AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

PRACA DYPLOMOWA MAGISTERSKA

Wykorzystanie technik mrówkowych i ewolucyjnych w rozwiązywaniu dyskretnych problemów optymalizacji

Ant-colony and evolutionary techniques in solving discrete optimization problems

Autor: *inż. Tomasz Pachana*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr hab. inż. Aleksander Byrski*

Kraków, 2016

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim", oświadczam, że niniejszą pracę dyplomową wykonałem osobiście, samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

……

Spis treści

1. Wprowadzenie

Opis problemu

Pojęcie problemów optymalizacyjnych odzwierciedla wiele wyzwań stawianych przed współczesną algorytmiką przez liczne dziedziny inżynierii i przemysłu. Związane z nim zagadnienie poszukiwania minimalizacji bądź maksymalizacji wartości jakiejś właściwości stanowi istotę ogromnej ilości zadań wynikających z rzeczywistych zapotrzebowań. Znacząca większość tego typu problemów należy do grupy problemów NP-trudnych, dla których znalezienie rozwiązania w odpowiednio krótkim czasie, bądź odpowiednio dobrej jakości może stanowić wyzwanie nawet przy dostępie do zaawansowanej infrastruktury obliczeniowej. Z tego powodu od lat prowadzone są badania nad rozmaitymi technikami, które pozwolą na ich sprawne rozwiązywanie. Jedną z podgrup tych technik stanowią algorytmy heurystyczne, pozwalające na poszukiwanie rozwiązań niekoniecznie najbardziej dokładnych, ale wystarczająco dobrych w obliczu stawianych przed nimi oczekiwań. Ich znaczną zaletą jest znacznie mniejszy koszt obliczeń w porównaniu z algorytmami dokładnymi. Biorąc powyższe pod uwagę, heurystyki zostały wybrane jako przedmiot badań opisanych w niniejszej pracy. W początkowych rozdziałach przedstawiono formalne definicje i szczegółową analizę tematyki związanej z problemami optymalizacji dyskretnej oraz problemu komiwojażera, jako reprezentatywnego przykładu tego typu zagadnienia. Opisane zostały także najbardziej popularne grupy metod stosowanych w rozwiązywaniu tego typu problemów ze szczególnym naciskiem na metody heurystyczne, jako najczęściej znajdujących zastosowanie w tej dziedzinie. W dalszej części przybliżony został wyczerpujący przegląd niektórych technik heurystycznych, stanowiących przedmiot licznych badań, takich jak algorytmy ewolucyjne, mrówkowe, a także hybrydowe. Praca demonstruje również problem wynikający ze stałego rozwoju stawianych w tej dziedzinie zadań, a także dogłębnie opisuje proponowane rozwiązanie wraz ze szczegółami implementacyjnymi wypracowanego prototypu. Jednym z kluczowych elementów są opisywane w dalszych rozdziałach badania i eksperymenty, demonstrujące wyniki uzyskiwane przez zastosowanie proponowanej koncepcji. Przeprowadzone zostało także gruntowne porównanie jakości działania badanej techniki

w stosunku do metod tradycyjnie stosowanych do rozwiązywania podobnych problemów. Zakończenie obejmuje analizę uzyskanych wyników oraz przedstawia możliwości prowadzenia dalszych badań nad omawianym tematem.

