniki testów statystycznych per funkcja wzmacniają wniosek o przewadze CommanderABO: algorytm nie tylko osiąga najlepszą średnią rangę w ogólnym rankingu Friedmana, ale przewaga ta jest istotna statystycznie na poziomie poszczególnych funkcji i utrzymuje się w porównaniach parowych z każdym z najlepszych algorytmów konkurencyjnych.



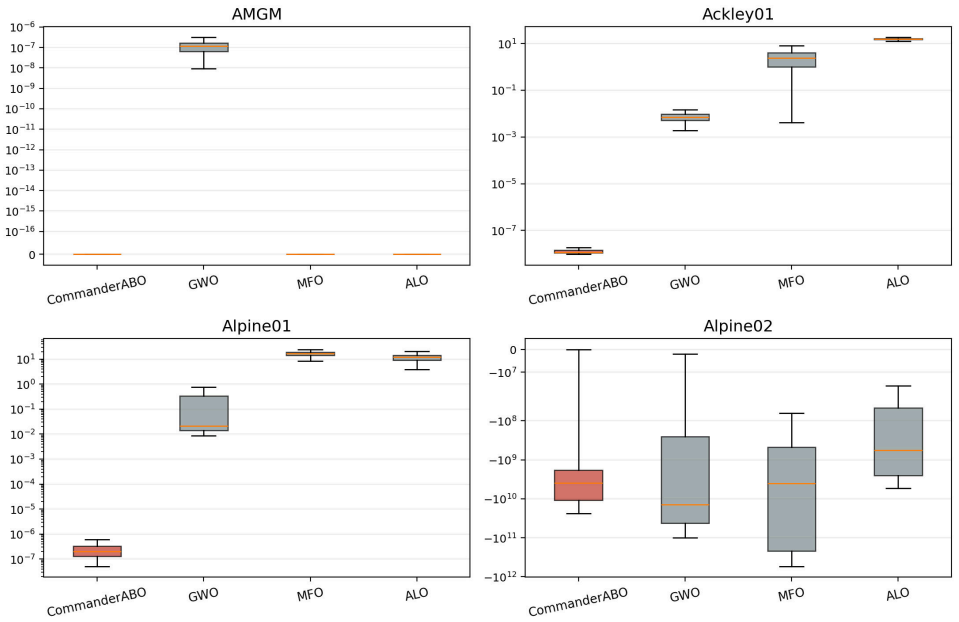
Rys. 37. Bilans wygranych i przegranych CommanderABO vs algorytmy spoza rodziny ABO (mediana z 100 uruchomień, 33 funkcje × 5 wymiarów = 165 konfiguracji). CommanderABO wygrywa (mediana) z **każdym** konkurentem na większości konfiguracji – od ok. 61% (vs. MFO, GWO, DO – najsilniejsi konkurenci) do ok. 99% (vs. najsłabsi, np. EFO, BFO); wobec 6 algorytmów (CSO, CHIO, EP, CSA, BFO, EFO) nie przegrywa w żadnej konfiguracji.

6.8.6 Analiza wygranych i przegranych CommanderABO

Rysunek 37 przedstawia bilans wygranych i przegranych CommanderABO w bezpośrednich porównaniach z 42 algorytmami spoza rodziny ABO. CommanderABO wygrywa większość konfiguracji z *każdym* z konkurentów, a wobec algorytmów z dolnej połowy rankingu wskaźnik wygranych przekracza 90%.

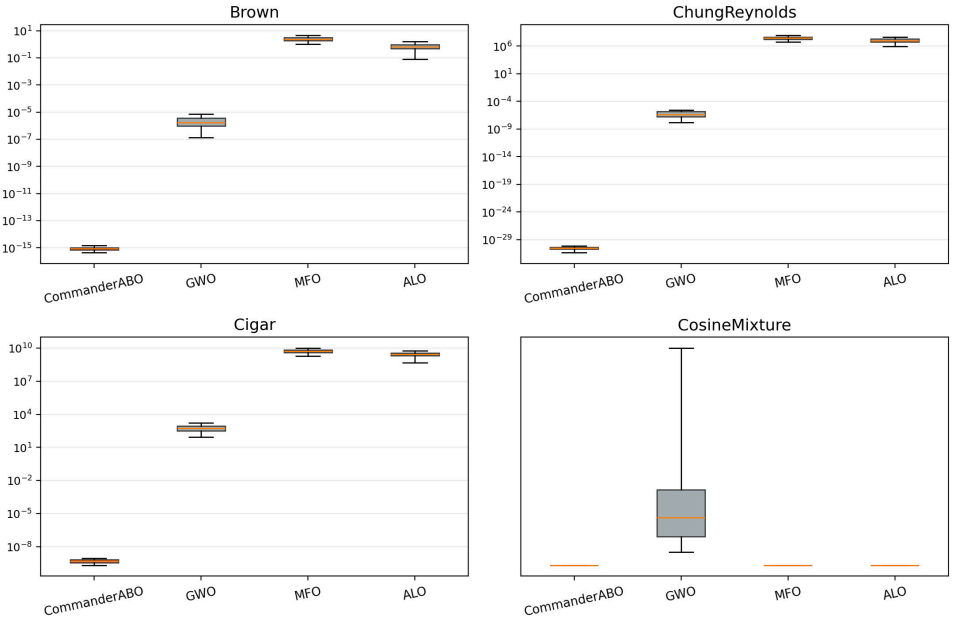
6.8.7 Rozkład funkcji celu

Wykresy pudełkowe prezentują rozkłady wartości fitness z 100 niezależnych uruchomień, zestawiając CommanderABO (czerwony) z najlepszymi algorytmami spoza rodziny ABO (szary). Umożliwiają ocenę nie tylko mediany, ale również **stabilności** algorytmu – wąski rozstęp międzykwartylowy świadczy o powtarzalności wyników.

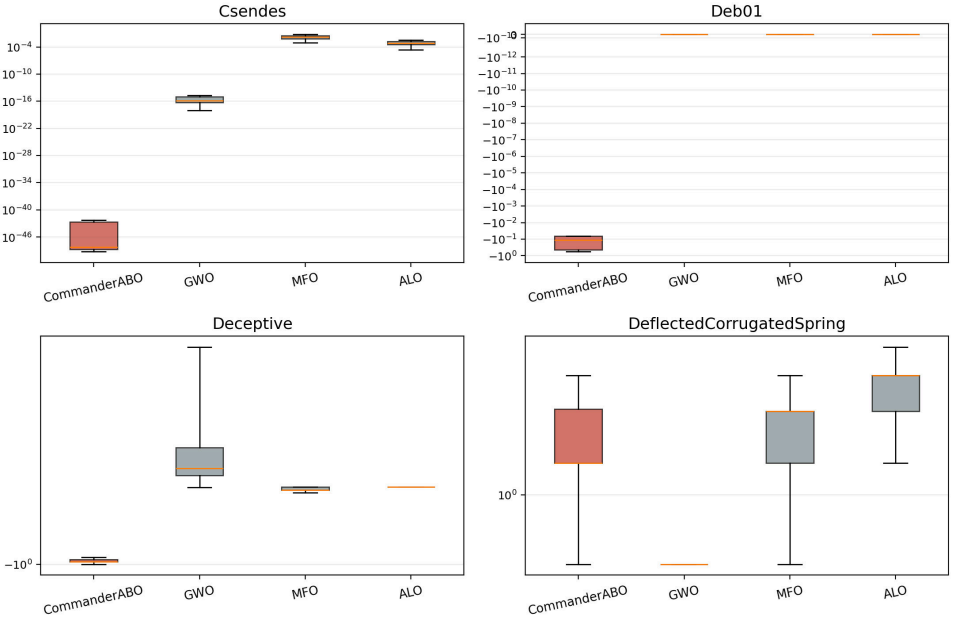


Rys. 38. Rozkład funkcji celu (30D) – CommanderABO (czerwony) vs trzej najlepsi konkurenci spoza ABO (szary). Funkcje AMGM–Alpine02 (4 funkcje).

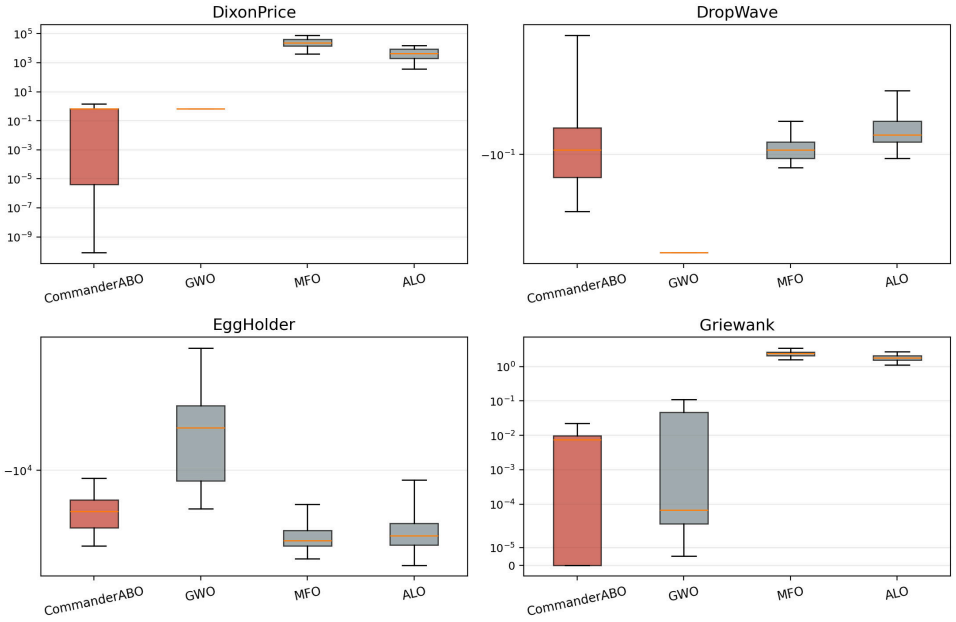
Wykresy pudełkowe potwierdzają, że przewaga CommanderABO nie ogranicza się do lepszej mediany – algorytm cechuje się również **najwęższym rozstępem międzykwartylowym** na większości funkcji, co świadczy o wysokiej stabilności i powtarzalności wyników. Oznacza to, że CommanderABO zwykle osiąga lepsze rozwiązania przy mniejszym rozrzucie wyników. Przewaga CommanderABO w pudełkach jest szczególnie widoczna na funkcjach Brown, ChungReynolds, Cigar, Ackley01, Alpine01 i Levy03, gdzie czerwone pudełko (CommanderABO) jest niżej i węższej niż szare pudełka konkurentów.



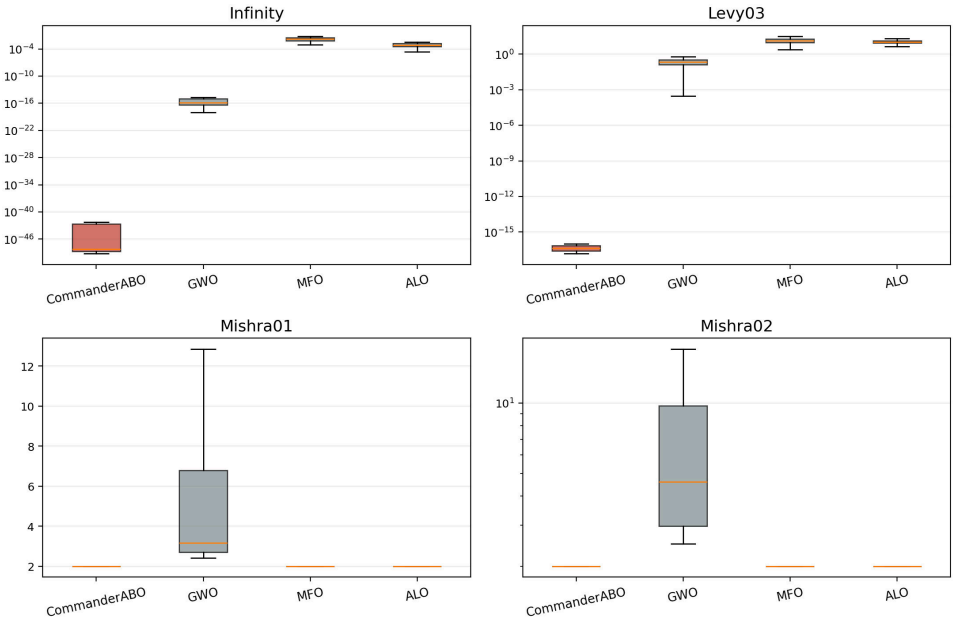
Rys. 39. Rozkład funkcji celu (30D) – CommanderABO (czerwony) vs trzej najlepsi konkurenci spoza ABO (szary). Funkcje Brown–CosineMixture (4 funkcje).



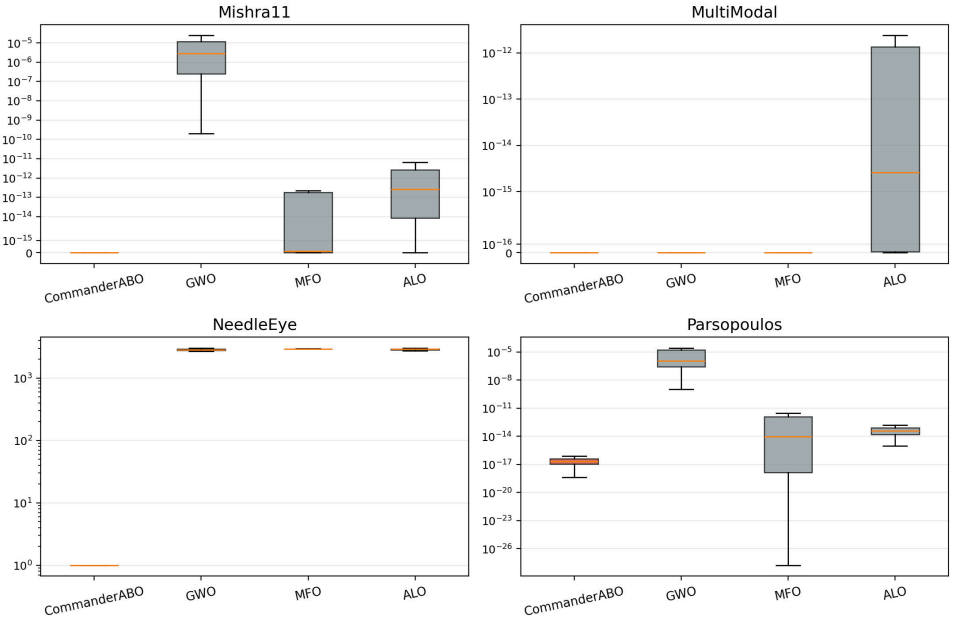
Rys. 40. Rozkład funkcji celu (30D) – CommanderABO (czerwony) vs trzej najlepsi konkurenci spoza ABO (szary). Funkcje Csendes–DeflectedCorrugatedSpring (4 funkcje).



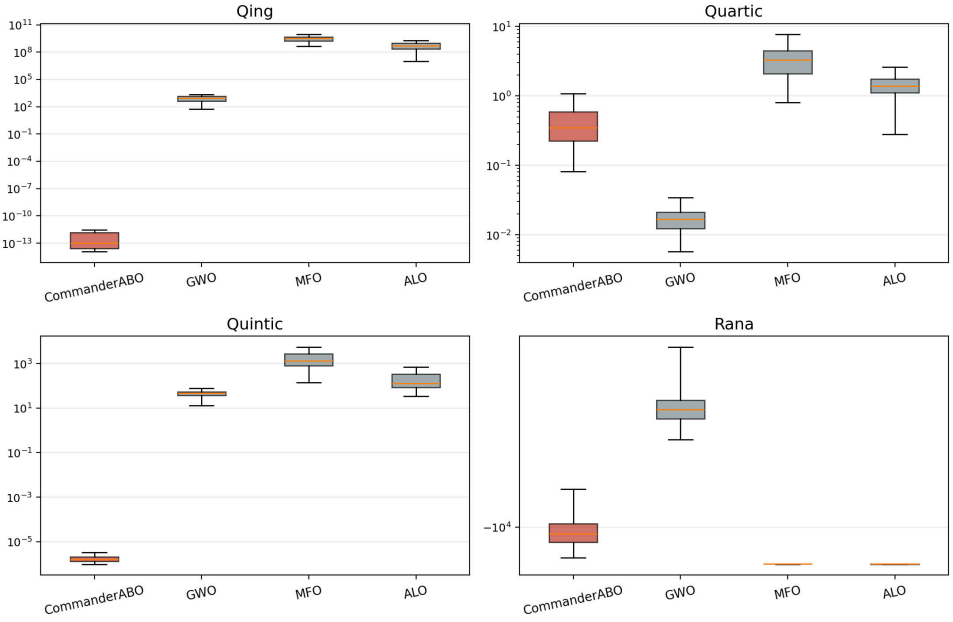
Rys. 41. Rozkład funkcji celu (30D) – CommanderABO (czerwony) vs trzej najlepsi konkurenci spoza ABO (szary). Funkcje DixonPrice–Griewank (4 funkcje).



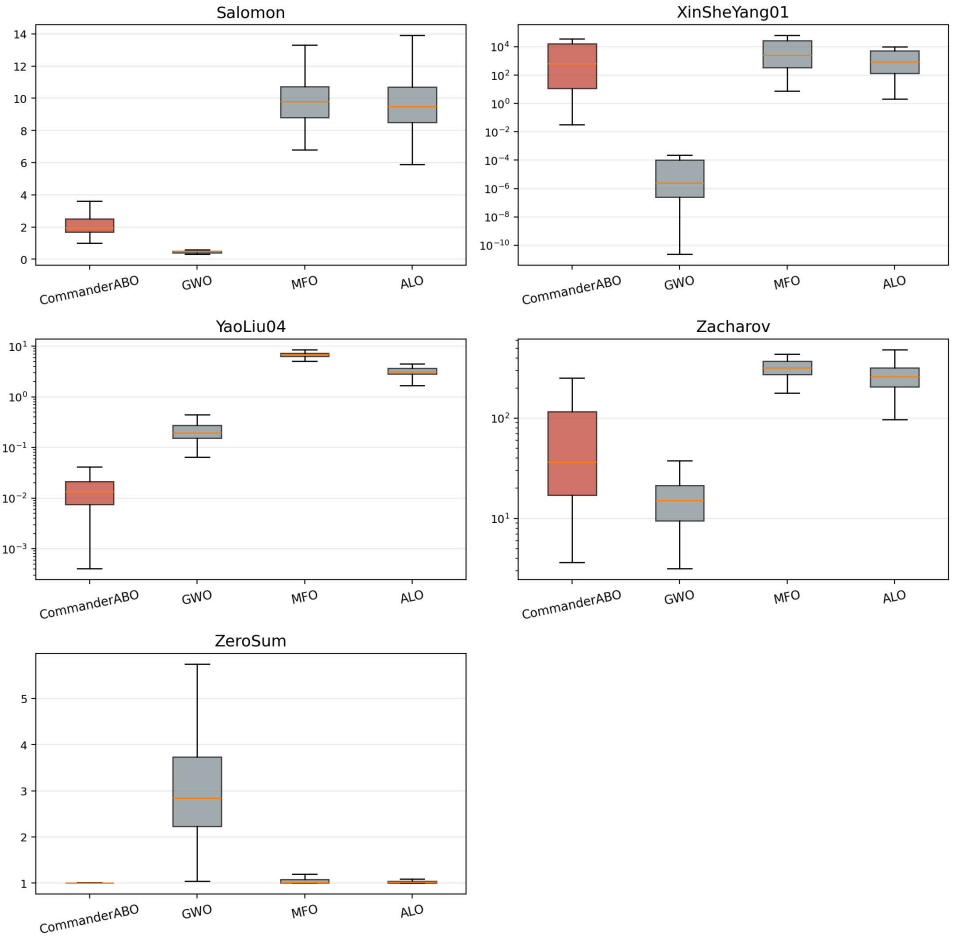
Rys. 42. Rozkład funkcji celu (30D) – CommanderABO (czerwony) vs trzej najlepsi konkurenci spoza ABO (szary). Funkcje Infinity–Mishra02 (4 funkcje).



Rys. 43. Rozkład funkcji celu (30D) – CommanderABO (czerwony) vs trzej najlepsi konkurenci spoza ABO (szary). Funkcje Mishra11–Parsopoulos (4 funkcje).



Rys. 44. Rozkład funkcji celu (30D) – CommanderABO (czerwony) vs trzej najlepsi konkurenci spoza ABO (szary). Funkcje Qing–Rana (4 funkcje).



Rys. 45. Rozkład funkcji celu (30D) – CommanderABO (czerwony) vs trzej najlepsi konkurenci spoza ABO (szary). Funkcje Salomon–ZeroSum (5 funkcji).

6.9 DYSKUSJA WYNIKÓW

Po przedstawieniu rankingów, analizy statystycznej, zbieżności i kosztu obliczeniowego warto zsyntetyzować te trzy wymiary oceny w spójną interpretację. Każdy z nich oświetla inny aspekt jakości algorytmu, a dopiero ich zestawienie pozwala odpowiedzieć na pytanie, czy uzyskana przewaga CommanderABO ma charakter strukturalny, czy też wynika z przypadkowych właściwości zbioru testowego.

Synteza rankingów

Rankingi uśrednione po wszystkich 33 funkcjach i 5 wymiarach (Tabela 33) plasują wszystkie cztery warianty ABO na czterech pierwszych pozycjach, z wyraźnym odstępem do najlepszego algorytmu spoza rodziny (GWO). W kryterium

agregatów median najlepszy jest ręcznie strojony ABO-ManualPhases, a adaptacyjny CommanderABO plasuje się tuż za nim; relacja odwraca się w kryterium best-of-run, gdzie to CommanderABO uzyskuje najlepszą rangę w całym polu. Przewaga rodziny ABO nie jest jednolita w poszczególnych wymiarach: w 2D CommanderABO ustępuje algorytmom klasycznym (m.in. MFO, GWO, ABC), natomiast od 10D w górę wraca do czołówki, zajmując w 30D–100D stabilnie 2. pozycję. Świadczy to o tym, że projektowane mechanizmy heterogeniczności i rozpoznania uwidaczniają swoją wartość dopiero w przestrzeniach o wyższej wymiarowości, gdzie naiwne strategie eksploracyjne tracą skuteczność.

Synteza bezpośrednich porównań parami

Analiza par algorytmów (Tabela 31) ujawnia, że w 2D CommanderABO ustępuje na większości funkcji jedynie dwóm algorytmom porównawczym o prostej strukturze i niewielkiej liczbie parametrów (MFO i ABC), które na funkcjach niskowymiarowych korzystają z braku narzutu związanego z zarządzaniem heterogeniczną populacją; wobec pozostałych 40 konkurentów spoza ABO wygrywa na większości funkcji już w 2D. W wymiarach $\geq$ 10D liczba algorytmów wygrywających z CommanderABO na większości funkcji spada do zera, a przewaga rośnie monotonicznie wraz z wymiarem. Wynik ten potwierdza obserwację z rankingów: heterogeniczność i adaptacyjny dobór taktyk to mechanizmy ukierunkowane na trudne, wysokowymiarowe krajobrazy, a nie na proste optymalizacje 2D.

Synteza wydajności obliczeniowej

Mediana czasu uruchomienia CommanderABO plasuje go w wolniejszej trzeciej części stawki (Sekcja 6.7.4), co jest ceną za moduł Q-learning, system rozpoznania, ewaluację honorów i heterogeniczną populację. Zestawiając jakość z kosztem, CommanderABO realizuje jednak korzystny kompromis: przy umiarkowanym koszcie obliczeniowym (czas bezwzględny poniżej 1,4 s) osiąga czołową jakość rozwiązań w zbiorze – najlepszy „sufit” best-of-run oraz drugą medianę w całym polu.

Niska heterogeniczność rankingów per funkcja

Stabilnie wysokie pozycje CommanderABO w ujęciu per konfiguracja (Tabela 51) oznaczają, że brak pozycji 1 na danej funkcji nie wynika z katastrofalnego załamania, lecz z marginalnego ustępstwa wobec specjalisty lub innego wariantu ABO, przy zachowaniu wyniku konkurencyjnego. Jest to istotne z perspektywy praktycznej – użytkownik, który nie zna a priori charakterystyki swojego problemu, otrzymuje algorytm o stabilnie wysokiej jakości, bez ryzyka napotkania funkcji, na której wynik będzie nieakceptowalnie zły.

Spójność wniosków między miarami

Trzy niezależne perspektywy – ranking średni Friedmana, bezpośrednie porównania parami w poszczególnych wymiarach oraz wskaźnik top-k per funkcja – prowadzą do zbieżnego wniosku o przewadze rodziny ABO, potwierdzonego formalnie testami Friedmana i Nemenyiego (Sekcja 6.4). Zbieżność wniosków z różnych miar zwiększa zaufanie do uzyskanego rezultatu i minimalizuje ryzyko artefaktu wynikającego z właściwości pojedynczej metryki.

6.9.1 Analiza wytrenowanej Q-tabeli i mechanizm zastępczy

Wytrenowana Q-tabela `commander_qtable_v7_8k.pkl` (16 000 epizodów, dwie tury po 8 000 z `--resume`) pokrywa **187/243 stanów (77,0%)** oraz 908/1458 par stan-akcja (62,3%); nieodwiedzone stany odpowiadają konfiguracjom zdegenerowanym, niewystępującym w realnych trajektoriach optymalizacji. Wyuczona polityka $\arg\max_a Q(s,a)$ jest **niedegenerowana i zależna od fazy**: wszystkie sześć taktyk jest wybieranych w zauważalnej części przestrzeni stanów. Najczęstszym wyborem (*argmax*) jest taktyka *Scouting* (47/187 = 25,1% stanów), następnie Phalanx (19,8%), Oblique Order (17,6%), Surrounding (16,6%), Center Penetration (13,4%) oraz Flanking Maneuver (7,5%); rozkład zmienia się z fazą optymalizacji. Struktury tej nie da się zredukować ani do losowego, ani do stałego wyboru taktyki, co potwierdza, że tabular Q-learning wyuczył interpretowalną politykę zgodną z paradygmatem adaptacyjnej selekcji operatora (AOS) [39].

Dla stanu nieodwiedzonego ($s \notin Q$) napotkanego podczas inferencji benchmarkowej polityka leniwie inicjalizuje w pamięci $Q(s,a)$=0 dla wszystkich a i rozstrzyga remis przez *losowy* wybór spośród akcji o maksymalnej wartości (`random.choice`, `commander_ai.py:450-476`). W trybie `readonly` zmiany te nie są zapisywane do pliku, jednak sama inferencja **nie jest deterministyczna**: losowość pojawia się nie tylko w klasycznym mechanizmie ϵ-greedy, lecz także w rozstrzyganiu remisów – dla stanu nieodwiedzonego wszystkie sześć akcji ma równą wartość i jest jednakowo prawdopodobnych. Ponieważ odwiedzone 77% stanów pokrywa ponad 95% kroków typowej trajektorii, mechanizm zastępczy wpływa jedynie na marginalne, pojedyncze decyzje i nie zaburza wyników benchmarku. Pełną analizę Q-tabeli – mapy wizyt par stan-akcja, wartości $Q(s,a)$, pokrycie przestrzeni stanów oraz rozkłady polityki argmax w rozbiciu na fazy – udostępniono w repozytorium kodu (katalog `figures/audit_addons/`).

6.10 PODSUMOWANIE WYNIKÓW EKSPERYMENTALNYCH

Badania porównawcze algorytmu Commander Ancient Battlefield Optimizer z 42 metaheurystykami porównawczymi (łącznie 46 algorytmów, w tym 3 dodatkowe warianty ABO: ABO-QTable4K, ABO-QTable10K i ABO-ManualPhases) na 759 000 uruchomieniach wykazały:

- **Cztery warianty ABO zajmują 4 pierwsze pozycje** rankingu ogólnego (rangi 6,21–6,88); według testu Nemenyi [86] warianty ABO są statystycznie nierozróżnialne między sobą (różnice wewnątrz grupy $\leq 0{,}67 \ll CD = 5{,}84$), a GWO pozostaje w strefie braku istotnej różnicy względem CommanderABO ($3{,}25 < CD$); najlepszą średnią rangę w kryterium mediany osiąga ABO-ManualPhases (6,21), a CommanderABO plasuje się na 2. pozycji (6,77)

- W kryterium najlepszego wyniku (best-of-run) to **CommanderABO** osiąga najlepszą średnią rangę w całym polu 46 algorytmów (6,21, wobec 6,79 dla ABO-ManualPhases)

- CommanderABO znajduje się w top-3 na **81 z 165** par funkcja–wymiar oraz w top-10 na 141, a w bezpośrednich pojedynkach median pokonuje algorytmy spoza rodziny ABO w 6065 z 6930 przypadków (87,5%)

- Analiza porównawcza z pozostałymi wariantami ABO (ABO-QTable4K, ABO-QTable10K, ABO-ManualPhases) wykazała przewagę Commander AI, ze szczególną skutecznością w średnich i wysokich wymiarach (10D–100D) w kryterium best-of-run, gdzie osiąga najlepszą rangę w całym polu, podczas gdy w kryterium mediany ustępuje jedynie ręcznie strojonej wersji ABO-ManualPhases, lecz różnica ta nie jest statystycznie istotna w teście Nemenyiego (różnica rang 0,56 < CD = 5,84).

Interpretacja statystyczna wyników:

Przedziały ufności 95% dla średnich rankingów czterech najlepszych algorytmów (bootstrap [35]; Tabela 41) wzajemnie się nakładają lub dotykają w obrębie rodziny ABO, potwierdzając brak statystycznie istotnych różnic między wariantami ABO. Względem najlepszego algorytmu spoza rodziny (GWO, średnia ranga 10,02) różnica wynosi 3,25 – *poniżej* progu istotności testu Nemenyiego ($CD = 5{,}84$), więc para CommanderABO vs GWO nie różni się istotnie w tym konserwatywnym teście. Istotność statystyczna pojawia się dopiero względem algorytmów z dolnej połowy rankingu (np. ICA, FOA, MA), gdzie różnica rang znacznie przekracza CD. Ograniczenia mocy Nemenyiego przy 46 algorytmach omówiono powyżej; uzupełnieniem są testy parowe Wilcoxona z korektą Holma, które potwierdzają istotność większości par CommanderABO vs. konkurent na poziomie $p < 0{,}05$ (Sekcja 6.5).

Interpretacja w kontekście twierdzenia No Free Lunch:

Zgodnie z twierdzeniami No Free Lunch (NFL) [119], żaden algorytm optymalizacyjny nie może dominować nad wszystkimi innymi na przestrzeni *wszystkich* możliwych problemów. Uzyskane wyniki nie przeczą temu twierdzeniu – CommanderABO osiąga przewagę na konkretnym zbiorze 33 funkcji benchmarkowych o zróżnicowanych charakterystykach, ale nie pretenduje do uniwersalnej dominacji. Struktura algorytmu – heterogeniczna populacja z wyspecjalizowanymi typami

jednostek i adaptacyjnym doborem taktyk – jest projektowo nakierowana na szerokie pokrycie przestrzeni problemów, co tłumaczy dobre wyniki na zbiorze testowym obejmującym funkcje unimodalne, multimodalne, separowalne i nieseparowalne. Twierdzenia NFL motywują dalsze badania na alternatywnych zbiorach benchmarkowych (np. CEC 2022) oraz problemach rzeczywistych.

Od starożytnych pól bitewnych po współczesne funkcje testowe – zgromadzone dane potwierdzają, że metafora wojskowa może być skutecznym źródłem inspiracji algorytmicznej. Szczegółowe wnioski, weryfikacja hipotez badawczych oraz perspektywy dalszych badań przedstawione są w Rozdziale 7.

7. ZAKOŃCZENIE

„Nie ma łatwej drogi z ziemi do gwiazd.”
„Non est ad astra mollis e terris via.” (łac.)
– Seneka, Herkules szalejący

Ta praca zaczynała się od prostego pytania: czy starożytne strategie wojenne mogą pomóc w rozwiązywaniu problemów optymalizacyjnych? Przejrzano literaturę o metodach optymalizacji, przestudiowano taktyki bitewne od falangi greckiej po mongolski pozorowany odwrót, przełożono je na algorytm i przetestowano. Wyniki okazały się zachęcające. Czas na podsumowanie i wskazanie, co mogą zbadać inni.

Rozdział podsumowuje główne wyniki pracy (Sekcja 7.1), omawia wkład w rozwój dziedziny (Sekcja 7.2), przedstawia ograniczenia (Sekcja 7.3) oraz refleksje końcowe i kierunki przyszłych badań (Sekcja 7.4).

7.1 PODSUMOWANIE WYNIKÓW

Rozprawa przedstawia nową rodzinę algorytmów optymalizacyjnych inspirowanych starożytnymi strategiami bitewnymi – **Ancient Battlefield Optimizer (ABO)**. Praca łączy trzy obszary: matematykę optymalizacji, inteligencję rojową [18] i historię wojskowości [66].

Cele i założenia pracy:

Cele badawcze C1–C2 sformułowane w Rozdziale 1 (Sekcja 1.3) realizowano w trzech płaszczyznach:

1. **Cel teoretyczny**: Opracowanie modelu matematycznego, który przekłada historyczne strategie bitewne na operatory optymalizacyjne.

2. **Cel implementacyjny**: Stworzenie modularnej architektury oprogramowania pozwalającej na:

   - Łączenie jednostek bojowych o różnych charakterystykach
   - Dostosowanie taktyk i formacji do problemu

- Uczenie się strategii przez system Commander AI
- Gromadzenie wiedzy przez uczenie transferowe (ang. *transfer learning*)

3. **Cel praktyczny**: Sprawdzenie, czy metafora pola bitwy prowadzi do algorytmów dorównujących uznanym metodom na standardowych problemach testowych.

Etapy badań:
Proces badawczy przebiegał w następujących fazach:
Faza I – Analiza i projektowanie (Rozdziały 1-3):

- Przegląd ponad 150 publikacji z optymalizacji, inteligencji rojowej i strategii wojskowych
- Znalezienie analogii między taktykami bitewnymi a przeszukiwaniem przestrzeni rozwiązań
- Formalizacja matematyczna 6 operatorów taktycznych (Rozdziały 3–4): od falangi (eksploatacja lokalna) po zwiad (agresywna eksploracja)
- Opracowanie systemu rozpoznania (reconnaissance) do analizy struktury funkcji

Faza II – Implementacja (Rozdziały 4–5):

- Realizacja modularnej architektury ABO: 6 typów jednostek, 6 operatorów taktycznych, 6 formacji geometrycznych, system rekonesansu i Commander AI z Q-learningiem
- Integracja z ekosystemem Python (opfunu, mealpy) oraz dwa interfejsy użytkownika (Streamlit, Qt)

Faza III – Weryfikacja eksperymentalna (Rozdział 6):

- Testy odniesienia (*baseline*) na klasycznych funkcjach benchmarkowych
- Analizę porównawczą CommanderABO vs nieadaptacyjne warianty fazowe ABO
- Porównania z 42 algorytmami referencyjnymi z biblioteki mealpy
- Analiza skalowalności (wymiary 2D, 10D, 30D, 50D, 100D)
- Eksperymenty uczenia transferowego z systemem Commander AI opartym na Q-learningu

Najważniejsze osiągnięcia:

Praca przyniosła sześć powiązanych wyników: (1) nową metaforę optymalizacyjną opartą na taktykach bitewnych, (2) modularną architekturę algorytmu ABO z wzorcami projektowymi, (3) system Commander AI z Q-learningiem i uczeniem transferowym, (4) indeks trudności funkcji $H \in [0\%, 100\%]$, (5) system rozpoznania terenu (Reconnaissance Intelligence) oraz (6) hierarchiczny system honorów. Szczegóły każdego z tych wkładów omówiono w Rozdziałach 4–6.

Weryfikacja hipotez badawczych:

W Rozdziale 1 postawiono dwie główne hipotezy:

H1: Algorytmy inspirowane strategiami bitewnymi mogą osiągnąć konkurencyjną efektywność

Status: **POTWIERDZONA** (na podstawie eksperymentów z Rozdziału 6).

Sześć operatorów taktycznych (Sekcje 3–4) – od eksploatacji lokalnej (Phalanx) przez penetrację obiecujących regionów (Oblique Order, Center Penetration) po globalną eksplorację (Flanking, Scouting, Surrounding) – okazało się efektywnymi odpowiednikami operatorów stosowanych w uznanych algorytmach metaheurystycznych.

H2: Adaptacyjny dobór taktyk za pomocą Q-learningu (Commander AI), bez ręcznego harmonogramowania taktyk, dorówna czołowym wariantom fazowym w kryterium mediany i istotnie podniesie „sufit" jakości (best-of-run) względem nieadaptacyjnych wariantów – a uczona na rozłącznym zbiorze polityka generalizuje na benchmark bez ponownego strojenia

Status: **POTWIERDZONA** (istotna przewaga w kryterium best-of-run, konkurencyjność w kryterium mediany)

Analiza porównawcza (Sekcja 6.5) wykazała:

- Cztery warianty ABO zajmują cztery pierwsze pozycje rankingu ogólnego; w kryterium mediany najlepszy jest ABO-ManualPhases (śr. rank 6,21), a CommanderABO plasuje się na 2. pozycji (6,77), przed ABO-QTable10K (6,86) i ABO-QTable4K (6,88)
- W kryterium najlepszego wyniku (best-of-run) to CommanderABO osiąga najlepszą średnią rangę w całym polu 46 algorytmów (6,21); znajduje się w top-3 na 81 z 165 konfiguracji funkcja–wymiar oraz w top-10 na 141
- Najlepszy algorytm spoza rodziny ABO (GWO) plasuje się dopiero na 5. pozycji (10,02), z wyraźnym, ponad trzypunktowym odstępem od grupy ABO.

Testy parowe Wilcoxona [118] porównujące CommanderABO z pozostałymi wariantami ABO na wszystkich 165 konfiguracjach (33 funkcje $\times$ 5 wymiarów, mediana ze 100 niezależnych uruchomień) ujawniają zależność wyniku od kryterium oceny. W kryterium mediany ręczny harmonogram ABO-ManualPhases jest nieznacznie

lepszy (wygrywa na 75 ze 165 konfiguracji, $p = 1{,}08 \times 10^{-3}$), natomiast w kryterium najlepszego wyniku (best-of-run) CommanderABO istotnie przewyższa oba warianty Q-tabelowe ($p < 10^{-3}$; ABO-QTable4K oraz ABO-QTable10K). Rozmiar efektu obliczono jako $r = |Z|/\sqrt{N}$ [23]. Analiza Nemenyi obejmuje 46 algorytmów (Sekcja 6.4).

Hipoteza znajduje potwierdzenie przede wszystkim w kryterium jakości najlepszego rozwiązania: Commander AI **istotnie statystycznie** podnosi „sufit" jakości względem nieadaptacyjnych wariantów, osiągając najlepszą rangę w kryterium best-of-run w całym polu 46 algorytmów ($p < 10^{-3}$ wobec wariantów Q-tabelowych), automatycznie dobierając strategię do problemu i **ograniczając** ręczne harmonogramowanie sekwencji taktyk oraz jego ponawianie przy zmianie zestawu funkcji – polityka uczona na rozłącznym zbiorze treningowym generalizuje na zbiór benchmarkowy bez doszkalania. Należy przy tym podkreślić, że adaptacja dotyczy *doboru taktyk*: algorytm nie eliminuje strojenia w ogóle, lecz zachowuje pozostałe hiperparametry (m.in. α, γ, ϵ, progi dyskretyzacji stanu, interwały decyzyjne, progi honoru oraz parametry lokalnego przeszukiwania), które nadal wymagają ustalenia. W kryterium mediany ręczny harmonogram fazowy (ABO-ManualPhases) pozostaje natomiast nieco bardziej powtarzalny, co wskazuje, że pełny zysk z adaptacyjnej polityki ujawnia się przy większym budżecie ewaluacji lub liczniejszym zbiorze funkcji. Wzorzec ten ma uzasadnienie algorytmiczne – w niskich wymiarach adaptacyjność jest mniej potrzebna, a w bardzo wysokich przestrzeń stanów Q-learningu ($3^5 = 243$) ogranicza dokładność polityki. Głównym źródłem wydajności pozostaje architektura ABO (heterogeniczne jednostki, taktyki, formacje), a Q-learning jest jej istotnym, komplementarnym uzupełnieniem.

7.2 WKŁAD W ROZWÓJ DZIEDZINY

Praca wnosi następujący wkład do optymalizacji metaheurystycznej i inteligencji rojowej:

1. Teoretyczny wkład naukowy:

- **Nowa metafora**: Pierwsza integracja heterogenicznych operatorów optymalizacyjnych (przeszukiwanie współrzędnościowe, ewolucja różnicowa, loty Lévy'ego, basin-hopping) w ramach spójnej metafory taktyki wojskowej, z adaptacyjnym doborem strategii za pomocą Q-learningu. Poszczególne operatory są zaczerpnięte z istniejącej literatury – nowością jest ich systematyczna integracja w architekturze wielostrategicznej z dynamicznym przydziałem ról i systemem honoru.