Motywacja i cel pracy

Współczesna algorytmika zna oczywiście wiele rozwiązań, które w sposób efektywny są w stanie poszukiwać rozwiązań skomplikowanych problemów optymalizacyjnych. Wynika to z faktu, że tego typu zadania stawiane są przed naukowcami już od kilkudziesięciu lat. Im bardziej złożony jest jednak dany problem, tym więcej trudności jest on w stanie przysporzyć. Z tego powodu, większość znanych technik w tej dziedzinie, to przede wszystkim systemy dające rozwiązania przybliżone, czyli wystarczająco dobre dla ich odbiorcy, jednak nie będące najbardziej optymalnymi spośród wszystkich możliwych. Posiadają one pewne ograniczenia, związane nie tylko z jakością otrzymywanych wyników, ale także z zastosowaniem dla konkretnego rodzaju problemów oraz czasem działania. Wraz z ciągłym rozwojem przemysłu, który stanowi głównego odbiorcę rozwiązań problemów optymalizacyjnych, rosną również potrzeby z tym związane. Zwiększają się nie tylko rozmiary znanych już problemów, ale powstaje także ich wiele nowych, nieznanych do tej pory, dotyczących bardzo specyficznych dziedzin. Fakt ten prowadzi do potrzeby prowadzenia ciągłych prac nad nowymi algorytmami, które będą w stanie sprostać rosnącym potrzebom i dostarczać nie tylko rozwiązania bardziej satysfakcjonujące dla istniejących problemów, ale także potrafiące poradzić sobie z co raz to bardziej złożonymi, nowymi zadaniami. Analizowane w pracy techniki metaheurystyczne znajdują obecnie szerokie zastosowanie w tej dziedzinie, a więc należy zastanowić się, co można zrobić, aby usprawnić efekty ich działania i poprawić otrzymywane z nich wyniki.

Ostatnie lata przyniosły także bardzo dynamiczny rozwój architektur komputerowych. Co raz bardziej powszechny staje się dostęp do systemów o bardzo wysokiej wydajności, wysoce zrównoleglonych, a często nawet rozproszonych. Łatwo dostrzegalny jest także trend migracji systemów informatycznych do chmur obliczeniowych. Jest to kolejny czynnik, który otwiera wiele nowych możliwości, a jednocześnie wymusza konieczność prowadzenia dalszych badań w obszarze rozwiązywania problemów optymalizacyjnych z wykorzystaniem tychże architektur.

Niniejsza praca stanowi propozycję rozwiązania, które bazując na dobrze znanych i sprawdzonych ideach w dziedzinie optymalizacji dyskretnej, pozwoli na sprostanie rosnącym oczekiwaniom odbiorców tego typu systemów. Jako sformułowanie problemu, można więc zdefiniować zadanie poszukiwania możliwości wykorzystania algorytmów ewolucyjnych i mrówkowych do rozwiązywania dyskretnych problemów optymalizacyjnych.

2. Wybrane metaheurystyki dla rozwiązywania problemów dyskretnych

2.1. Problemy optymalizacji dyskretnej

Problemy optymalizacyjne są jednymi z najbardziej popularnych wśród współcześnie rozważanych w dziedzinie algorytmiki. Warto po krótce przytoczyć czym charakteryzuje się ta grupa zagadnień.

2.1.1. Definicja

Dokładną definicję problemu optymalizacji stawiają autorzy [40]. Określają go oni jako problem poszukiwania najlepszego z możliwych rozwiązań z pośród zbioru wszystkich możliwych dla danego zagadnienia. Jako wyznacznik jakości rozwiązania przyjmowana jest minimalna lub maksymalna wartość funkcji powiązanej z danym zagadnieniem (*funkcji kryterialnej*). Funkcja ta może reprezentować koszt, zysk, ilość lub inną podobną, dającą się łatwo zmierzyć wartość.

Jedną z wersji problemu optymalizacji jest optymalizacja dyskretna. Jej formalną definicję proponują autorzy [34]. W najbardziej ogólnej formie określają go jako *(1):*

$$\min \text{ (lub } \max) \ \alpha(T), \quad T \in F \tag{1}$$

Przyjąć tu można T – uporządkowanie, F - możliwe uporządkowania, a $\alpha(T)$ - miara wartości uporządkowań z F. Wspomniane rozważania na temat uporządkowań są oczywiście również podstawą kombinatoryki. Można więc stwierdzić, że ponieważ optymalizacja dyskretna opiera się na wyborze pomiędzy wzajemnie wykluczającymi się alternatywami, może być ona postrzegana jako pewna gałąź kombinatoryki, dotykająca problemów ekstremizacji [34].