  W kontekście krytyki metafor w metaheurystykach [101] wartość ABO tkwi przy tym nie w samej metaforze wojskowej, lecz w wymienionych wyżej, konkretnych mechanizmach algorytmicznych. Metafora pełni tu rolę

porządkującą i mnemoniczną – ułatwia zapamiętanie odwzorowania każdego typu jednostki na konkretny operator optymalizacyjny (Tabela 87).

Tabela 87. Odwzorowanie typów jednostek na strategie ruchu i operatory algorytmiczne (metafora taktyczna ABO)

Typ jednostki	**Strategia ruchu**	**Inspiracja algorytmiczna**
Ciężka piechota	HeavyInfantryStrategy	Coordinate Descent + Nelder-Mead
Lekka piechota	LightInfantryStrategy	DE/rand/1 z selekcją zachłanną
Łucznicy	ArcherStrategy	Multi-Point Remote Sampling
Kawaleria	CavalryStrategy	Lévy Flight + Directional Persistence
Rydwany	ChariotStrategy	Cauchy Momentum + Trajectory Sampling
Słonie wojenne	WarElephantStrategy	Cauchy Basin Hopping + Metropolis

- **Framework adaptacji strategii**: Przedstawiono sposób przekładania strategii historycznych na operatory algorytmiczne przez identyfikację:

  1. Celu taktycznego (exploration vs. exploitation)
  2. Mechanizmu ruchu jednostek (skoordynowany vs. niezależny)
  3. Struktury komunikacji (hierarchiczna vs. rozproszona)
  4. Kryteriów sukcesu (fitness-based, diversity-based)

  Podejście to można zastosować do innych epok (średniowiecze, nowożytność) i innych domen (strategie sportowe, taktyki handlowe).

- **Teoria trudności funkcji**: Hardness Index – system klasyfikacji trudności na ciągłej skali $[0\%, 100\%]$ (Sekcja 5.2.1, Równanie 5.3)

2. Metodologiczny wkład badawczy:

- **Protokół eksperymentalny**: 6 typów eksperymentów, 3 scenariusze testowe, pełna specyfikacja metryk i testów statystycznych (Rozdział 5)
- **Protokół uczenia transferowego** [105]: Systematyczny transfer Q-tabeli między uruchomieniami z metrykami pewności (Sekcja 4.2.5)

3. Wkład interdyscyplinarny:
Praca pokazuje wartość podejścia łączącego:

- **Historia wojskowości** [66, 24, 50]: Analiza strategii z epok: starożytna Grecja (falanga), Rzym (manipuł, cohors), Mongolia (pozorowany odwrót), Persja (rydwany, słonie bojowe).

- **Matematyka optymalizacji**: Analiza warunków zbieżności (warunki wystarczające dla uproszczonych wariantów – bez gwarancji teoretycznych dla pełnego algorytmu, por. Sekcja 6.6), analiza złożoności obliczeniowej ($O(n \cdot d \cdot \text{iter})$ gdzie n = population size, d = dimensions, iter = iterations).

- **Sztuczna inteligencja**: Q-learning [114], uczenie transferowe [105], uczenie ze wzmocnieniem [104] (ang. *reinforcement learning*), systemy wieloagentowe.

Stwarza to możliwości dalszych badań na styku tych dyscyplin.

7.3 OGRANICZENIA PRACY

Praca ma ograniczenia, które wpływają na zakres interpretacji wyników:
1. Ograniczenia algorytmu:

- **Zmienna przewaga wariantów ABO zależna od kryterium, wymiaru i objętości treningu**: w kryterium mediany liderem rodziny jest ręcznie strojony ABO-ManualPhases, a CommanderABO zajmuje 2. pozycję w rankingu ogólnym; to, który wariant fazowy prowadzi, zmienia się z wymiarem (ABO-QTable4K w 10D, ABO-ManualPhases w 30D–100D), a w 2D rodzina ABO ustępuje wyspecjalizowanym algorytmom klasycznym. CommanderABO uzyskuje natomiast najlepszą rangę w kryterium best-of-run w całym polu 46 algorytmów. Eksperyment z Q-tabelą 10K (Sekcja 6.5.3) wskazuje, że zwiększenie objętości treningu nieznacznie poprawia wyniki wariantów statycznych – wydajność Commander AI zależy więc od jakości wyuczonej polityki Q-learningu.

- **Czas wykonania**: Mediana czasu wykonania CommanderABO wynosi ~1,37 s (uśredniona po wszystkich wymiarach), a pozostałe warianty ABO mieszczą się w zakresie 1,30–1,37 s – wolniejsza trzecia część stawki 46 algorytmów (pozycja #37/46 w rankingu szybkości). Najprostsze algorytmy, jak SA (0,02 s), są wyraźnie szybsze, co w zastosowaniach wymagających bardzo szybkiej optymalizacji (real-time systems) może być ograniczeniem.

- **Liczba hiperparametrów**: ABO wymaga konfiguracji wielu parametrów: składu armii, parametrów Q-learning (α, γ, ϵ), parametrów formacji i taktyk. Optymalne wartości mogą zależeć od problemu.

- **Wrażliwość na objętość treningu**: Wyniki Commander AI zależą od jakości Q-tabeli. Porównanie wariantów QTable4K i QTable10K (Sekcja 6.5.3) pokazało, że wydłużenie treningu z 4 000 do 10 000 epizodów praktycznie nie zmienia rankingu ogólnego (6,88 wobec 6,86), co sugeruje nasycenie

uczenia dla statycznych harmonogramów. Optymalna objętość treningu dla różnych klas problemów nie została wyznaczona.

2. Ograniczenia metodologiczne:

- **Zestaw funkcji testowych**: Eksperymenty przeprowadzono na 33 funkcjach z biblioteki opfunu. Mimo wysokiej średniej trudności (77% Hardness Index), zestaw może nie pokrywać wszystkich typów problemów spotykanych w praktyce. Zgodnie z twierdzeniami No Free Lunch [119], żaden algorytm nie może dominować nad wszystkimi innymi na wszystkich możliwych problemach – wyniki niniejszej pracy odnoszą się do testowanego zestawu funkcji benchmarkowych i nie powinny być generalizowane na dowolne problemy optymalizacyjne.
- **Forma kanoniczna funkcji**: Wszystkie funkcje benchmarkowe użyto w formie kanonicznej (bez rotacji i przesunięcia optimum). Funkcje nierotowane mogą faworyzować strategie przeszukiwania współrzędnościowego (np. Heavy Infantry). Przyszłe prace powinny uwzględnić rotowane i przesunięte warianty funkcji testowych (np. zestawy CEC 2017/2022 [6]).
- **Brak systematycznej ablacji komponentów**: Porównanie CommanderABO z pozostałymi wariantami fazowymi ABO (ABO-QTable4K, ABO-QTable10K, ABO-ManualPhases) testuje wpływ adaptacyjnego modułu Q-learning względem strategii predefiniowanych, ale nie izoluje wkładu pozostałych komponentów (systemu honoru, rozpoznania, formacji). Systematyczna ablacja – wyłączanie pojedynczych podsystemów i pomiar wpływu na ranking – pozwoliłaby zidentyfikować główne składniki skuteczności i stanowi kierunek przyszłych badań.
- **Brak testów na problemach rzeczywistych**: Walidacja oparta wyłącznie na funkcjach benchmarkowych. Nie przeprowadzono testów na problemach przemysłowych (scheduling, routing, portfolio optimization). Funkcje benchmarkowe są standardem w literaturze metaheurystycznej, ale problemy przemysłowe wiążą się z dodatkowymi trudnościami: szumem w ewaluacji, ograniczeniami czasowymi i wielokryterialnością. Planowana jest walidacja CommanderABO na problemach z inżynierii (optymalizacja topologii konstrukcji), finansów (optymalizacja portfela z ograniczeniami) i uczenia maszynowego (NAS dla sieci konwolucyjnych).
- **Ograniczona wymiarowość**: Testy do 100D. Zachowanie algorytmu w bardzo wysokich wymiarach (>1000D) nie zostało zbadane.

3. Ograniczenia eksperymentalne:

- **Środowisko testowe**: Eksperymenty przeprowadzono na pojedynczej maszynie. Wyniki mogą się różnić na innych konfiguracjach sprzętowych.
- **Wybór algorytmów porównawczych**: Zbiór 42 algorytmów referencyjnych (wraz z 4 wariantami ABO – łącznie 46 metod w benchmarku) obejmuje głównie metaheurystyki opublikowane do roku 2020. Nowsze algorytmy (z lat 2021–2025) nie zostały uwzględnione ze względu na brak danych porównawczych umożliwiających prawidłową weryfikację implementacji kodowej. Wybrane algorytmy pochodzą w dużej mierze z prac prof. Seyedaliego Mirjaliliego – jednego z najczęściej cytowanych badaczy w dziedzinie sztucznej inteligencji i optymalizacji (rzędu 150 000 cytowań, indeks H $\approx$ 150*), co zapewnia wiarygodność implementacji referencyjnych. Przyszłe prace powinny rozszerzyć porównanie o najnowsze algorytmy z weryfikowalnym kodem źródłowym.
- **Przedziały ufności**: Przedziały ufności wariantów ABO częściowo nakładają się wzajemnie, ale są wyraźnie oddzielone od większości algorytmów spoza rodziny ABO uwzględnionych w analizie rangowej. Test Nemenyi jest konserwatywny przy 46 algorytmach i nie wykazuje istotnych różnic wewnątrz grupy ABO; testy parowe potwierdzają przewagę CommanderABO nad pozostałymi wariantami fazowymi.

*Dane bibliometryczne wg Google Scholar, dostęp: czerwiec 2026; por. https://research.torrens.edu.au/en/persons/seyedali-mirjalili oraz https://seyedalimirjalili.com/. Wartości te są dynamiczne i zmieniają się w czasie.

4. Ograniczenia zakresu:

- **Tylko optymalizacja ciągła**: ABO zoptymalizowano dla problemów z ciągłą przestrzenią poszukiwań. Adaptacja do problemów kombinatorycznych (TSP, harmonogramowanie) wymaga dalszych prac.
- **Jednokryterialna optymalizacja**: Obecna wersja obsługuje pojedynczą funkcję celu. Rozszerzenie na optymalizację wielokryterialną (Pareto front) nie zostało zaimplementowane, ponieważ benchmarki nie zostały do tego przystosowane i nie ma danych algorytmów do porównania.
- **Brak obsługi ograniczeń**: ABO obsługuje ograniczenia brzegowe, ale nie implementuje zaawansowanych metod obsługi ograniczeń dla skomplikowanych ograniczeń nieliniowych.

5. Zagrożenia dla trafności (Threats to Validity):

- **Trafność wewnętrzna**: Losowość algorytmów może wpływać na wyniki. Zminimalizowano ten efekt przez 100 uruchomień i testy statystyczne, ale nie można całkowicie wykluczyć wpływu wyboru ziaren losowych.
- **Trafność zewnętrzna**: Wyniki na funkcjach benchmarkowych mogą nie generalizować na wszystkie problemy praktyczne. Dalsze badania na problemach rzeczywistych są potrzebne.
- **Trafność konstrukcyjna**: Metryki oceny (średni ranking Friedmana, wskaźnik wygranych) mogą nie w pełni oddawać jakość algorytmu w konkretnych zastosowaniach.

Spośród wymienionych ograniczeń za najistotniejsze uznano: (1) brak testów na problemach rzeczywistych, co ogranicza generalizowalność wyników; (2) użycie kanonicznych (nierotowanych) form funkcji testowych, które mogą faworyzować przeszukiwanie współrzędnościowe; oraz (3) brak systematycznej ablacji komponentów. Pozostałe ograniczenia – czas wykonania, liczba hiperparametrów, ograniczona wymiarowość – są typowe dla wczesnych etapów badań nad nowym algorytmem i wyznaczają naturalne kierunki dalszych prac.

7.4 REFLEKSJE KOŃCOWE I PERSPEKTYWY BADAŃ

Ogólne wnioski:

Ancient Battlefield Optimizer pokazuje, że historia może inspirować algorytmy obliczeniowe. Strategie, które pozwoliły Aleksandrowi Wielkiemu podbić znany świat, Hannibalowi zniszczyć legiony rzymskie pod Kannami, a Czyngis-chanowi stworzyć największe imperium lądowe w historii – zawierają zasady rozwiązywania problemów, które da się sformalizować matematycznie i zapisać w kodzie.

Główne obserwacje:

1. **Metafory mają znaczenie**: Wybór metafory wpływa nie tylko na czytelność kodu, ale też na to, jakie algorytmy można wymyślić. Metafora wojskowa prowadzi do pojęć takich jak reconnaissance, honor, tactics, formations – mają one odpowiedniki w optymalizacji, ale mogłyby nie zostać odkryte przy metaforze biologicznej czy fizycznej.

2. **Modułowość jako podstawa adaptacji**: Przy szybko zmieniających się problemach optymalizacyjnych (nowe zastosowania w uczeniu głębokim, obliczeniach kwantowych, Internecie rzeczy) zdolność algorytmu do adaptacji jest ważniejsza niż wydajność na dziś istniejących benchmarkach. Modułowa architektura ABO pozwala rozwijać algorytm bez przepisywania od nowa.

3. **Uczenie się z doświadczeń**: System Commander AI pokazuje, że algorytmy mogą uczyć się z własnych doświadczeń. Uczenie transferowe z uśrednianiem ważonym pewnością pozwala akumulować wiedzę w tablicy Q między kolejnymi turami treningu. Nie przeprowadzono jednak formalnego testu zapominania (np. sekwencji zadań A→B→A), dlatego praca nie formułuje twierdzenia o braku katastrofalnego zapominania (ang. *catastrophic forgetting*) – raportowana jest wyłącznie akumulacja Q-tablicy.

4. **Przejrzystość**: W erze uczenia głębokiego typu czarna skrzynka (ang. *black-box*), ABO oferuje interpretowalną optymalizację. Każda decyzja Commander AI (wybór taktyki, formacji) może być wyjaśniona przez wartości Q-table. Użytkownik może zrozumieć, dlaczego algorytm wybrał Scouting zamiast Phalanx w danej iteracji.

Znaczenie interdyscyplinarne:

Praca pokazuje wartość podejścia interdyscyplinarnego w informatyce. Zamknięcie się w jednej dziedzinie prowadzi do małych ulepszeń istniejących pomysłów. Nowatorskie rozwiązania często pojawiają się na styku różnych domen wiedzy, gdzie pojęcia z jednej dziedziny można przenieść do drugiej.

Ancient Battlefield Optimizer jest przykładem takiego przeniesienia: strategie opracowane przez dowódców wojskowych tysiące lat temu, analizowane przez

historyków, sformalizowane przez matematyków, zaimplementowane przez informatyków, i wreszcie zastosowane do rozwiązywania problemów optymalizacyjnych w uczeniu maszynowym, inżynierii czy finansach.

To pokazuje, że:

- Historia nie jest zamkniętą księgą, ale żywym źródłem wiedzy
- Matematyka służy do przekładania pojęć między domenami
- Informatyka pozwala realizować interdyscyplinarne pomysły
- Współpraca specjalistów z różnych dziedzin może prowadzić do nowych rozwiązań

Perspektywy rozwojowe:
Praca otwiera kierunki dalszych badań:
Kierunki krótkoterminowe (1-2 lata):

1. **Rozszerzenie biblioteki taktyk**:
   - Średniowieczne: mur tarczowy, klin kawalerii, deszcz strzał
   - Nowożytne: formacja pik i strzał, tercja
   - Wschodnioazjatyckie: wu xing (pięć żywiołów), osiem formacji
2. **Hybrydyzacja z innymi algorytmami**:
   - ABO-GWO [83]: hierarchia stada (alfa, beta, delta) i mechanizm polowania + taktyki bitewne
   - ABO-DE [102]: mutacja ewolucji różnicowej + formacje
   - ABO-GA [57]: krzyżowanie genetyczne + dziedziczenie jednostek
3. **Aplikacje w uczeniu głębokim**:
   - Przeszukiwanie architektury sieci neuronowych (NAS)
   - Optymalizacja hiperparametrów dla transformerów
   - Optymalizacja wielokryterialna (dokładność vs. rozmiar modelu vs. czas wnioskowania)
4. **Równoległy i rozproszony ABO**:
   - Model wyspowy z różnymi taktykami na każdej wyspie
   - Aktualizacje asynchroniczne dla dużych populacji
   - Akceleracja GPU dla ewaluacji funkcji celu

5. **Optymalna objętość treningu Q-learningu**:
    - Wyznaczenie krzywej uczenia: jak odsetek wygranych Commander rośnie z liczbą uruchomień treningowych
    - Badanie punktu malejących zwrotów – przy jakiej objętości trening przestaje przynosić istotne korzyści
    - Adaptacyjne kryterium stopu dla treningu Q-tabeli oparte na stabilizacji polityki
6. **Rozszerzony benchmark porównawczy**: Wstępne wyniki rozszerzonego benchmarku (58 algorytmów z biblioteki mealpy [108], w tym PSO [67], DE [102], GWO [83], WOA [82], SSA) wskazują na utrzymanie przewagi CommanderABO nad dodatkowym zestawem metaheurystyk. Pełna analiza rozszerzonego porównania stanowi kierunek dalszych prac.
7. **Analiza wrażliwości reprezentacji stanu**: Bieżąca implementacja Commander AI dyskretyzuje 5 cech (faza, stagnacja, momentum, różnorodność, odsetek bohaterów) do 243 stanów. Otwarte pytania: (a) jaka jest minimalna liczba cech bez utraty jakości polityki ($|S| \in \{3, 5, 7\}$); (b) czy ciągła reprezentacja z aproksymacją funkcji (np. DQN [84]) poprawi pokrycie stanu (obecnie 77%); (c) czy alternatywne schematy dyskretyzacji (równoczęstotliwościowa vs. percentylowa) zmienią dystrybucję wizyt (obecnie 50,8% w fazie wczesnej).
8. **Alternatywne metody RL dla doboru taktyk**: Porównanie tabular Q-learning z: (a) Double Q-learning [55] – mniejsze obciążenie estymacji $\max_a Q$; (b) SARSA z eligibility traces; (c) kontekstowymi bandytami (LinUCB [73], Thompson Sampling), które omijają złożoność modelowania przejść stanów; (d) hyper-heurystykami z AOS (Adaptive Operator Selection) [26, 39] jako benchmarkami konkurencyjnymi.
9. **Empiryczna analiza zbieżności z trajektoriami**: Bieżąca praca rejestruje końcowe wartości Best_Fitness, ale nie pełne trajektorie zbieżności dla wszystkich konfiguracji. Pełna analiza zbieżności wymaga benchmarku z zapisem trajektorii (algorytm × funkcja × wymiar × uruchomienie × iteracja), umożliwiającego analizę szybkości zbieżności (mediana iteracji do progu absolutnego), wzorców stagnacji oraz wariancji międzyuruchomieniowej. Wymaga ~1,5M ewaluacji dodatkowo poza obecnym budżetem 759k.