Problemami optymalizacji dyskretnej lub dyskretno-ciągłej nazywamy przede wszystkim takie, w których zmienne decyzyjne (bądź ich część) przyjmują wartości dyskretne, całkowitoliczbowe lub binarne [10]. Należy zauważyć, że większość problemów planowania i sterowania (jak np. te rozważane w niniejszej pracy) należy do takiej właśnie klasy problemów. Możliwe jest sformułowanie większości z nich w następującej formie [34] *(2)*:

$$\min \text{ (lub } \max) \sum_{j=1}^{n} c_j x_j \qquad (2)$$

Gdzie x_j to zmienne decyzyjne o niezerowych wartościach, a c_j stanowią o ich wagach. Problemy tej klasy należą do wyjątkowo trudnych obliczeniowo, głównie z uwagi na mnogość występowania ekstremów lokalnych i często brak własności wypukłości, ciągłości i różniczkowalności funkcji kryterialnej. Istnienie wielu ekstremów lokalnych, kłopotliwe już dla przypadku optymalizacji ciągłej, nabiera dla problemów dyskretnych szczególnego znaczenia.

2.1.2. Problem komiwojażera jako przykład zadania optymalizacji dyskretnej

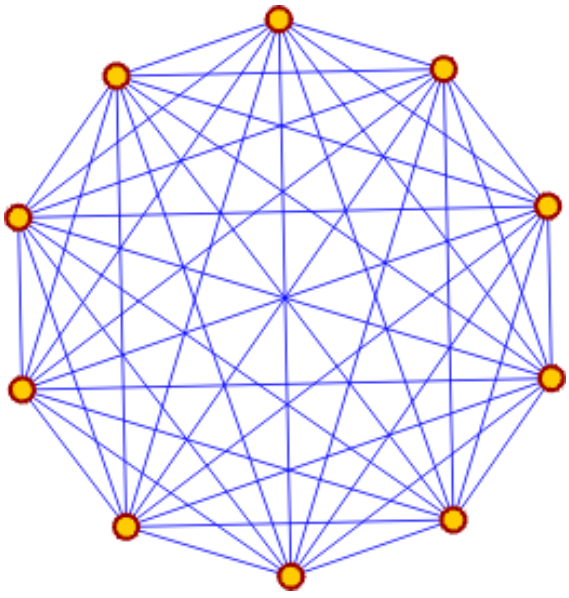
Wybór konkretnego problemu w celu przeprowadzenia eksperymentów ma znaczący wpływ na jego reprezentację w implementacji prototypu. Jako przykładowy problem do przeprowadzenia eksperymentów wybrany został problem komiwojażera. Wybór ten wynika z jego uniwersalności jako reprezentanta dyskretnych problemów optymalizacyjnych. Warto podkreślić, że proponowane rozwiązanie może dotyczyć dowolnego dyskretnego problemu optymalizacyjnego, a problem komiwojażera został wybrany dla przykładowej implementacji ze względu na szerokie możliwości uogólnienia do niego dowolnego przykładu takiego problemu.

Jest to jeden z koronnych problemów rozważanych w dziedzinie współczesnych, trudnych w rozwiązaniu problemów obliczeniowych. W najprostszej formie można zobrazować go jako problem osoby, która podróżuje pomiędzy miastami w celu np. sprzedaży towarów lub zawierania innych transakcji. Jednym z podstawowych założeń jest wyruszenie z określonego miasta rodzinnego i odwiedzenie wszystkich miast po drodze dokładnie jeden raz, kończąc podróż ponownie w mieście, z którego się rozpoczęła. Dość

oczywistym wydaje się więc być tutaj wymóg pokonania trasy jak najmniejszym możliwym kosztem, podczas gdy pomiędzy dowolną parą miast, pomiędzy którymi należy się przemieścić, może istnieć wiele możliwych połączeń, dłuższych lub krótszych. W literaturze problem ten nazywany jest problemem wędrującego komiwojażera (ang. TSP - Travelling Salesman Problem).