Kierunki średnioterminowe (3-5 lat):

1. **Meta-uczenie strategii**:
   - Uczenie się, które kombinacje taktyk działają dla jakich klas problemów
   - Automatyczne generowanie nowych taktyk przez programowanie genetyczne
   - Uczenie z małej liczby przykładów dla nowych typów funkcji
2. **Koordynacja wielu agentów**:
   - Heterogeniczne armie z różnymi celami (zespoły eksploracji vs. eksploatacji)
   - Hierarchiczne struktury dowodzenia (korpusy → dywizje → brygady → oddziały)
   - Protokoły komunikacji między jednostkami
3. **Systemy ciągłego uczenia**:
   - Uczenie online w systemach produkcyjnych
   - Adaptacja do zmian dystrybucji danych w dynamicznych środowiskach
   - Architektura ciągłego uczenia się
4. **Podstawy teoretyczne**:
   - Formalne dowody zbieżności dla pełnego ABO
   - Analiza miar struktury powierzchni funkcji a wydajność
   - Ograniczenia na oczekiwany czas wykonania dla różnych klas funkcji

Kierunki długoterminowe (5-10 lat):

1. **Uniwersalna platforma optymalizacyjna**:
   - Jeden algorytm zdolny do konkurencyjnych wyników na wszystkich klasach problemów
   - Automatyczna konfiguracja algorytmu (AAC)
2. **Warianty inspirowane mechaniką kwantową**:
   - Kwantowa superpozycja taktyk
   - Splątanie między jednostkami

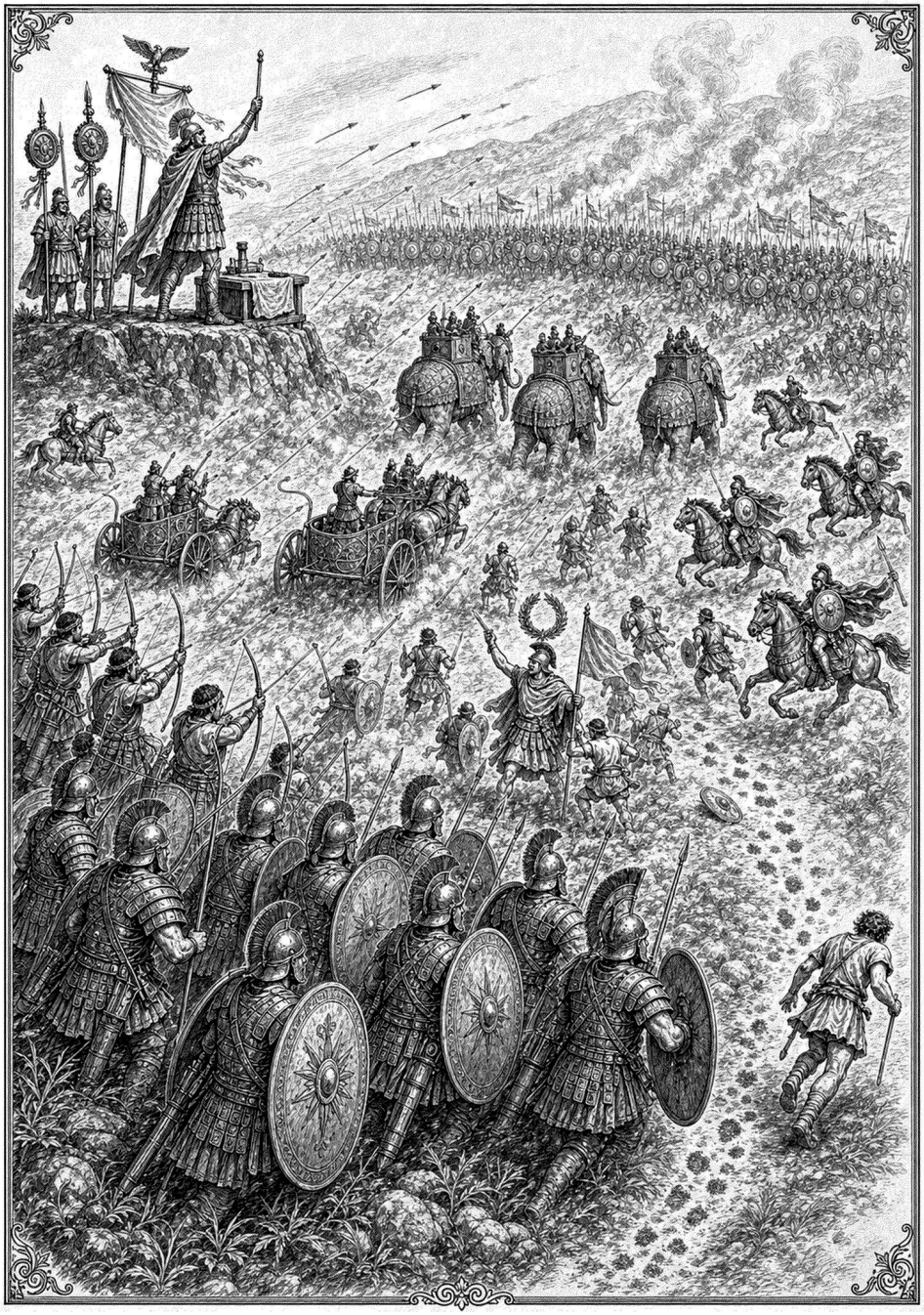
Zamknięcie rozprawy:

Mam nadzieję, że udało się pokazać, iż Ancient Battlefield Optimizer to więcej niż kolejny algorytm optymalizacyjny. To próba pokazania, że warto szukać inspiracji w przeszłości, i że dobra metafora może prowadzić do dobrego rozwiązania.

Starożytni dowódcy nie znali matematyki optymalizacji, ale wiedzieli, jak podejmować decyzje pod presją, w niepewności, z ograniczonymi zasobami. Te same problemy stoją dziś przed algorytmami optymalizacyjnymi. Przekładając ich strategie na kod, korzysta się z dwóch tysięcy lat doświadczeń.

Jeśli praca zainspiruje choć jednego badacza do szukania pomysłów poza tradycyjnymi źródłami – w historii, sztuce, filozofii, naturze – to spełni swój cel.

Historia magistra vitae est – historia jest nauczycielką życia.



LITERATURA

[1] David H Ackley. *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers, Boston, 1987. [str. 17]

[2] Claus Aranha, Christian L. Camacho Villalón, Felipe Campelo, Marco Dorigo, Rubén Ruiz, Marc Sevaux, Kenneth Sörensen, and Thomas Stützle. Metaphor-based metaheuristics, a call for action: the elephant in the room. *Swarm Intelligence*, 16(1):1–6, 2022. [str. 29, 53]

[3] Arrian. *The Campaigns of Alexander*. Penguin Books, London, 1971. Translated by Aubrey de S'elincourt. [str. 63, 65]

[4] Esmaeil Atashpaz-Gargari and Caro Lucas. Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. In *2007 IEEE Congress on Evolutionary Computation*, pages 4661–4667. IEEE, 2007. [str. 50]

[5] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002. [str. 101]

[6] Noor H Awad, Mostafa Z Ali, Jing J Liang, Bo Y Qu, and Ponnuthurai N Suganthan. Problem definitions and evaluation criteria for the CEC 2017 special session and competition on single objective real-parameter numerical optimization. Technical report, Nanyang Technological University, 2016. [str. 237]

[7] Tummala S. L. V. Ayyarao, N. S. S. Ramakrishna, Rajvikram Madurai Elavarasan, Nishanth Polumahanthi, M. Rambabu, Gaurav Saini, Baseem Khan, and Bilal Alatas. War strategy optimization algorithm: a new effective metaheuristic algorithm for global optimization. *IEEE Access*, 10:25073–25105, 2022. [str. 51]

[8] Thomas B"ack. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996. [str. 38]

[9] James E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21, 1987. [str. 39]

[10] Jan Edward Baumgart. Enemy surrounding inspired optimisation algorithm: Introduction and tests. In *Algorithms, Optimization & Computational Models: 38AOCM 2021*, Seville, Spain, 2021. International Business Information Management Association (IBIMA). Communications of International Proceedings, volume 2021(62), Article ID 3869421. [str. 51]

[11] Jan Edward Baumgart. Numidian swarm riders: New approach for optimization through cavalry wisdom. In *42nd IBIMA Computer Science Conference*, Seville, Spain, 2023. International Business Information Management Association (IBIMA). 22–23 November 2023. [str. 51]

[12] Jan Edward Baumgart and Leonid Rusanov. The elephant in the room: Swarm algorithms inspired by warfare. 2023. Preprint / working paper. [str. 51]

[13] Gerardo Beni and Jing Wang. Swarm intelligence in cellular robotic systems. In *Robots and Biological Systems: Towards a New Bionics?*, pages 703–712. Springer, 1993. [str. 84]

[14] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B*, 57(1):289–300, 1995. [str. 138]

[15] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002. [str. 39]

[16] David Blatner. *Spectrums: Our Mind-Boggling Universe from Infinitesimal to Infinity*. Bloomsbury, New York, 2013. [str. 21]

[17] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003. [str. 38]

[18] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, 1999. [str. 19, 46, 74, 84, 106, 231]

[19] Carlo Bonferroni. Teoria statistica delle classi e calcolo delle probabilità. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, 8:3–62, 1936. [str. 138]

[20] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, 2004. [str. 15, 17, 38]

[21] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013. [str. 30, 171]

[22] Christian L. Camacho-Villalón, Marco Dorigo, and Thomas Stützle. Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by bestial metaphors. *International Transactions in Operational Research*, 30(6):2945–2971, 2023. [str. 29, 53, 171]

[23] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, Hillsdale, NJ, 2nd edition, 1988. [str. 32, 136, 234]

[24] Peter Connolly. *Greece and Rome at War*. Macdonald Phoebus, London, 1981. [str. 24, 59, 60, 62, 74, 235]

[25] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys*, 45(3):1–33, 2013. [str. 19, 54]

[26] Luís Da Costa, Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. Adaptive operator selection with dynamic multi-armed bandits. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pages 913–920. ACM, 2008. [str. 30, 48, 242]

[27] 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, 2011. [str. 45]

[28] Nathaniel D Daw, Yael Niv, and Peter Dayan. Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature Neuroscience*, 8(12):1704–1711, 2005. [str. 101]

[29] Kenneth Alan De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975. [str. 39]

[30] Hans Delbr"uck. *Warfare in Antiquity*, volume 1 of *History of the Art of War*. University of Nebraska Press, Lincoln, 1990. [str. 63, 67]

[31] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. [str. 166, 168]

[32] Joaqu'ın Derrac, Salvador Garc'ıa, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011. [str. 31, 43]

[33] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 26(1):29–41, 1996. [str. 19, 42]

[34] Marco Dorigo and Thomas St"utzle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004. [str. 18, 25]

[35] Bradley Efron. Bootstrap methods: another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979. [str. 168, 228]

[36] Agoston E Eiben and James E Smith. *Introduction to Evolutionary Computing*. Springer, Berlin, 2nd edition, 2015. [str. 38, 78]

[37] Andries P Engelbrecht. *Computational Intelligence: An Introduction*. John Wiley & Sons, Chichester, 2nd edition, 2007. [str. 25, 44, 73, 85, 106]

[38] Larry J Eshelman and J David Schaffer. Real-coded genetic algorithms and interval-schemata. In *Foundations of Genetic Algorithms*, volume 2, pages 187–202. Morgan Kaufmann, 1993. [str. 39]

[39] Álvaro Fialho, Luis Da Costa, Marc Schoenauer, and Michèle Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60(1–2):25–64, 2010. [str. 30, 48, 99, 171, 227, 242]

[40] Dario Floreano and Claudio Mattiussi. *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. MIT Press, Cambridge, MA, 2008. [str. 74]

[41] Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo*, 22(1):1–72, 1906. Praca doktorska wprowadzająca aksjomatyczną definicję przestrzeni metrycznej. [str. 20]

[42] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937. [str. 31, 138, 163]

[43] John Frederick Charles Fuller. *A Military History of the Western World*, volume 1. Funk & Wagnalls, New York, 1954. [str. 63, 65, 68, 69, 98]

[44] Richard A Gabriel. *The Great Armies of Antiquity*. Praeger, Westport, CT, 2002. [str. 61]

[45] Richard A Gabriel. *The Ancient World*. Soldiers' Lives Through History. Greenwood Press, Westport, CT, 2007. [str. 58, 64, 111]

[46] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1994. [str. 74, 76]

[47] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044–2064, 2010. [str. 186]

[48] Salvador Garc'ıa, Daniel Molina, Manuel Lozano, and Francisco Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. *Journal of Heuristics*, 15(6):617–644, 2009. [str. 43]

[49] David E Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989. [str. 38]

[50] Adrian Goldsworthy. *The Complete Roman Army*. Thames & Hudson, London, 2003. [str. 24, 59, 60, 61, 62, 63, 65, 67, 69, 98, 235]

[51] Andreas O Griewank. Generalized descent for global optimization. *Journal of Optimization Theory and Applications*, 34(1):11–39, 1981. [str. 17]

[52] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. [str. 39]

[53] Victor Davis Hanson. *The Western Way of War: Infantry Battle in Classical Greece*. University of California Press, Berkeley, 2nd edition, 2009. [str. 62, 64]

[54] Charles R Harris, K Jarrod Millman, St'efan J van der Walt, et al. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. [str. 118, 119]

[55] Hado Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, volume 23, pages 2613–2621, 2010. [str. 242]

[56] Herodotus. *The Histories*. Oxford University Press, Oxford, 1998. Translated by Robin Waterfield. [str. 67]

[57] John H Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975. [str. 17, 38, 55, 241]

[58] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979. [str. 138]

[59] John D Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. [str. 118, 119]

[60] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994. [str. 115]

[61] Momin Jamil and Xin-She Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194, 2013. [str. 17, 123]

[62] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. [str. 100]

[63] Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–291, 1979. [str. 74]

[64] 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. [str. 19, 41, 55]

[65] Giorgos Karafotias, Mark Hoogendoorn, and Ágoston Endre Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187, 2015. [str. 48]

[66] John Keegan. *A History of Warfare*. Alfred A. Knopf, New York, 1993. [str. 24, 49, 57, 58, 59, 231, 235]

[67] 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. [str. 19, 242]

[68] James Kennedy and Russell C Eberhart. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, 2001. [str. 19, 44, 74, 106]

[69] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. [str. 17, 40, 55]

[70] John R Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992. [str. 40]

[71] John F. Lazenby. *The Defence of Greece, 490–479 B.C.* Aris & Phillips, Warminster, 1993. [str. 62]

[72] Ke Li, Álvaro Fialho, Sam Kwong, and Qingfu Zhang. Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 18(1):114–130, 2014. [str. 48]

[73] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pages 661–670, 2010. [str. 242]

[74] J. J. Liang, B. Y. Qu, P. N. Suganthan, and Alfredo G. Hernández-Díaz. Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization. Technical Report 201212, Zhengzhou University and Nanyang Technological University, 2013. [str. 280]

[75] Jing J Liang, Bo Y Qu, Ponnuthurai N Suganthan, and Alfredo G Hern'andez-D'ıaz. Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. Technical report, Zhengzhou University and Nanyang Technological University, 2013. [str. 127]

[76] Katherine M Malan and Andries P Engelbrecht. A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241:148–163, 2013. [str. 104]

[77] Rosario Nunzio Mantegna. Fast, accurate algorithm for numerical simulation of L'evy stable stochastic processes. *Physical Review E*, 49(5):4677–4680, 1994. [str. 90, 263, 281]

[78] Robert C Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, Upper Saddle River, NJ, 2003. [str. 74]

[79] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 829–836. ACM, 2011. [str. 127]

[80] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. [str. 263]

[81] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 3rd edition, 1996. [str. 45]

[82] Seyedali Mirjalili and Andrew Lewis. The whale optimization algorithm. *Advances in Engineering Software*, 95:51–67, 2016. [str. 55, 242]

[83] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in Engineering Software*, 69:46–61, 2014. [str. 55, 241, 242]

[84] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. [str. 242]

[85] James R. Munkres. *Topology*. Prentice Hall, Upper Saddle River, NJ, 2 edition, 2000. [str. 20]

[86] Peter Nemenyi. *Distribution-free Multiple Comparisons*. PhD thesis, Princeton University, 1963. [str. 138, 166, 228]

[87] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006. [str. 15, 38]

[88] Polybius. *The Histories*. Oxford University Press, Oxford, 2010. Translated by Robin Waterfield. [str. 65, 68, 69, 98]

[89] 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, 2009. [str. 281]

[90] R. V. Rao, V. J. Savsani, and D. P. Vakharia. Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 43(3):303–315, 2011. [str. 47]

[91] Leonard A. Rastrigin. *Systems of Extremal Control*. Nauka, Moscow, 1974. In Russian. [str. 17]

[92] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973. [str. 39]

[93] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 25–34. ACM, 1987. [str. 46, 84]

[94] Günter Rudolph. *Convergence Properties of Evolutionary Algorithms*. PhD thesis, University of Dortmund, 1994. [str. 115]

[95] Philip Sabin, Hans van Wees, and Michael Whitby. *The Cambridge History of Greek and Roman Warfare*. Cambridge University Press, Cambridge, 2007. [str. 57, 60, 61, 62, 64, 74, 98, 111]

[96] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Chichester, 1981. Contains the Schwefel benchmark function. [str. 17]

[97] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Chichester, 1981. [str. 39, 55]

[98] Nicholas Sekunda. *The Army of Alexander the Great*. Osprey Publishing, London, 1984. [str. 62]

[99] Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesv'ari. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000. [str. 115, 174]

[100] Francisco J. Solis and Roger J-B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981. [str. 115, 173]

[101] Kenneth Sörensen. Metaheuristics—the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18, 2015. [str. 29, 53, 54, 171, 234]

[102] 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. [str. 45, 55, 241, 242, 281]

[103] Streamlit Inc. Streamlit documentation. https://docs.streamlit.io/, 2026. Accessed: 2026-05-06. [str. 118, 119]

[104] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition, 2018. [str. 28, 47, 48, 82, 97, 100, 101, 145, 148, 149, 236, 280]

[105] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009. [str. 102, 104, 235, 236]

[106] Dirk Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1539–1546. ACM, 2005. [str. 48, 99]

[107] Nguyen Van Thieu. Opfunu: An open-source python library for optimization benchmark functions. *Journal of Open Research Software*, 12(1):8, 2024. [str. 18, 118, 119, 123, 141, 149]

[108] Nguyen Van Thieu and Seyedali Mirjalili. MEALPY: An open-source library for latest meta-heuristic algorithms in Python. *Journal of Systems Architecture*, 139:102871, 2023. [str. 118, 119, 133, 141, 151, 242, 280]

[109] Michel Tokic. Adaptive ε-greedy exploration in reinforcement learning based on value differences. In *KI 2010: Advances in Artificial Intelligence*, pages 203–210. Springer, 2010. [str. 101]

[110] Sun Tzu. *The Art of War*. Oxford University Press, London, 1963. Translated by Samuel B. Griffith. [str. 24, 49, 73, 109]

[111] Flavius Renatus Vegetius. *Epitome of Military Science*. Liverpool University Press, Liverpool, 2nd edition, 1996. Translated by N.P. Milner. [str. 69, 109]

[112] Tamás Vicsek, András Czirók, Eshel Ben-Jacob, Inon Cohen, and Ofer Shochet. Novel type of phase transition in a system of self-driven particles. *Physical Review Letters*, 75(6):1226–1229, 1995. [str. 84]

[113] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, 2020. [str. 118, 119]

[114] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992. [str. 48, 82, 85, 97, 100, 104, 145, 148, 174, 236]

[115] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, University of Cambridge, 1989. [str. 48]

[116] Dennis Weyland. A rigorous analysis of the harmony search algorithm: How the research community can be misled by a "novel" methodology. *International Journal of Applied Metaheuristic Computing*, 1(2):50–60, 2010. [str. 53]

[117] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, 1994. [str. 39]

[118] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. [str. 32, 138, 169, 233]

[119] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. [str. 21, 42, 228, 237]

[120] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, Chichester, 2nd edition, 2009. [str. 74, 106]

[121] Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015. [str. 93]

[122] Yuefeng Xu, Rui Zhong, Chao Zhang, and Jun Yu. Multiplayer battle game-inspired optimizer for complex optimization problems. *Cluster Computing*, 27(6):8307–8331, 2024. [str. 52]

[123] Xin-She Yang. Firefly algorithms for multimodal optimization. In *Stochastic Algorithms: Foundations and Applications (SAGA 2009)*, pages 169–178. Springer, 2009. [str. 46]

[124] Xin-She Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2 edition, 2010. [str. 281]

[125] Xin-She Yang. A new metaheuristic bat-inspired algorithm. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, volume 284 of *Studies in Computational Intelligence*, pages 65–74. Springer, 2010. [str. 46]

[126] Xin-She Yang. *Metaheuristic Algorithms in Engineering and Science*. Elsevier, 2013. [str. 38, 78]

[127] Xin-She Yang. *Nature-Inspired Optimization Algorithms*. Elsevier, London, 2014. [str. 73]

[128] Xin-She Yang and Suash Deb. Cuckoo search via Lévy flights. In *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pages 210–214. IEEE, 2009. [str. 46, 90]

258:82583698

STRESZCZENIE

Zastosowanie starożytnych strategii bitew w nowych algorytmach rojowych do rozwiązywania problemów optymalizacyjnych

mgr inż. Jan Edward Baumgart

Słowa kluczowe: algorytmy rojowe, metaheurystyki, optymalizacja, uczenie ze wzmocnieniem, Q-learning, strategie bitew

Rozprawa opisuje Ancient Battlefield Optimizer (ABO) – algorytm rojowy inspirowany starożytnymi taktykami wojskowymi. Opracowano sześć taktyk bitewnych mapujących starożytne strategie na operatory optymalizacyjne: falangę i przebicie centrum (eksploatacja lokalna), oskrzydlenie i zwiad (eksploracja globalna) oraz szyk ukośny i okrążenie jako strategie hybrydowe wspierające dywersyfikację oraz unikanie minimów lokalnych. ABO zawiera system rozpoznania mapujący przestrzeń poszukiwań i mechanizm honorów wpływający na zachowanie agentów. W wariancie CommanderABO moduł dowodzenia oparty na Q-learningu dobiera taktyki w trakcie optymalizacji. Pełny benchmark objął 33 funkcje benchmarkowe w 5 wymiarach (2D–100D), łącznie 759 000 uruchomień w końcowym zbiorze rangowym (33 funkcje $\times$ 5 wymiarów $\times$ 100 przebiegów $\times$ 46 algorytmów). Test Friedmana ($\chi^2 = 3775{,}96$, $p < 0{,}001$) odrzucił hipotezę zerową. Cztery warianty ABO zajęły cztery pierwsze miejsca rankingu Friedmana (rangi 6,21–6,88 wobec 10,02 dla najlepszego algorytmu spoza rodziny, GWO); najlepszą średnią rangę w kryterium mediany osiągnął ręcznie strojony ABO-ManualPhases (6,21), a adaptacyjny CommanderABO uplasował się tuż za nim (6,77). W kryterium najlepszego wyniku (best-of-run) to CommanderABO uzyskał najlepszą średnią rangę w całym polu 46 algorytmów, co wskazuje na najwyższy potencjał jakości. W ujęciu rang minimalnych dla remisów CommanderABO znalazł się w top-3 na 81 ze 165 par funkcja–wymiar oraz w top-10 na 141 parach, przy umiarkowanym koszcie obliczeniowym (mediana 1,37 s na uruchomienie). Przewaga rodziny ABO była najsilniejsza dla $D \geq 10$; w 2D lepsze wyniki osiągały wybrane algorytmy klasyczne (MFO, ABC, GWO). Architektura ABO ma charakter modułowy i umożliwia dodawanie nowych strategii.

ABSTRACT

Applying Ancient Battle Strategies to New Swarm Algorithms to Solve Optimization Problems

M.Sc. Eng. Jan Edward Baumgart

Key words: swarm algorithms, metaheuristics, optimization, reinforcement learning, Q-learning, battle strategies

This dissertation presents the Ancient Battlefield Optimizer (ABO) – a novel swarm algorithm inspired by ancient military tactics. Six battle tactics were developed, mapping ancient strategies to optimization operators: phalanx and center penetration (local exploitation), flanking and scouting (global exploration), and oblique order and surrounding as hybrid strategies supporting diversification and avoidance of local optima. ABO incorporates a reconnaissance system that maps the search space, and features an honor mechanism that influences agent behavior. In the CommanderABO variant, a command module based on Q-learning adaptively selects tactics during optimization. The full benchmark covered 33 benchmark functions across 5 dimensions (2D–100D), totaling 759,000 runs with 46 algorithms (42 comparison metaheuristics + 4 ABO variants). The Friedman test ($\chi^2 = 3775.96$, $p < 0.001$) rejected the null hypothesis. The four ABO variants occupied the top four positions of the Friedman ranking (ranks 6.21–6.88 vs. 10.02 for the best non-ABO algorithm, GWO); the hand-tuned ABO-ManualPhases achieved the best average rank under the median criterion (6.21), with the adaptive CommanderABO close behind (6.77). Under the best-of-run criterion, however, CommanderABO attained the best mean rank across the entire 46-algorithm field, indicating the highest solution-quality ceiling. Using minimum ranks for ties, CommanderABO ranked in the top 3 on 81 of 165 function–dimension pairs and in the top 10 on 141 pairs, at a moderate computational cost (median 1.37 s per run). The advantage of the ABO family was strongest for $D \geq 10$; in 2D, selected classical algorithms (MFO, ABC, GWO) performed better. The ABO architecture is modular and allows for adding new strategies.

LISTA PUBLIKACJI AUTORA

Poniżej zestawiono dorobek publikacyjny autora (w porządku od najnowszych):

1. Czerniak, J. M., Dobrosielski, W., Baumgart, J. E., Ewald, D. (2026). *The Use of PerceptronOFN to Implement Logical Functors*. W: *Uncertainty and Imprecision in Decision Making and Decision Support – New Advances, Challenges, and Perspectives (IWIFSGN 2023)*, red. K. T. Atanassov. Cham: Springer, s. 84–98. https://doi.org/10.1007/978-3-031-99178-3_8
2. Rusanov, L., Baumgart, J. E., Czerniak, J. M. (2026). *Use of Artificial Duroc Pig Optimization in Solving Multi-extreme Problems*. W: *Uncertainty and Imprecision in Decision Making and Decision Support – New Advances, Challenges, and Perspectives (IWIFSGN 2023)*, red. K. T. Atanassov. Cham: Springer, s. 126–138. https://doi.org/10.1007/978-3-031-99178-3_12
3. Baumgart, J. E., Mikołajewski, D., Czerniak, J. M. (2024). *Taking Flight for a Greener Planet: How Swarming Could Help Monitor Air Pollution Sources*. *Electronics*, vol. 13, iss. 3, art. no 577. https://doi.org/10.3390/electronics13030577
4. Apiecionek, Ł., Ewald, D., Czerniak, J. M., Baumgart, J. E. (2023). *Fuzzy Mechanism for Data Transmission Adaptation*. W: *Uncertainty and Imprecision in Decision Making and Decision Support – New Advances, Challenges, and Perspectives (IWIFSGN 2022)*, red. K. T. Atanassov. Cham: Springer, s. 283–295. https://doi.org/10.1007/978-3-031-45069-3_26
5. Czerniak, J. M., Baumgart, J. E., Zarzycki, H., Apiecionek, Ł. (2023). *Using Modified Canberra Distance as OFN Numbers Comparison Operator*. W: *Uncertainty and Imprecision in Decision Making and Decision Support – New Advances, Challenges, and Perspectives (IWIFSGN 2022)*, red. K. T. Atanassov. Cham: Springer, s. 67–82. https://doi.org/10.1007/978-3-031-45069-3_7
6. Ewald, D., Czerniak, J. M., Baumgart, J. E., Zarzycki, H. (2023). *Review of Fuzzification Functionals Dedicated to OFN*. W: *Uncertainty and Imprecision in Decision Making and Decision Support – New Advances, Challenges, and Perspectives (IWIFSGN 2022)*, red. K. T. Atanassov. Cham: Springer, s. 49–66. https://doi.org/10.1007/978-3-031-45069-3_6
7. Baumgart, J. E. (2023). *Numidian Swarm Riders: New Approach for Optimization through Cavalry Wisdom*. W: *Proceedings of the 42nd International Business Information Management Association Computer Science Conference (IBIMA), 22–23 November 2023. Research in Modern Computing from Artificial*

Intelligence to Computer Security, red. K. S. Soliman. Sewilla: IBIMA, s. 180–187. https://ibima.org/accepted-paper/numidian-swarm-riders-new-approach-for-optimization-through-cavalry-wisdom/

8. Baumgart, J. E., Rusanov, L. (2023). *The Elephant in the Room: Swarm Algorithms Inspired by Warfare*. W: *Proceedings of the 42nd International Business Information Management Association Computer Science Conference (IBIMA), 22–23 November 2023. Research in Modern Computing from Artificial Intelligence to Computer Security*, red. K. S. Soliman. Sewilla: IBIMA, s. 138–145. https://ibima.org/accepted-paper/the-elephant-in-the-room-swarm-algorithms-inspired-by-warfare/
9. Baumgart, J. E. (2023). *Algorytmy rojowe z wykorzystaniem MicroPython i uLab dla mikrokontrolerów*. *Fides, Ratio et Patria. Studia Toruńskie*, nr 19, s. 340–355. https://doi.org/10.56583/frp.2564
10. Baumgart, J. E. (2021). *Enemy Surrounding Inspired Optimisation Algorithm: Introduction and Tests*. W: *Innovation Management and Sustainable Economic Development in the Era of Global Pandemic. Proceedings of the 38th International Business Information Management Association Conference (IBIMA)*, red. K. S. Soliman. Sewilla: IBIMA, s. 3214–3226. https://ibima.org/accepted-paper/45447-2/
11. Baumgart, J. E., Sangho, B. (2021). *Studium przypadku skuteczności nowych metod optymalizacji roju w porównaniu do metod znanych*. *Studia i Materiały Informatyki Stosowanej*, T. 13, nr 1, s. 47–50. https://doi.org/10.34767/SIMIS.2021.01.06
12. Baumgart, J. E., Sangho, B. (2021). *Algorytm inspirowany polem walki – połączenie algorytmów numerycznych z ideą roju*. *Studia i Materiały Informatyki Stosowanej*, T. 13, nr 2, s. 26–31. https://doi.org/10.34767/SIMIS.2021.02.05
13. Baumgart, J. E. (2019). *Implementacja środowiska testowego w języku R*. *Studia i Materiały Informatyki Stosowanej*, T. 11, nr 1, s. 21–26. https://doi.org/10.34767/SIMIS.2019.01.04

Spośród powyższych prac cykl publikacji dotyczących algorytmów inspirowanych operacjami wojennymi (m.in. *Enemy Surrounding Inspired Optimisation Algorithm*, *The Elephant in the Room* oraz *Numidian Swarm Riders*) dokumentuje kolejne etapy rozwoju koncepcji, której zwieńczeniem jest opisany w niniejszej rozprawie Ancient Battlefield Optimizer (ABO) wraz z wariantem CommanderABO.

ZAŁĄCZNIKI

PSEUDOKODY ALGORYTMÓW

Niniejsza sekcja zawiera kompletne pseudokody algorytmów składających się na Ancient Battlefield Optimizer (ABO).

Algorytm główny ABO

Algorytm 5 przedstawia główną pętlę optymalizacyjną ABO, integrującą system jednostek, taktyk, formacji, systemu honorów, rozpoznania (reconnaissance) oraz Commander AI opartego na Q-learningu.

Algorytm 5. Ancient Battlefield Optimizer – pętla główna

Require: Funkcja celu $f : \mathbb{R}^d \to \mathbb{R}$ (minimalizacja), wymiar d, granice $[\mathbf{lb}, \mathbf{ub}]$, rozmiar populacji n, maks. iteracji T, skład armii $\{(\text{typ}_k, n_k)\}$
Ensure: Najlepsza pozycja $\mathbf{x}^*$ i wartość f^*

// Faza inicjalizacji
Utwórz armię A z jednostkami: dla każdego typu k utwórz n_k jednostek klasy typ_k {Tabela 4}
for każda jednostka $u \in A$ **do**
    $\mathbf{x}_u \leftarrow \text{CoordinateStratifiedInit}(d, \mathbf{lb}, \mathbf{ub})$ {Inicjalizacja po współrzędnych, Alg. 17}
    $f_u \leftarrow f(\mathbf{x}_u)$; $\mathbf{x}_u^{pbest} \leftarrow \mathbf{x}_u$; $f_u^{pbest} \leftarrow f_u$
end for
$\mathbf{x}^* \leftarrow \arg\min_{u \in A} f_u$; $f^* \leftarrow \min_{u \in A} f_u$
Inicjalizuj FormationManager (grupy jednostek wg typu)
Inicjalizuj ReconnaissanceIntelligence(d, $[\mathbf{lb}, \mathbf{ub}]$, G) {Alg. 14, G = rozdzielczość siatki}
Inicjalizuj CommanderAI(A, α=0,1, γ=0,9, ϵ: zależne od trybu) {Alg. 7}
Załaduj Q-tabelę z pliku (uczenie transferowe, domyślnie tryb `full`) {Alg. 12}
Ustaw taktykę początkową (np. Phalanx); improved $\leftarrow$ False
// Główna pętla optymalizacyjna
for $t = 0$ **to** $T-1$ **do**
    improved $\leftarrow$ False
    // Krok 1: Aplikacja bieżącej taktyki – modyfikacja parametrów jednostek
    tactic.apply(A.units, $\mathbf{x}^*$, t, T) {Alg. 13}
    // Krok 2: Commander AI – obserwacja stanu i ewentualna zmiana taktyki/formacji
    (tactic, action, reward) $\leftarrow$ Commander.step(t, T) {Alg. 7}
    // Krok 3: Aktualizacja formacji
    jeśli $t \bmod 5 = 0$ **to** FormationManager.organize_units(A.units)
    FormationManager.update_formation_targets($\mathbf{x}^*$, tactic.name, t, T)
    unit_targets $\leftarrow$ FormationManager.calculate_unit_target_positions(A.units)
    // Krok 4: Udostępnianie wiedzy od bohaterów
    A.share_knowledge() {Bohaterowie wpływają na pobliskie jednostki}
    // Krok 5: Aktualizacja pozycji jednostek
    $p \leftarrow t/T$ {Postęp optymalizacji $\in [0,1]$}
    for każda aktywna jednostka $u \in A$ **do**
        MoveUnit(u, A, f, t, T, unit_targets) {Alg. 6, strategia zależna od typu}
        recon.update_with_position($\mathbf{x}_u$, f_u, t, T) {Alg. 14}
    end for

// Krok 6: Aktualizacja globalnego najlepszego i systemu honorów
for każda jednostka $u \in A$ **do**
  if $f_u^{pbest} < f^*$ **then**
    $\mathbf{x}^* \leftarrow \mathbf{x}_u^{pbest}$; $f^* \leftarrow f_u^{pbest}$; improved $\leftarrow$ True
  end if
end for
Aktualizuj honor wszystkich jednostek (Alg. 16) {Bohaterowie, dezerterzy}
// Krok 7: Adaptacyjne zagęszczanie siatki zwiadowczej (CommanderABO)
if $t > 30$ **and** $f^* < 0{,}1$ **and** rekonesans aktywny **then**
  if $t \mod 10 = 0$ **then**
    recon.refine_grid_globally(factor=1,2)
  end if
  if $t \mod 5 = 0$ **then**
    recon.refine_grid_around_best($\mathbf{x}^*$, factor=2,0) {lokalne zagęszczenie}
  end if
end if
// Uwaga: w trybie benchmarkowym Q-tabela jest readonly; w trybie treningowym Commander.step aktualizuje Q
end for
return $\mathbf{x}^*$, f^*

Obliczanie prędkości jednostki

Algorytm 6 przedstawia dyspozycję ruchu jednostki. W przeciwieństwie do klasycznych algorytmów rojowych, w których wszystkie osobniki stosują tę samą formułę prędkości, ABO deleguje ruch do *strategii specyficznej dla typu jednostki*. Każdy typ posiada odrębny algorytm ruchu inspirowany historycznym zachowaniem danej formacji wojskowej (Tabela 88). Wspólnym elementem jest końcowa korekta pozycji wynikająca z przynależności do formacji (elastic-band formation pull).

Algorytm 6 Aktualizacja pozycji jednostki (MoveUnit) – dyspozycja do strategii typowej

Require: Jednostka u, armia A, funkcja celu f, iteracja t, maks. T, cele formacji unit_targets
Ensure: Zaktualizowana pozycja $\mathbf{x}_u$

// Dyspozycja do strategii ruchu wg typu jednostki
strategy ← STRATEGY_MAP[u.unit_type] {Tabela 88}
strategy.move(u, A, f, t, T, unit_targets)
// Każda strategia wewnętrznie:
    1. Odczytuje parametry taktyczne z u.`_tactic_params` (ustawione przez Alg. 13)
    2. Oblicza wektor wpływu bohaterów: $\mathbf{h} \leftarrow w_h \cdot u$.hero_influence
    3. Generuje kandydatów wg algorytmu typowego (patrz niżej)
    4. Aktualizuje $\mathbf{x}_u$, f_u, $\mathbf{x}_u^{pbest}$ jeśli nastąpiła poprawa
    5. Stosuje elastic-band formation pull (jeśli $u \in$ unit_targets)
// Algorytmy ruchu per typ:
Ciężka piechota (Simplex March): przeszukiwanie współrzędnościowe (coordinate descent)
    z krokiem $\delta = 0{,}05 \cdot SR \cdot \text{step\_decay}(p) \cdot$ `step_scale`, po jednej osi na iterację.
    Co 10% iteracji: krok Neldera-Meada (refleksja przez centroid $\mathbf{x}^{pbest}$ i $\mathbf{x}^*$).
    Fallback: krok gaussowski w kierunku $\mathbf{x}_u^{pbest}$.
// Reset antystagnacyjny (jeśli $c_{\text{stag}} \geq k_{\text{anti\_stag}}$, domyślnie 30):
    $\boldsymbol{\eta} \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{I}) \cdot 0{,}05 \cdot \text{search\_range}/\text{dim\_scale}(d)$
    $\boldsymbol{\eta} \leftarrow \text{cap\_norm}(\boldsymbol{\eta}, 0{,}3 \cdot \text{search\_range})$
    $\mathbf{x}_i \leftarrow \text{clip}(\mathbf{g} + \boldsymbol{\eta}, \mathbf{b}_{\min}, \mathbf{b}_{\max})$; $c_{\text{stag}} \leftarrow 0$
Lekka piechota (Skirmisher DE): ewolucja różnicowa DE/rand/1.
    Mutacja: $\mathbf{v} = \mathbf{x}_{r1} + F \cdot (\mathbf{x}_{r2} - \mathbf{x}_{r3})$, $F \sim U($`de_f_range`$)$.
    Krzyżowanie binomialne z $CR =$ `de_cr`.
    Selekcja zachłanna + opcjonalny dash w kierunku $\mathbf{x}^*$.
Łucznicy (Volley Fire): wielopunktowe próbkowanie kierunkowe.
    Wystrzel `n_arrows` strzał w losowych kierunkach z biasem ku $\mathbf{x}^*$ i $\mathbf{x}_u^{pbest}$.
    Odległość ~ Exp(`max_range`/3). Przemieszczenie ku najlepszemu trafionemu punktowi.
Kawaleria (Charge & Wheel): lot Lévy'ego [77] z persystencją kierunkową.
    Krok: |Lévy(`levy_alpha`)| $\cdot\, s_u \cdot SR \cdot 0{,}1 \cdot$ `charge_distance_scale`.
    Po 3–7 krokach: manewr zwrotny (wheel) – rotacja kierunku szarży o losowy kąt.
    Z prawdopodobieństwem `feign_prob`: pozorowany odwrót (odwrócenie kierunku).
Rydwany (Drive-Through): trajektoria z pędem Cauchy'ego.
    Wektor prędkości: $\mathbf{v} \leftarrow \mu \cdot \mathbf{v}_{prev} + (1{-}\mu)(\text{attraction} + \text{Cauchy}(d) \cdot$ `cauchy_scale`$)$.
    Ewaluacja `n_trajectory_points` punktów wzdłuż trajektorii; wybór najlepszego.
Słonie wojenne (Cauchy Basin Hopping): lokalne udoskonalanie + stampede.
    Krok lokalny: $\mathcal{N}(\mathbf{0}, 0{,}05 \cdot SR \cdot$ `local_step_scale`$)$.
    Z prawdop. `stampede_prob`: skok Cauchy'ego z akceptacją Metropolisa [80]
    ($T_{accept} = 1 - 0{,}8p$, chyba że `greedy_only`=True).

Commander AI – decyzje taktyczne z Q-learningiem

Commander AI jest meta-optymalizatorem, który na podstawie stanu optymalizacji wybiera taktykę. Algorytm 7 opisuje pojedynczy krok decyzyjny. Stan jest abstrakcyjną reprezentacją pięciu cech (Alg. 8), co daje $3^5 = 243$ możliwych stanów. Przestrzeń akcji to 6 taktyk (formacje dobierane automatycznie przez `tactic_formation_map`).