W reprezentacji grafowej problem ten można łatwo przedstawić jako wierzchołki, reprezentujące miasta, a trasy pomiędzy nimi jako krawędzie. Każda krawędź ma odpowiednią wagę, reprezentującą odległość, jaką należy pokonać w celu odbycia podróży pomiędzy miastami (wierzchołkami), które są przez nią połączone. Jak zostało, to już wcześniej wspomniane, trasę komiwojażera możemy przedstawić jako cykl, przechodzący przez każdy wierzchołek grafu dokładnie jeden raz - cykl Hamiltona. Znalezienie odpowiedniego cyklu jest zadaniem bardzo złożonym i często podejmowanym przez badaczy. Praca [29] zwraca dużą uwagę na tą złożoność, a także przedstawia kilka znanych algorytmów, które reprezentują różne podejścia do jego rozwiązywania. Ciekawą dyskusję na ten temat podejmuje również autor [8], próbując przedstawić nieznaną wcześniej metodę jego rozwiązywania.

Jako dobry przykład jego dużej złożoności można przedstawić tutaj graf zupełny (ang. *complete graph* - taki, w którym każdy wierzchołek jest połączony z każdym) o 10 wierzchołkach – *Rysunek 2.1* (co nie wydaje się dużą liczbą z perspektywy rzeczywistych problemów, modelowanych przez to sformułowanie).



Rysunek 2.1: Przykład grafu zupełnego o 10 wierzchołkach [47]

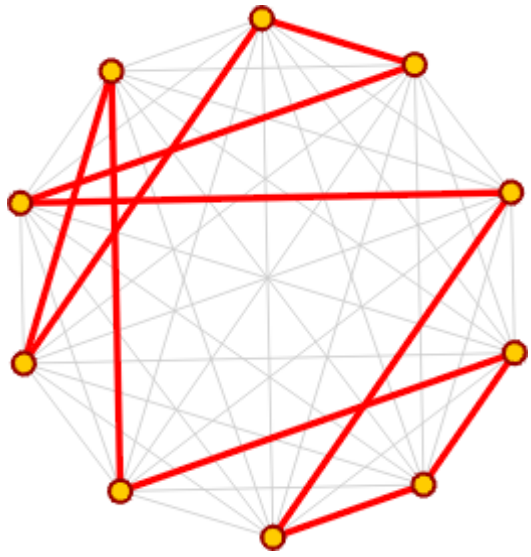
W grafie takim pierwszą krawędź cyklu znaleźć można na 9 różnych sposobów, z uwagi na fakt, że każdy wierzchołek grafu jest połączony krawędzią ze wszystkimi pozostałymi. Do wyboru kolejnej pozostaje nam 8 możliwości, itd. W ostateczności możliwą liczbę cykli Hamiltona dla tego przykładu, określić możemy jako *(3):*

$$L_H = 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 9! = 362880 \qquad (3)$$

Co dla *n*-wierzchołkowego grafu można uogólnić jako *(4):*

$$L_H = (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1 = (n-1)! \qquad (4)$$

Autor artykułu [47] zauważa, że dla tego przykładu, taki sposób postępowania prowadzi do bardzo niekorzystnej złożoności obliczeniowej $O(n!)$. Przykładowy cykl został zaznaczony na *Rysunku 2.2*.