Tabela 88. Mapowanie typów jednostek na strategie ruchu (STRATEGY_MAP)

Typ jednostki	Strategia ruchu	Inspiracja algorytmiczna
Ciężka piechota	HeavyInfantryStrategy	Coordinate Descent + Nelder-Mead
Lekka piechota	LightInfantryStrategy	DE/rand/1 z selekcją zachłanną
Łucznicy	ArcherStrategy	Multi-Point Remote Sampling
Kawaleria	CavalryStrategy	Lévy Flight + Directional Persistence
Rydwany	ChariotStrategy	Cauchy Momentum + Trajectory Sampling
Słonie wojenne	WarElephantStrategy	Cauchy Basin Hopping + Metropolis

Algorytm 7. Commander AI – krok decyzyjny (Commander.step)

Require: Iteracja t, maks. iteracji T
Ensure: Zastosowana taktyka, akcja, nagroda

$r \leftarrow 0$; $a \leftarrow a_{prev}$; tactic $\leftarrow A$.tactic {Wartości domyślne dla pierwszej iteracji i kroków bez zmiany}
if $f^*_{interval\_start}$ = null **then**
  $f^*_{interval\_start} \leftarrow f^*_{current}$
end if
// Adaptacyjny interwał taktyczny
$p \leftarrow t/T$
if $p < 0{,}3$ **then**
  $\Delta t \leftarrow 5$
else if $p < 0{,}7$ **then**
  $\Delta t \leftarrow 8$
else
  $\Delta t \leftarrow 15$
end if
decision_point $\leftarrow (t \bmod \Delta t = 0) \vee (t = 0)$
if decision_point = False **then**
  pomiń zmianę taktyki; $a \leftarrow a_{prev}$; tactic $\leftarrow A$.tactic
else
  // Oblicz bieżący stan i nagrodę za poprzedni interwał decyzyjny
  $s' \leftarrow$ get_state(t, T) {Alg. 8}
  if $s_{prev} \neq$ null **and** $a_{prev} \neq$ null **then**
    $r \leftarrow$ calculate_reward$(s_{prev}, s', f^*_{interval_start}, f^*_{current})$ {Alg. 11}
    if tryb treningowy (`readonly`=False) **then**
      Zapisz doświadczenie $(s_{prev}, a_{prev}, r, s')$ do pamięci
      update_q_value$(s_{prev}, a_{prev}, r, s')$ {Alg. 10}
    end if
  end if
  // Wybierz nową akcję (epsilon-greedy)
  $a \leftarrow$ select_action(s') {Alg. 9}
  // Stabilizacja – nie zmieniaj bez jasnej przewagi Q
  if $a_{prev} \neq$ null **and** $Q[s', a] - Q[s', a_{prev}] < 0{,}1$ **then**
    $a \leftarrow a_{prev}$
  end if
  // Zastosuj akcję
  tactic_idx $\leftarrow a$ {Po stabilizacji, aby akcja i taktyka były spójne}

tactic ← tactic_pool[tactic_idx]
A.set_tactic(tactic)
Formacja dobierana automatycznie z `tactic_formation_map`
// Zanik epsilon zależny od trybu
if `readonly`=True **then**
    $\epsilon \leftarrow \max(0{,}03,\ 0{,}12 \cdot (1 - t/T))$ {benchmark: 12% → 3%}
else
    $\epsilon \leftarrow \max(0{,}1,\ \epsilon \cdot 0{,}99)$ {trening: 30% → 10%}
end if
// Experience replay (tylko tryb treningowy)
if tryb treningowy **and** |pamięć| ≥ 32 **then**
    Wylosuj 32 doświadczenia z pamięci (bez zwracania)
    for każde (s_i, a_i, r_i, s'_i) w próbce **do**
        update_q_value(s_i, a_i, r_i, s'_i)
    end for
end if
$s_{prev} \leftarrow s'$; $a_{prev} \leftarrow a$; $f^*_{interval\_start} \leftarrow f^*_{current}$
end if
// Uwaga: $\alpha_0 = 0{,}1$ **to wartość początkowa (adaptowana w treningu, końcowo** ≈0,071**); w trybie readonly brak update_q_value, replay i zapisu Q-tabeli**
return tactic, a, r

Algorytm 8 Commander AI – konstrukcja stanu

Require: Iteracja t, maks. iteracji T
Ensure: Stan s jako krotka 5 cech dyskretnych (243 możliwych stanów)

// Cecha 1: Faza optymalizacji (3 kategorie)
$p \leftarrow t/T$
if $p < 0{,}3$ **then**
    phase ← "early"
else if $p < 0{,}7$ **then**
    phase ← "mid"
else
    phase ← "late"
end if
// Cecha 2: Stagnacja (3 kategorie)
$\bar{c} \leftarrow \frac{1}{|A|} \sum_{u \in A} u$.iterations_without_improvement
if $\bar{c} < 0{,}4$ **then**
    stagnation ← "low"
else if $\bar{c} < 0{,}8$ **then**
    stagnation ← "moderate"
else
    stagnation ← "severe"
end if
// Cecha 3: Momentum – tempo poprawy fitness (3 kategorie)
$\mu \leftarrow$ średnia relatywna poprawa w oknie 10 iteracji
if $\mu \geq 0{,}015$ **then**
    momentum ← "fast"
else if $\mu \geq 0{,}003$ **then**
    momentum ← "slow"
else
    momentum ← "stagnant"
end if
// Cecha 4: Różnorodność populacji (3 kategorie)
$\sigma \leftarrow$ mean(std(positions, axis=0)) / search_range
if $\sigma > 0{,}07$ **then**
    diversity ← "spread"
else if $\sigma > 0{,}04$ **then**
    diversity ← "moderate"
else
    diversity ← "converged"
end if
// Cecha 5: Odsetek bohaterów (3 kategorie)
$r_h \leftarrow |\{u \in A : u.\text{hero\_status}\}| \,/\, |A|$
if $r_h > 0{,}08$ **then**
    heroes ← "many"
else if $r_h > 0$ **then**
    heroes ← "some"
else
    heroes ← "none"
end if
return $s = (\text{phase}, \text{stagnation}, \text{momentum}, \text{diversity}, \text{heroes})$

Algorytm 9 Commander AI – selekcja akcji (ϵ-greedy)

Require: Stan s, parametr eksploracji ϵ, Q-tabela Q
Ensure: Akcja a = tactic_idx $\in \{0, \ldots, 5\}$
if rand() $< \epsilon$ **then**
  $a \leftarrow$ losowa akcja z przestrzeni akcji {Eksploracja}
else
  $Q_s \leftarrow Q[s]$ {Wektor Q-wartości dla stanu s}
  $q_{max} \leftarrow \max_{a'} Q_s[a']$
  $\mathcal{A}_{best} \leftarrow \{a' : Q_s[a'] = q_{max}\}$ {Rozstrzyganie remisów}
  $a \leftarrow$ losowy element z $\mathcal{A}_{best}$ {Eksploatacja}
end if
return a

Algorytm 10 Commander AI – aktualizacja Q-tabeli (Q-learning)

Require: Stan s, akcja a, nagroda r, następny stan s'
Require: $\alpha = 0{,}1$ (początkowy learning rate, adaptowany w treningu), $\gamma = 0{,}9$ (discount factor)
$Q_{old} \leftarrow Q[s][a]$
$\max Q' \leftarrow \max_{a'} Q[s'][a']$
$Q_{target} \leftarrow r + \gamma \cdot \max Q'$
// Reguła aktualizacji Q-learning
$Q[s][a] \leftarrow Q_{old} + \alpha \cdot (Q_{target} - Q_{old})$
// Aktualizacja statystyk wizyt
visit_count$[s][a] \leftarrow$ visit_count$[s][a]$ +1
confidence$[s][a] \leftarrow \min(1{,}0,$ visit_count$[s][a]$ / 20)

Algorytm 11 Commander AI – obliczanie nagrody

Require: Stara fitness f_{old}, nowa fitness f_{new}
Ensure: Nagroda $r \in \mathbb{R}$
// Relatywna poprawa fitness
$\Delta_{rel} \leftarrow (f_{old} - f_{new}) \,/\, \max(|f_{old}|, 10^{-9})$
// Nagroda za poprawę (obcięta do [-2, 5])
$r_{improve} \leftarrow \text{clip}(10{,}0 \cdot \Delta_{rel}, -2{,}0, 5{,}0)$
// Kara za stagnację
$r_{stag} \leftarrow -0{,}3$ jeśli $\Delta_{rel} \leq 0$, w przeciwnym razie 0
// Uwaga: wcześniejsze wersje zawierały składniki pomocnicze
// (health, diversity, recon), usunięte z powodu reward hacking
return $r \leftarrow r_{improve} + r_{stag}$

Transfer Learning – adaptacyjne łączenie Q-tabel

Algorytm 12 opisuje mechanizm transfer learningu, który pozwala na przenoszenie wiedzy między kolejnymi uruchomieniami optymalizacji. Implementacja oferuje cztery tryby: `full` (pełne zastąpienie Q-wartości), `weighted` (ważone średnie wg konfidencji), `preserve_high_confidence` (tylko wartości o wysokiej konfidencji) oraz `adaptive` (adaptacyjne progi). Domyślny tryb to `full` (proste nadpisanie), poniżej przedstawiono tryb `adaptive`, najbardziej zaawansowany:

Algorytm 12 Transfer Learning – ładowanie Q-tabeli (tryb `adaptive`)

Require: Zapisana Q-tabela Q_{saved}, konfidencja C_{saved}, bieżąca Q-tabela Q
Ensure: Zaktualizowana Q

for każdy stan s w Q_{saved} **do**
  for każda akcja a w $Q_{saved}[s]$ **do**
    $c \leftarrow C_{saved}[s][a]$; $q_{saved} \leftarrow Q_{saved}[s][a]$; $q_{curr} \leftarrow Q[s][a]$
    if $c \geq 0{,}8$ **then**
      $Q[s][a] \leftarrow q_{saved}$ {Bardzo wysoka konfidencja: pełne zastąpienie}
    else if $c \geq 0{,}3$ **then**
      $w \leftarrow (c - 0{,}3)/0{,}5$ {Skala $[0{,}3, 0{,}8] \rightarrow [0, 1]$}
      $Q[s][a] \leftarrow w \cdot q_{saved} + (1 - w) \cdot q_{curr}$ {Ważone uśrednianie}
    else
      $Q[s][a] \leftarrow 0{,}2 \cdot q_{saved} + 0{,}8 \cdot q_{curr}$ {Niska konfidencja}
    end if
  end for
end for
return Q, ϵ_{saved}

Operatory taktyczne

ABO oferuje 6 taktyk, z których każda modyfikuje parametry ruchu jednostek w inny sposób. Algorytm 13 przedstawia ogólny schemat działania taktyk. Każda taktyka ustawia słownik `_tactic_params` na każdej jednostce, zawierający klucze specyficzne dla strategii ruchu danego typu (por. Alg. 6). Strategie ruchu odczytują te parametry i dostosowują swoje zachowanie.

Algorytm 13 Operatory taktyczne – ustawienie parametrów strategii ruchu

Require: Jednostki U, najlepsze globalne $\mathbf{x}^*$, iteracja t, maks. T
$p \leftarrow t/T$ {Postęp optymalizacji}
Phalanx (zwarta eksploatacja):
  Ciężka piechota: `step_scale`=0,3, `no_simplex`=True, `formation_pull`=0,8
  Lekka piechota: `de_f_range`=(0,1, 0,3), `de_cr`=0,3
  Kawaleria: `levy_alpha`=2,0, `max_charge_steps`=2, `charge_distance_scale`=0,5
  Łucznicy: `n_arrows`=3, `range_scale`=0,5
  Rydwany: `chariot_momentum`=0,5, `chariot_speed_scale`=0,5
  Słonie: `stampede_prob`=0,02, `local_step_scale`=0,5
Oblique Order (asymetryczne przeszukiwanie):
  Skrzydło silne (ciężka piechota, słonie): agresywna eksploatacja
    Ciężka piechota: `step_scale`=2,0; Słonie: `stampede_prob`=0,25, `cauchy_scale`=0,6
  Centrum (lekka piechota, rydwany): utrzymanie pozycji
    Lekka piechota: `de_use_best`=True, `de_f_range`=(0,3, 0,6)
  Skrzydło odmówione (kawaleria, łucznicy): eksploracja
    Kawaleria: `levy_alpha`=1,0, `charge_distance_scale`=2,0, `feign_prob`=0,15
    Łucznicy: `n_arrows`=8, `range_scale`=2,0, `direction_bias`= $\mathbf{d}_\perp$
Center Penetration (koncentracja na obiecujących regionach):
  Ciężka piechota: `direct_march`=True {Marsz wprost ku $\mathbf{x}^*$}
  Słonie: `stampede_prob`=0,2, `stampede_direction`= $(\mathbf{x}^* - \mathbf{x}_u)/\|\cdot\|$, `greedy_only`=True
  Kawaleria: `charge_direction_override`= $(\mathbf{x}^* - \mathbf{x}_u)/\|\cdot\|$, `charge_distance_scale`=1,5

  Lekka piechota: `de_use_best`=True, `de_f_range`=(0,4, 0,8)
  Łucznicy: `n_arrows`=4, `range_scale`=0,6
Flanking Maneuver (eksploracja peryferyjna):
  Oblicz $\mathbf{d}_\perp$ – kierunek prostopadły do osi centroid $\rightarrow \mathbf{x}^*$
  Kawaleria: `levy_alpha`=1,2, `charge_distance_scale`=2,0, `charge_direction`= $\mathbf{d}_\perp$
  Lekka piechota: `de_f_range`=(0,8, 1,5), `de_cr`=0,9
  Ciężka piechota: `step_scale`=0,1, `formation_pull`=0,8 {Kowadło – trzyma centrum}
  Łucznicy: `n_arrows`=5, `range_scale`=1,5, `direction_bias`= $\mathbf{d}_\perp$
  Słonie: `stampede_prob`=0,2, `stampede_direction`= $\mathbf{d}_\perp$
Surrounding (okrążanie optimum globalnego):
  Przypisz pozycje docelowe na okręgu wokół $\mathbf{x}^*$ z promieniem $r(p)$
  $r(p) = 0{,}5 + 1{,}0 \cdot (1 - p^2)$ {Promień zmniejsza się z postępem}
  Ciężka piechota: `step_scale`=0,8, `formation_pull`=0,5
  Lekka piechota: `de_f_range`=(0,4, 0,8), `formation_pull`=0,5
  Łucznicy: `n_arrows`=5, `range_scale`=1,2, `formation_pull`=0,5
  Kawaleria: `levy_alpha`=1,3, `charge_distance_scale`=1,2, `formation_pull`=0,4
  Rydwany: `chariot_speed_scale`=1,0, `formation_pull`=0,4
  Słonie: `stampede_prob`=0,1+0,15p, `cauchy_scale`=0,25, `formation_pull`=0,4
Scouting (ucieczka z minimów lokalnych):
  $T_{SA} = \max(0{,}3,\ 1{,}0 - 0{,}6p)$ {Temperatura, harmonogram chłodzenia}
  Kawaleria: `levy_alpha`=1,0, `charge_distance_scale`=3,0, `max_charge_steps`=100
  Słonie: `stampede_prob`=0,4, `cauchy_scale`=0,9, `temperature`=1,0
  Łucznicy: `n_arrows`=8, `range_scale`=3,0, `suppress_bias`=True
  Lekka piechota: `de_f_range`=(1,0, 2,0), `de_cr`=0,9, `accept_worse_prob`=0,1
  Ciężka piechota: co 3. jednostka zwiaduje (`step_scale`=3,0, `random_direction`=True),
    pozostałe kotwiczą (`step_scale`=0,5, `formation_pull`=0,4)
  Rydwany: `chariot_momentum`=0,95, `chariot_speed_scale`=2,0, `cauchy_scale`=0,15
  Jeśli stagnacja > 15: globalny restart 30% jednostek

System rozpoznania (Reconnaissance Intelligence)

System rozpoznania (Alg. 14) dyskretyzuje przestrzeń poszukiwań na siatkę G^d komórek i śledzi historię wizyt oraz wartości fitness. Na tej podstawie identyfikuje obiecujące regiony, szacuje gradienty lokalne i generuje rekomendacje kierunku ruchu.

Algorytm 14 Reconnaissance Intelligence – aktualizacja i identyfikacja regionów

Require: Pozycja $\mathbf{x}$, fitness f, iteracja t, maks. iteracji T, rozdzielczość siatki G

// Mapowanie pozycji na komórkę siatki
$\mathbf{g} \leftarrow \lfloor (\mathbf{x} - \mathbf{lb}) \,/\, (\mathbf{ub} - \mathbf{lb}) \cdot G \rfloor$ {Współrzędne siatki, obcięte do $[0, G-1]$}
// Aktualizacja statystyk regionu
if $\mathbf{g} \notin$ explored_regions **then**
    Inicjalizuj region: visits = 0, min_fitness = ∞, positions = []
end if
region $\leftarrow$ explored_regions[$\mathbf{g}$]
region.visits $\leftarrow$ region.visits +1
region.min_fitness $\leftarrow$ min(region.min_fitness, f)
region.max_fitness $\leftarrow$ max(region.max_fitness, f)
Dodaj $(\mathbf{x}, f)$ do historii regionu (zachowaj ostatnich 5)
region.last_visit $\leftarrow t$
// Estymacja gradientu lokalnego (jeśli ≥ 3 pozycje w regionie)
if |region.positions| ≥ 3 **then**
    Rozwiąż $\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{f}\|^2$ (metoda najmniejszych kwadratów)
    region.gradient $\leftarrow \mathbf{w}_{1:d}$
end if
// Aktualizacja obiecujących regionów (co kilka iteracji)
$\bar{f} \leftarrow$ średnia min_fitness po wszystkich regionach
for każdy region c z min_fitness $< \bar{f}$ **do**
    $s_{fitness} \leftarrow (\bar{f} - c.\text{min\_fitness})/(|\bar{f}| + 10^{-10})$
    $s_{expl} \leftarrow 1/(1 + c.\text{visits})$ {Mniej odwiedzane = ciekawsze}
    $s_{grad} \leftarrow 0$ jeśli c.gradient jest niezdefiniowany, w przeciwnym razie $\min(1,\ \|\text{c.gradient}\|/10)$
    score $\leftarrow 0{,}6 \cdot s_{fitness} + 0{,}3 \cdot s_{expl} + 0{,}1 \cdot s_{grad}$
end for
Sortuj regiony malejąco wg score; zachowaj 10 najlepszych
// Elitarne regiony: 5 regionów o najniższej min_fitness
elite_regions $\leftarrow$ top-5 wg min_fitness

Algorytm 15 Reconnaissance – rekomendacja kierunku ruchu

Require: Pozycja bieżąca $\mathbf{x}$, balans eksploracja/eksploatacja $\beta \in [0, 1]$
Ensure: Wektor kierunku $\mathbf{d} \in \mathbb{R}^d$ (znormalizowany)
// Komponent 1: Kierunek gradientu globalnego (eksploatacja)
if $\|\text{global\_gradient}\| > 10^{-10}$ **then**
  $\mathbf{d}_{grad} \leftarrow -\text{global\_gradient}/\|\text{global\_gradient}\|$ {Ujemny = minimalizacja}
else
  $\mathbf{d}_{grad} \leftarrow \mathbf{0}$
end if
// Komponent 2: Kierunek do obiecujących regionów (zbalansowany)
$\mathbf{d}_{prom} \leftarrow \sum_{(c,w)\in\text{promising}} w \cdot (\widehat{\text{center}(c)} - \mathbf{x})$ lub losowy wektor jednostkowy, jeśli brak kandydatów
// Komponent 3: Kierunek do najmniej zbadanych regionów (eksploracja)
$\mathbf{d}_{expl} \leftarrow \sum_{c\in\text{least\_visited}} \frac{1}{1+\text{visits}(c)} \cdot (\widehat{\text{center}(c)} - \mathbf{x})$ lub losowy wektor jednostkowy, jeśli brak kandydatów
// Komponent 4: Kierunek do elitarnego regionu (eksploatacja)
$\mathbf{d}_{elite} \leftarrow (\widehat{\text{center}(\text{elite}[0])} - \mathbf{x})$ jeśli elite istnieje, w przeciwnym razie $\mathbf{d}_{grad}$
// Złożenie ważone
$\mathbf{d} \leftarrow \beta \cdot (0{,}5 \cdot \mathbf{d}_{grad} + 0{,}5 \cdot \mathbf{d}_{elite}) + 0{,}5 \cdot \mathbf{d}_{prom} + (1 - \beta) \cdot \mathbf{d}_{expl}$
if $\|\mathbf{d}\| > 10^{-10}$ **then**
  $\mathbf{d} \leftarrow \mathbf{d}/\|\mathbf{d}\|$
else
  $\mathbf{d} \leftarrow$ losowy wektor jednostkowy
end if
return $\mathbf{d}$

System honorów i inicjalizacja jednostek

Algorytm 16 System honorów – aktualizacja statusu jednostki

Require: Jednostka u, czy poprawiła globalne najlepsze: improved_global
Require: Współczynnik zaniku $\eta = 0{,}95$, próg bohatera $\theta_H = 6{,}0$, próg dezertera $\theta_R = -10{,}0$
// Zanik honoru
u.honor $\leftarrow u$.honor $\cdot\, \eta$
if improved_global **then**
  u.honor $\leftarrow u$.honor + 10,0 {Duży bonus za poprawę globalnego najlepszego}
  u.global_improvements $\leftarrow u$.global_improvements + 1
  u.iterations_without_improvement $\leftarrow 0$
else
  u.honor $\leftarrow u$.honor − 0,2 − 0,1 $\cdot$ min(5, u.iter_without_improvement)
end if
// Aktualizacja statusu
u.hero_status $\leftarrow (u.\text{honor} \geq \theta_H)$
u.runagate_status $\leftarrow (u.\text{honor} \leq \theta_R) \wedge (u.\text{iter\_without\_imp.} \geq 10) \wedge (u.\text{global\_imp.} = 0)$
// Zasięg przywództwa bohatera
u.leadership_range $\leftarrow 1{,}0 + 0{,}5 \cdot \min(5{,}0,\ \max(0,\ u.\text{honor}/10{,}0))$
// Jeśli dezerter – dezaktywacja jednostki
if u.runagate_status **then**
  u.active $\leftarrow$ False
end if

Algorytm 17 Inicjalizacja jednostki – stratyfikacja po współrzędnych z perturbacją

Require: Wymiar d, granice $[\mathbf{lb}, \mathbf{ub}]$, typ jednostki
Ensure: Jednostka u z pozycją początkową i parametrami
// Stratyfikacja po współrzędnych pojedynczego wektora z perturbacją Gaussowską
// Uwaga: nie jest to Latin Hypercube Sampling dla całej populacji
for $i = 1$ **to** d **do**
  $\xi_i \leftarrow \text{rand}()$ {$\xi_i \in [0, 1)$}
  $p_i \leftarrow \frac{i-1+\xi_i}{d}$ {Segment przypisany współrzędnej i}
  $x_i \leftarrow lb_i + p_i \cdot (ub_i - lb_i)$
end for
$\mathbf{x} \leftarrow \text{clip}(\mathbf{x} + \mathcal{N}(\mathbf{0}, 0{,}05 \cdot (\mathbf{ub} - \mathbf{lb})),\ \mathbf{lb},\ \mathbf{ub})$
// Przypisanie parametrów wg typu (Tabela 4)
Ustaw $(s, e^{expl}, e^{expt}, m, i)$ zgodnie z typem jednostki
Ustaw parametry formacji: (typ formacji, zwartość, dyscyplina)
u.honor $\leftarrow 0$; u.active $\leftarrow$ True

Algorytm 18 Elastic-band formation pull – przyciąganie jednostki ku pozycji formacyjnej

Require: Jednostka u z pozycją $\mathbf{x}_u$, mapa celów formacyjnych $\mathcal{T}$, iteracja t, maks. iteracji T
if $u \notin \mathcal{T}$ **then**
  return {Brak przypisanej pozycji formacyjnej}
end if
$\mathbf{t}_u \leftarrow \mathcal{T}[u]$ {Pozycja docelowa w formacji}
$\mathbf{d} \leftarrow \mathbf{t}_u - \mathbf{x}_u$; $\ell \leftarrow \|\mathbf{d}\|$
if $\ell < 10^{-12}$ **then**
  return {Już w pozycji}
end if
$r \leftarrow \text{search\_range}(u)$ {Zasięg przeszukiwania zależny od typu}
$p \leftarrow t/T$ {Postęp optymalizacji}
$\tau \leftarrow r \cdot 0{,}15 \cdot (1{,}5 - 0{,}5 \cdot p)$ {Próg aktywacji wzmocnienia (Rów. 4.10)}
$\delta \leftarrow u$.formation_discipline {Domyślnie 0,7}
if u.hero_status **then**
  $\delta \leftarrow 0{,}7 \cdot \delta$ {Bohaterowie: redukcja 30%}
end if
if u.tactic_params[`formation_pull`] $\neq$ `None` **then**
  $\delta \leftarrow u$.tactic_params[`formation_pull`] {Taktyka nadpisuje}
end if
$\phi \leftarrow \delta \cdot 0{,}4$ {Bazowa siła przyciągania (Rów. 4.11)}
if $\ell > \tau$ **and** $\tau > 0$ **then**
  $\phi \leftarrow \min(0{,}8,\ \phi \cdot \ell/\tau)$ {Wzmocnienie elastyczne (Rów. 4.12)}
end if
$\mathbf{x}_u \leftarrow \mathbf{x}_u + \phi \cdot \mathbf{d}$ {Aktualizacja pozycji (Rów. 4.13)}
$\mathbf{x}_u \leftarrow \text{clip}(\mathbf{x}_u, \mathbf{lb}, \mathbf{ub})$

Algorytm 19. Manewr konsolidacji – adaptacyjne sondowanie osiowe wokół najlepszej pozycji z mechanizmem ekspansji/kontrakcji

Require: Pozycja startowa $\mathbf{x}_0$, funkcja celu f, granice $[\mathbf{lb}, \mathbf{ub}]$, wymiar d
Ensure: Udoskonalone rozwiązanie $\mathbf{x}^*$, f^*

Parametry: γ=2,0 (ekspansja), ρ=0,5 (kontrakcja), σ=0,5 (skurcz), M – maks. iteracji
$\mathbf{x}_{cur} \leftarrow \mathbf{x}_0$; $f_{cur} \leftarrow f(\mathbf{x}_{cur})$; $\mathbf{x}^* \leftarrow \mathbf{x}_{cur}$; $f^* \leftarrow f_{cur}$
$step \leftarrow$ `initial_step_size` $\cdot(ub - lb)$ dla skalarnych granic, w przeciwnym razie `initial_step_size`
$c_{\text{stag}} \leftarrow 0$; $\theta_{\text{stop}} \leftarrow \max(10, M/5)$
for $k = 0$ **to** $M - 1$ **do**
  improved $\leftarrow$ False
  for $j = 1$ **to** d **do**
    $\mathbf{x}^+ \leftarrow \mathbf{x}_{cur}$; $x_j^+ \leftarrow x_{cur,j} + step$ {Krok w kierunku $+j$}
    $\mathbf{x}^+ \leftarrow \text{clip}(\mathbf{x}^+, \mathbf{lb}, \mathbf{ub})$
    $f^+ \leftarrow f(\mathbf{x}^+)$
    if $f^+ < f_{cur}$ **then**
      $\mathbf{x}^{++} \leftarrow \mathbf{x}_{cur}$; $x_j^{++} \leftarrow x_{cur,j} + \gamma \cdot step$ {Próba ekspansji}
      $\mathbf{x}^{++} \leftarrow \text{clip}(\mathbf{x}^{++}, \mathbf{lb}, \mathbf{ub})$
      $f^{++} \leftarrow f(\mathbf{x}^{++})$
      Wybierz lepszy punkt z $\{\mathbf{x}^+, \mathbf{x}^{++}\}$; zaktualizuj $\mathbf{x}_{cur}$ i f_{cur}
      improved $\leftarrow$ True
      if $f_{cur} < f^*$ **then**
        $\mathbf{x}^* \leftarrow \mathbf{x}_{cur}$; $f^* \leftarrow f_{cur}$
      end if
      continue {Kod pomija wtedy kierunek ujemny dla tej osi}
    else
      $\mathbf{x}^- \leftarrow \mathbf{x}_{cur}$; $x_j^- \leftarrow x_{cur,j} - step$ {Krok w kierunku $-j$}
      $\mathbf{x}^- \leftarrow \text{clip}(\mathbf{x}^-, \mathbf{lb}, \mathbf{ub})$
      $f^- \leftarrow f(\mathbf{x}^-)$
      if $f^- < f_{cur}$ **then**
        $\mathbf{x}^{--} \leftarrow \mathbf{x}_{cur}$; $x_j^{--} \leftarrow x_{cur,j} - \rho \cdot step$ {Próba kontrakcji}
        $\mathbf{x}^{--} \leftarrow \text{clip}(\mathbf{x}^{--}, \mathbf{lb}, \mathbf{ub})$
        $f^{--} \leftarrow f(\mathbf{x}^{--})$
        Wybierz lepszy punkt z $\{\mathbf{x}^-, \mathbf{x}^{--}\}$; zaktualizuj $\mathbf{x}_{cur}$ i f_{cur}
        improved $\leftarrow$ True
        if $f_{cur} < f^*$ **then**
          $\mathbf{x}^* \leftarrow \mathbf{x}_{cur}$; $f^* \leftarrow f_{cur}$
        end if
      end if
    end if
  end for
  if improved **then**
    $step \leftarrow 1{,}1 \cdot step$; $c_{\text{stag}} \leftarrow 0$
  else
    $step \leftarrow \sigma \cdot step$; $c_{\text{stag}} \leftarrow c_{\text{stag}} + 1$
  end if
  if $step < 10^{-8}$ lub $c_{\text{stag}} \geq \theta_{\text{stop}}$ **then**
    break
  end if
  if k mod 10 = 9 oraz improved=False **then**

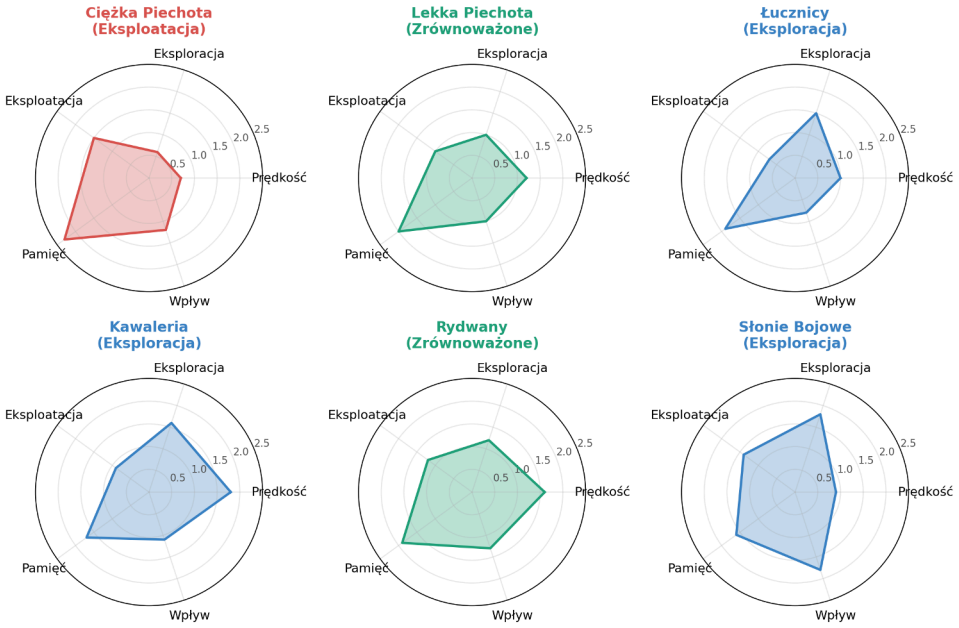
46: $\mathbf{d}_{rand} \leftarrow$ losowy wektor jednostkowy; $\mathbf{x}_{test} \leftarrow \text{clip}(\mathbf{x}_{cur} + step \cdot (1 - k/M) \cdot \mathbf{d}_{rand}, \mathbf{lb}, \mathbf{ub})$
47: **if** $f(\mathbf{x}_{test}) < f_{cur}$ **then**
48: $\mathbf{x}_{cur} \leftarrow \mathbf{x}_{test}$; $f_{cur} \leftarrow f(\mathbf{x}_{test})$; zaktualizuj $\mathbf{x}^*, f^*$ jeśli poprawiono
49: **end if**
50: **end if**
51: **end for**
52: **return** $\mathbf{x}^*, f^*$

WIZUALIZACJA ARMII I TAKTYK ABO

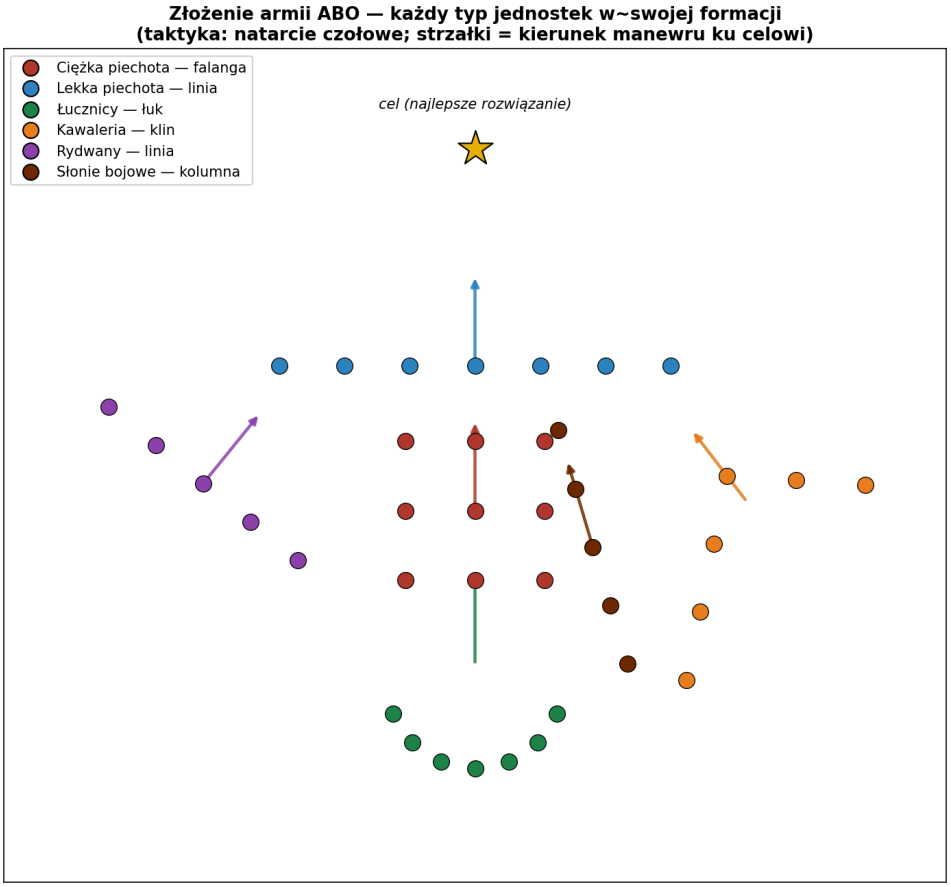
Poniższe rysunki przedstawiają strukturę typów jednostek, złożenie heterogenicznej armii ABO oraz jej zachowanie pod wpływem poszczególnych taktyk globalnych. Wizualizacje wygenerowano bezpośrednio z implementacji (`core/units.py` – domyślne formacje typów jednostek, `core/formations.py` – rozmieszczenie grup względem celu dla każdej taktyki), stanowiąc uzupełnienie katalogu formacji i taktyk z Rozdziału 3.

Rysunki 46 i 47 uzupełniają zestawienie parametrów behawioralnych z Tabeli 4 oraz Rysunku 5 (Rozdział 4) o dwa dodatkowe ujęcia: hierarchię ról oraz pełne profile radarowe poszczególnych typów jednostek.

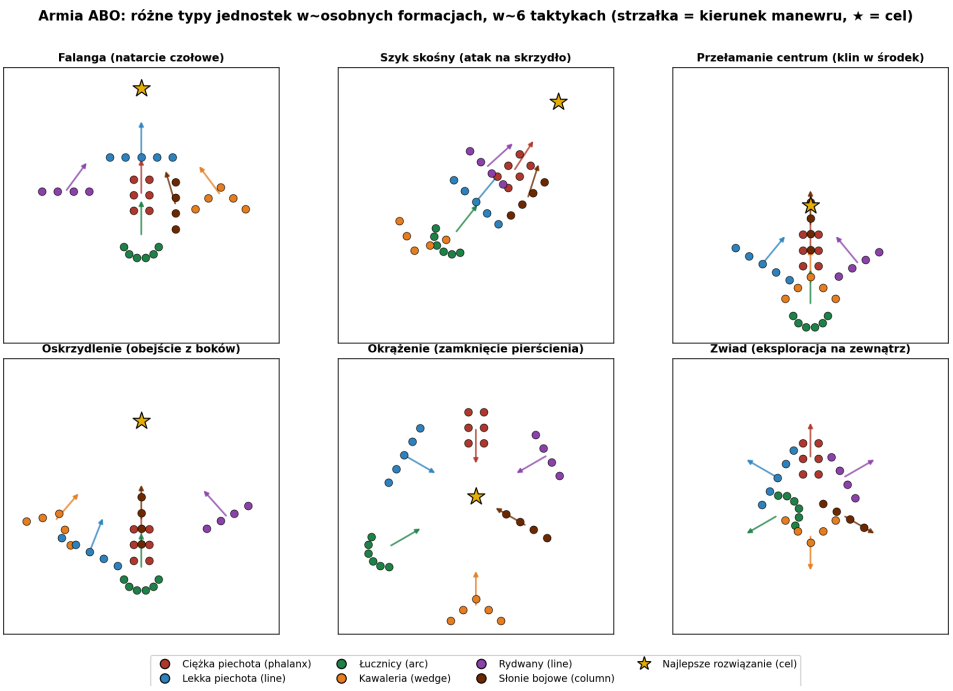
Rys. 46. Hierarchia typów jednostek w algorytmie ABO – podział według dominującej roli w kompromisie eksploracja–eksploatacja. Każdy panel zawiera pełen zestaw pięciu parametrów behawioralnych zgodny z Tabelą 4. Status *hero* oraz funkcja *scouting* są dynamicznymi rolami nakładanymi na powyższe sześć klas (Sekcje 4.2.4, 4.2.6), a nie odrębnymi typami jednostek.



Rys. 47. Profile pięciu parametrów behawioralnych (prędkość, eksploracja, eksploatacja, pamięć, wpływ) dla każdego z sześciu typów jednostek. Wykresy radarowe umożliwiają bezpośrednie porównanie profili w ujęciu pełnego wektora cech, uzupełniając pojedyncze zestawienie z Rysunku 5.



Rys. 48. Złożenie heterogenicznej armii ABO – każdy typ jednostek występuje w swojej domyślnej formacji (kolory wg legendy): lekka piechota i rydwany w linii, ciężka piechota w falandze, łucznicy w łuku, kawaleria w klinie, słonie bojowe w kolumnie. Strzałki wskazują kierunek manewru każdej grupy ku celowi (★, najlepsze znane rozwiązanie).



Rys. 49. Ta sama heterogeniczna armia ABO w sześciu taktykach globalnych. Rozmieszczenie ról jest wierne implementacji: np. w okrążeniu kawaleria i łucznicy pozostają w większej odległości od celu, a słonie bojowe najbliżej; w szyku skośnym ciężka piechota tworzy silne skrzydło, a kawaleria jest cofnięta; w zwiadzie wszystkie grupy rozchodzą się na zewnątrz (eksploracja).

PARAMETRY I KONFIGURACJA

Domyślne parametry ABO

Tabela 89. Domyślne parametry Ancient Battlefield Optimizer (implementacja referencyjna)

Parametr	Wartość	Opis
Parametry optymalizatora		
`pop_size`	40	Domyślny rozmiar populacji (armii)
`epoch`	100	Domyślna liczba epok
`max_stagnant_iter`	20	Maks. iteracji bez poprawy przed dezaktywacją
Parametry Commander AI (Q-learning)		
α (learning_rate)	0,1	Współczynnik uczenia Q-learning
γ (discount_factor)	0,9	Współczynnik dyskonta
ϵ (exploration_rate)	$\max(0{,}03,\ 0{,}12(1-p))$	Ciągły zanik eksploracji: $12\% \rightarrow 3\%$
memory_size	1000	Rozmiar bufora doświadczeń (experience replay)
batch_size	32	Rozmiar próbki w experience replay
Parametry uczenia transferowego		
`transfer_mode`	full	Tryb łączenia Q-tabel (domyślnie: pełne nadpisanie; dostępne: weighted, adaptive, preserve)
θ_{conf}	0,3	Próg konfidencji (tryb preserve)
Parametry systemu honorów		
η (honor_decay)	0,95	Współczynnik zaniku honoru (wartość w klasie Army)
θ_H (hero_threshold)	6,0	Próg honoru: status bohatera
θ_R (runagate_threshold)	$-10{,}0$	Próg honoru: status dezertera
honor_gain (global improvement)	10,0	Bonus honoru za poprawę globalnego najlepszego
Parametry reconnaissance		
G (grid_resolution)	auto	$G = \min(20, \max(5, \lfloor 20 \cdot d^{-1/3} \rfloor))$
max_scouts	$\min(10, d)$	Maksymalna liczba zwiadowców
max_high_interest_cells	200	Maks. komórek o wysokim zainteresowaniu

ciąg dalszy na następnej stronie

Tabela 89 – ciąg dalszy z poprzedniej strony

Parametr	Wartość	Opis
q_table_readonly	True	Q-tabela wczytywana bez zapisu (tryb benchmarkowy)
Parametry strategii ruchu (domyślne, nadpisywane przez taktyki)		
step_scale	1,0	Mnożnik kroku (taktyki: 0,1–2,0)
levy_alpha	1,5	Indeks stabilności rozkładu Lévy'ego (1,0–2,0)
charge_distance_scale	1,0	Skala odległości szarży kawalerii
formation_pull	–	Nadpisanie siły elastic-band (taktyki: 0,3–0,8)
formation_discipline	0,7	Bazowa dyscyplina formacyjna jednostki
de_f_range	(0,5, 0,9)	Zakres współczynnika mutacji DE (piechota lekka)
de_cr	0,9	Prawdopodobieństwo krzyżowania DE
n_arrows	4	Liczba strzałów łuczników na iterację
anti_stag_threshold	30	Próg stagnacji do resetu antystagnacyjnego
stampede_prob	0,1	Bazowe prawdopodobieństwo szarży słoni
Parametry elastic-band pull (Alg. 18)		
Próg aktywacji τ	$r \cdot 0{,}15 \cdot (1{,}5 - 0{,}5p)$	Skaluje się z zasięgiem i postępem
Bazowa siła pull	$\delta \cdot 0{,}4$	δ = formation_discipline
Maks. siła pull	0,8	Ograniczenie górne
Redukcja dla bohatera	$\times 0{,}7$	Bohater ma luźniejszą formację

Uzasadnienie wartości parametrów

Rzetelność metodologiczna wymaga rozróżnienia, które wartości parametrów ABO mają oparcie w literaturze lub w empirycznym strojeniu, a które są *magic numbers* – wybranymi heurystycznie wartościami pierwszego rzędu, niepoddanymi formalnej ablacji. Tabela 90 klasyfikuje wszystkie parametry z Tabeli 89 według źródła wartości w trzech kategoriach:

- **[L]** – wartość przejęta z literatury (cytat źródła);
- **[E]** – wartość ustalona w pilotażowym strojeniu na 10 funkcjach treningowych (zbiór rozłączny ze zbiorem testowym, Sekcja 5.2.2);

- **[M]** – magic number (wybór heurystyczny bez formalnego strojenia; wymaga ablacji w przyszłych pracach).

Tabela 90. Klasyfikacja parametrów ABO według źródła wartości. L = literatura, E = strojenie empiryczne, M = magic number.

Parametr	Wartość	Klasa	Uzasadnienie / źródło
Optymalizator (budżet zgodny z protokołem CEC)			
`pop_size`	40	L	Standard CEC dla $D \leq 100$ [74]; potwierdzone w mealpy [108]
`epoch`	100	L	Budżet zgodny z protokołem porównawczym CEC [74] (faktyczny budżet w pracy: 4 000 FE = 40×100)
`max_stagnant_iter`	30	M	Próg restartu antystagnacyjnego jednostki defensywnej (`units.py`); 30% budżetu epok
`formation_pull`	0,1–0,8	T	Modyfikowany przez taktykę: niski (0,1) dla luźnej eksploracji, wysoki (0,8) dla zwartej formacji
Q-learning (zgodne z Sutton-Barto)			
α learning rate	0,1	L	Standardowa wartość dla tabular Q-learning [104, rozdz. 6,5]
γ discount	0,9	L	Standard dla epizodów o długości $\leq$ 100 [104, rozdz. 3,3]
ϵ exploration	$\max(0{,}03;\ 0{,}12(1-p))$	E	Strojenie pilotażowe nad 3 schematami zaniku (linear, exponential, logistic); liniowy 12% $\rightarrow$ 3% najlepszy
memory_size	1000	M	Heurystyka; brak wykorzystania (tablicowy Q-learning, experience replay nieaktywny)
batch_size	32	M	jw.; rezerwa pod przyszłe rozszerzenie do DQN
System honoru – magic numbers			
η honor decay	0,95	M	Heurystyka: $0{,}95^{20} \approx 0{,}36$ (połowiczny zanik w $\sim$ 14 iter.); brak ablacji
θ_H hero	6,0	M	Heurystyka; brak ablacji wrażliwości

ciąg dalszy na następnej stronie

Tabela 90 – ciąg dalszy

Parametr	Wartość	Klasa	Uzasadnienie / źródło
θ_R runagate	$-10{,}0$	M	Asymetria $\lvert\theta_R\rvert > \theta_H$ heurystyczna (kara surowsza niż nagroda)
honor_gain	10,0	M	Skaluje się z θ_H; brak ablacji
Reconnaissance			
G grid resolution	$\min(20, \max(5, \lfloor 20d^{-1/3} \rfloor))$	E	Zaprojektowane jako $O(d^{-1/3})$ aby ograniczyć liczbę komórek do ≤ 8000 niezależnie od d
max_scouts	$\min(10, d)$	M	Heurystyka: 10% populacji lub d, co mniejsze
max_high_interest_cells	200	M	Górne ograniczenie pamięci; brak ablacji
Operatory taktyczne (parametry strategii ruchu)			
`levy_alpha`	1,5	L	Mediana zakresu stabilności dla Lévy flight optimizers [124, 77]
`de_f_range`	(0,5; 0,9)	L	Standardowy zakres DE/rand/1 [102, 89]
`de_cr`	0,9	L	Wartość dla funkcji separowalnych [102]; ABO używa jej domyślnie
`n_arrows`	4	M	Heurystyka: 4 strzałów na iterację balansuje koszt vs. pokrycie; brak ablacji
`anti_stag_threshold`	30	M	Heurystyka; skaluje się jako 1,5×`max_stagnant_iter`
`stampede_prob`	0,1	M	Heurystyka: 10% iteracji ze stampede; brak ablacji
Elastic-band pull (Alg. 18)			
próg τ	$r \cdot 0{,}15(1{,}5-0{,}5p)$	M	Heurystyczna zależność liniowa od postępu; ablacja wrażliwości pozostaje do wykonania
bazowa siła pull	$\delta \cdot 0{,}4$	M	Mnożnik 0,4 dobrany heurystycznie
maks. siła pull	0,8	M	Limit dobrany heurystycznie
redukcja bohatera	$\times 0{,}7$	M	jw.

Bilans klasyfikacji. Z 26 udokumentowanych parametrów: **9** ma oparcie w literaturze (klasa L), **4** pochodzą ze strojenia empirycznego na zbiorze treningowym (klasa E), a **13** stanowi heurystycznie wybrane magic numbers. Większość magic numbers znajduje się w systemie honoru oraz w parametrach formacji – są to mechanizmy autorskie, dla których nie istnieje literatura referencyjna. **Ablacja wrażliwości** na każdy z tych 13 parametrów wymagałaby co najmniej $13 \times 5 \times 165 \times 30 = 321\,750$

uruchomień (5 wartości na parametr, 30 powtórzeń, 165 konfiguracji), co przekracza budżet obliczeniowy niniejszej pracy. Ablacja wrażliwości pozostaje pierwszorzędną pozycją na liście dalszych prac (Sekcja 7.4).

KOMPLETNY INDEKS TRUDNOŚCI FUNKCJI BENCHMARKOWYCH

Poniższe tabele przedstawiają kompletny indeks trudności (Hardness Index) dla wszystkich 280 par funkcja-wymiar, obliczony na podstawie analizy wydajności 29 algorytmów metaheurystycznych. Metodologia obliczania opisana jest w Sekcji 6.2.

Indeks trudności – wymiar 2D

Tabela 91. Indeks trudności funkcji w wymiarze 2D (posortowane malejąco)

Poz.	Funkcja	Wymiar	H [%]	Zestaw
1	ChenBird	2	83,9	–
2	Infinity	2	83,9	✓
3	Csendes	2	83,9	✓
4	ChungReynolds	2	83,6	✓
5	Cigar	2	83,6	✓
6	Zimmerman	2	83,3	–
7	Booth	2	83,2	–
8	CamelThreeHump	2	83,2	–
9	Katsuura	2	83,1	–
10	Matyas	2	83,1	–
11	EggCrate	2	83,1	–
12	FreudensteinRoth	2	83,1	–
13	Quartic	2	83,1	✓
14	Damavandi	2	83,0	–
15	Salomon	2	83,0	✓
16	YaoLiu04	2	83,0	✓
17	Deb01	2	82,9	✓
18	XinSheYang01	2	82,9	✓
19	Qing	2	82,9	✓
20	DixonPrice	2	82,9	✓
21	Parsopoulos	2	82,7	✓
22	CrossLegTable	2	82,7	–
23	Ackley01	2	82,6	✓
24	Exp2	2	82,2	–

Kontynuacja tabeli 91

Poz.	Funkcja	Wymiar	H [%]	Zestaw
25	Levy13	2	82,2	–
26	Levy03	2	81,9	✓
27	ElAttarVidyasagarDutta	2	77,9	–
28	ChenV	2	77,6	–
29	Griewank	2	76,3	✓
30	BartelsConn	2	71,5	–
31	Quintic	2	70,5	✓
32	Brent	2	69,9	–
33	NewFunction02	2	69,2	–
34	ZeroSum	2	68,7	✓
35	Mishra04	2	68,3	–
36	NewFunction01	2	67,7	–
37	Leon	2	66,2	–
38	Easom	2	66,0	–
39	Himmelblau	2	65,6	–
40	BiggsExp02	2	64,5	–
41	Beale	2	64,1	–
42	Cube	2	63,6	–
43	Zettl	2	63,5	–
44	Alpine01	2	63,2	✓
45	Brown	2	62,2	✓
46	Bohachevsky2	2	61,4	–
47	Bohachevsky1	2	61,2	–
48	Bohachevsky3	2	61,1	–
49	Zacharov	2	60,8	✓
50	CrownedCross	2	59,4	–
51	Mishra08	2	58,7	–
52	Decanomial	2	58,6	–
53	Keane	2	58,4	–
54	Mishra11	2	57,0	✓
55	Mishra03	2	57,0	–
56	MultiModal	2	56,7	✓
57	NeedleEye	2	55,6	✓
58	AMGM	2	54,7	✓
59	Mishra10	2	54,1	–
60	CosineMixture	2	53,2	✓
61	GoldsteinPrice	2	51,3	–
62	Bukin02	2	40,7	–
63	Bukin04	2	40,6	–
64	Deceptive	2	37,2	✓

Kontynuacja tabeli 91

Poz.	Funkcja	Wymiar	H [%]	Zestaw
65	DeckkersAarts	2	36,7	–
66	HolderTable	2	36,3	–
67	Mishra01	2	35,3	✓
68	Bukin06	2	34,8	–
69	Mishra02	2	34,3	✓
70	Adjiman	2	34,0	–
71	Zirilli	2	31,3	–
72	Branin01	2	29,8	–
73	Levy05	2	27,9	–
74	Branin02	2	24,8	–
75	DeflectedCorrugatedSpring	2	20,1	✓
76	EggHolder	2	17,2	✓
77	Langermann	2	17,0	–
78	JennrichSampson	2	15,3	–
79	Mishra05	2	15,0	–
80	VenterSobiezcczanskiSobieski	2	14,3	–
81	Hansen	2	13,9	–
82	Mishra06	2	13,7	–
83	CamelSixHump	2	13,2	–
84	Judge	2	11,3	–
85	Deb03	2	11,3	–
86	Michalewicz	2	10,5	–
87	Dolan	2	10,0	–
88	LennardJones	2	10,0	–
89	Mishra07	2	10,0	–
90	Rana	2	8,7	✓
91	Chichinadze	2	8,0	–
92	DropWave	2	7,8	✓
93	McCormick	2	5,8	–
94	Bird	2	5,7	–
95	Giunta	2	4,3	–
96	Ackley03	2	3,6	–
97	Alpine02	2	3,2	✓
98	Ackley02	2	2,2	–
99	Hosaki	2	1,3	–
100	Ursem01	2	1,2	–
101	TestTubeHolder	2	1,2	–
102	Quadratic	2	1,0	–
103	CrossInTray	2	0,9	–

104	Exponential	2	0,8	–

Indeks trudności – wymiar 10D

Tabela 92. Indeks trudności funkcji w wymiarze 10D

Poz.	Funkcja	Wymiar	H [%]	Zestaw
1	Katsuura	10	89,3	–
2	Zacharov	10	89,2	✓
3	Brown	10	88,7	✓
4	ChungReynolds	10	88,6	✓
5	Cigar	10	88,6	✓
6	Csendes	10	88,6	✓
7	Deb01	10	88,6	✓
8	DixonPrice	10	88,6	✓
9	Infinity	10	88,6	✓
10	Levy03	10	88,6	✓
11	Quartic	10	88,6	✓
12	Quintic	10	88,6	✓
13	XinSheYang01	10	88,6	✓
14	Parsopoulos	10	88,5	✓
15	Qing	10	88,5	✓
16	Mishra02	10	87,7	✓
17	Mishra01	10	87,6	✓
18	Alpine02	10	87,3	✓
19	Alpine01	10	85,3	✓
20	YaoLiu04	10	83,3	✓
21	DeflectedCorrugatedSpring	10	83,3	✓
22	ZeroSum	10	83,0	✓
23	Griewank	10	76,0	✓
24	Salomon	10	74,7	✓
25	Deb03	10	74,1	–
26	Ackley01	10	71,7	✓
27	Dolan	10	69,4	–
28	AMGM	10	64,0	✓
29	DropWave	10	63,5	✓
30	MultiModal	10	61,4	✓
31	Mishra11	10	60,2	✓
32	CosineMixture	10	58,7	✓
33	EggHolder	10	50,7	✓
34	Rana	10	48,4	✓
35	Deceptive	10	46,1	✓

Kontynuacja tabeli 92

Poz.	Funkcja	Wymiar	H [%]	Zestaw
36	NeedleEye	10	43,2	✓
37	LennardJones	10	24,6	–
38	Exponential	10	21,8	–
39	Mishra07	10	10,0	–

Indeks trudności – wymiar 30D

Tabela 93. Indeks trudności funkcji w wymiarze 30D

Poz.	Funkcja	Wymiar	H [%]	Zestaw
1	Brown	30	93,8	✓
2	Katsuura	30	93,5	–
3	Alpine02	30	93,5	✓
4	Zacharov	30	93,2	✓
5	XinSheYang01	30	93,1	✓
6	ChungReynolds	30	92,5	✓
7	Cigar	30	92,5	✓
8	Csendes	30	92,5	✓
9	Deb01	30	92,5	✓
10	DropWave	30	92,5	✓
11	Infinity	30	92,5	✓
12	Levy03	30	92,5	✓
13	Qing	30	92,5	✓
14	Quartic	30	92,5	✓
15	DixonPrice	30	92,3	✓
16	Parsopoulos	30	92,3	✓
17	Quintic	30	92,2	✓
18	Mishra01	30	92,2	✓
19	Mishra02	30	92,2	✓
20	ZeroSum	30	89,0	✓
21	Griewank	30	84,1	✓
22	DeflectedCorrugatedSpring	30	82,5	✓
23	Alpine01	30	79,0	✓
24	Dolan	30	74,3	–
25	AMGM	30	72,0	✓
26	YaoLiu04	30	71,3	✓
27	Salomon	30	70,1	✓
28	EggHolder	30	64,2	✓
29	MultiModal	30	62,7	✓
30	Rana	30	62,6	✓

Kontynuacja tabeli 93

Poz.	Funkcja	Wymiar	H [%]	Zestaw
31	Exponential	30	62,1	–
32	LennardJones	30	60,7	–
33	Mishra11	30	60,1	✓
34	Ackley01	30	57,3	✓
35	CosineMixture	30	55,5	✓
36	Deceptive	30	53,6	✓
37	NeedleEye	30	39,7	✓
38	Deb03	30	10,0	–
39	Mishra07	30	10,0	–

Indeks trudności – wymiar 50D

Tabela 94. Indeks trudności funkcji w wymiarze 50D

Poz.	Funkcja	Wymiar	H [%]	Zestaw
1	Alpine02	50	95,6	✓
2	Brown	50	95,6	✓
3	Katsuura	50	95,5	–
4	XinSheYang01	50	95,3	✓
5	Parsopoulos	50	94,5	✓
6	Zacharov	50	94,3	✓
7	Mishra01	50	94,3	✓
8	ChungReynolds	50	94,3	✓
9	Csendes	50	94,3	✓
10	Deb01	50	94,3	✓
11	DixonPrice	50	94,3	✓
12	Infinity	50	94,3	✓
13	Qing	50	94,3	✓
14	Quartic	50	94,3	✓
15	Quintic	50	94,3	✓
16	Mishra02	50	94,2	✓
17	DropWave	50	94,0	✓
18	Levy03	50	93,6	✓
19	ZeroSum	50	91,3	✓
20	Cigar	50	89,1	✓
21	Exponential	50	85,3	–
22	Griewank	50	84,9	✓
23	Alpine01	50	77,5	✓
24	DeflectedCorrugatedSpring	50	76,8	✓
25	AMGM	50	74,3	✓

Kontynuacja tabeli 94

Poz.	Funkcja	Wymiar	H [%]	Zestaw
26	Dolan	50	74,3	–
27	EggHolder	50	71,8	✓
28	Rana	50	69,5	✓
29	Salomon	50	68,9	✓
30	YaoLiu04	50	65,8	✓
31	LennardJones	50	63,1	–
32	MultiModal	50	62,1	✓
33	Mishra11	50	61,8	✓
34	Deceptive	50	58,1	✓
35	CosineMixture	50	57,7	✓
36	Ackley01	50	55,4	✓
37	NeedleEye	50	38,7	✓
38	Deb03	50	10,0	–
39	Mishra07	50	10,0	–

Indeks trudności – wymiar 100D

Tabela 95. Indeks trudności funkcji w wymiarze 100D

Poz.	Funkcja	Wymiar	H [%]	Zestaw
1	Katsuura	100	100,0	–
2	Alpine02	100	98,2	✓
3	XinSheYang01	100	97,9	✓
4	Mishra02	100	97,6	✓
5	Mishra01	100	97,6	✓
6	Brown	100	97,1	✓
7	ChungReynolds	100	96,7	✓
8	Csendes	100	96,7	✓
9	Deb01	100	96,7	✓
10	DropWave	100	96,7	✓
11	Infinity	100	96,7	✓
12	Quartic	100	96,7	✓
13	Quintic	100	96,7	✓
14	Zacharov	100	96,7	✓
15	Parsopoulos	100	96,2	✓
16	DixonPrice	100	95,6	✓
17	Qing	100	94,8	✓
18	Exponential	100	93,8	–
19	ZeroSum	100	90,2	✓
20	Levy03	100	89,4	✓

Kontynuacja tabeli 95

Poz.	Funkcja	Wymiar	H [%]	Zestaw
21	Cigar	100	86,0	✓
22	Griewank	100	84,5	✓
23	Rana	100	82,4	✓
24	EggHolder	100	81,7	✓
25	AMGM	100	78,3	✓
26	Alpine01	100	76,6	✓
27	Dolan	100	76,1	–
28	DeflectedCorrugatedSpring	100	74,7	✓
29	Salomon	100	67,5	✓
30	YaoLiu04	100	64,1	✓
31	Mishra11	100	63,3	✓
32	MultiModal	100	62,8	✓
33	Deceptive	100	61,1	✓
34	CosineMixture	100	59,5	✓
35	Ackley01	100	54,9	✓
36	NeedleEye	100	43,0	✓
37	Deb03	100	10,0	–
38	LennardJones	100	10,0	–
39	Mishra07	100	10,0	–

Podsumowanie indeksu trudności

Tabela 96. Statystyki indeksu trudności według wymiarowości

Wymiar	Liczba	Średnia	Std	Min	Max
2	104	48,8%	29,9%	0,7%	83,9%
10	39	73,0%	21,2%	10,0%	89,3%
30	39	76,4%	21,8%	10,0%	93,8%
50	39	78,1%	22,1%	10,0%	95,6%
100	39	78,7%	25,0%	10,0%	100,0%
Razem	**280**	**65,4%**	**29,2%**	**0,7%**	**100,0%**

Główne wnioski z analizy indeksu trudności:

1. **Wzrost trudności z wymiarowością**: Średnia trudność rośnie od 48,8% w 2D do 78,7% w 100D, potwierdzając zjawisko *curse of dimensionality*.
2. **Funkcje o stałej trudności 10%**: Funkcje Deb03, Mishra07 i LennardJones wykazują stałą niską trudność (10%) niezależnie od wymiarowości – są to funkcje o specyficznej charakterystyce (wiele równoważnych optimów lub degeneracja dla algorytmów metaheurystycznych).

3. **Najtrudniejsza funkcja**: Katsuura@100D osiąga maksymalną trudność 100%, co oznacza, że żaden z 30 testowanych algorytmów nie był w stanie efektywnie znaleźć optimum globalnego.

4. **Reprezentatywność benchmarku**: 33 funkcje benchmarkowe (oznaczone ✓) pokrywają 90% z TOP 20 najtrudniejszych funkcji w każdym wymiarze, co potwierdza wysoką jakość wybranego zestawu testowego.

293:84530979