Rysunek 2.2: Przykładowy cykl Hamiltona w grafie o 10 wierzchołkach [47]

Oczywiście przedstawiony przykład jest wariantem pesymistycznych pod względem odniesienia do rzeczywistych warunków, gdzie połączenia pomiędzy miastami nie są budowane w sposób zupełny. Pozwala on jednak dobrze zobrazować jego istotę. Przy współczesnym rozwoju i dostępności rozbudowanych zasobów obliczeniowych, rozwią-

zywanie problemów tego typu, dotyczących 10-15 miast z pomocą najprostszych, rekurencyjnych algorytmów, które analizują wszystkie możliwości i starają się znaleźć cykl o najniższej sumie wag nie stanowi problemu. Wyzwaniem mogą się okazać natomiast problemy o większym poziomie skomplikowania (dotyczące rzędu kilkudziesięciu / kilkuset miast), które coraz częściej dotykają wielu rzeczywistych branż, jak handel, logistyka, czy transport, a dla których znalezienie dokładnego rozwiązania w rozsądnym czasie okazuje się niemożliwe. Warto zauważyć jednak, że przy odpowiednio dużej skali problemu, znalezienie rozwiązania dokładnego nie jest koniecznością, a stosownie dobrane przybliżenie (które możliwe jest do znalezienia w akceptowalnym czasie z zastosowaniem algorytmów heurystycznych), okazuje się być w zupełności wystarczające.

2.1.3. Przegląd algorytmów optymalizacyjnych

Metody dokładne

Podstawową grupą metod, jakie możliwe są do zastosowania przy szukaniu rozwiązania dla dyskretnych problemów optymalizacyjnych są algorytmy dokładne. Swoje działanie opierają one na poszukiwaniu rozwiązań globalnie optymalnych. Stanowią one grupę algorytmów najbardziej kosztownych obliczeniowo w zastosowaniach tego typu, a pamiętać należy, że większość dyskretnych problemów optymalizacyjnych należy klasyfikować jako silnie NP-trudne. Koszt czasowy i pamięciowy przy ich wykorzystaniu jest bardzo wysoki, a rozmiary problemów, dla których czas poszukiwania rozwiązania jest akceptowalny, są wciąż zbyt małe. Metody te są na przykład wykorzystywane do poszukiwania minimalnego czasu cyklu dla powtarzalnego procesu produkcyjnego z niewielkim zakresem wyrobów. Dla małej liczby zadań można wyznaczyć rozwiązanie, którego nawet mały zysk otrzymany w jednym cyklu, zwielokrotni się o liczbę cykli w całym procesie. Ten rodzaj algorytmów często znajduje zastosowanie także w wyznaczaniu rozwiązań referencyjnych, do których następnie porównuje się wyniki algorytmów przybliżonych, oceniając ich jakość.

Wśród najbardziej znanych przykładów algorytmów z tej grupy, wymienić można następujące pozycje:

- **przeszukiwanie wyczerpujące** - w tej podgrupie znajdują się algorytmy prowadzące przeszukiwanie przestrzeni rozwiązań w sposób najprostszy. Zwykle odbywa

się to poprzez rekurencyjne generowanie kolejnych możliwych rozwiązań, obliczanie ich jakości i zapamiętywanie najlepszego. Grupa ta obejmuje algorytmy najbardziej kosztowne obliczeniowo, jak np. *brute force*.

- **podział i ograniczenia** - podzbiór algorytmów dokładnych, który stara się eliminować konieczność nadmiernego sprawdzania wszystkich możliwych rozwiązań. Na początku wykonania programu, budowane jest drzewo rozwiązań, po czym dla każdego poddrzewa uruchamiane jest sprawdzanie jakości rozwiązania każdego z węzłów. W każdym momencie wykonania, zapamiętywany jest znany do danego momentu najlepszy wynik, a jeśli w trakcie działania okazuje się, że dane poddrzewo nie będzie w stanie wygenerować wyniku lepszego niż znany do danego momentu, jego dalsze sprawdzanie nie jest przeprowadzane.
- **programowanie dynamiczne** - opiera się na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów. W odróżnieniu od techniki dziel i zwyciężaj podproblemy w programowaniu dynamicznym nie są rozłączne, ale musi je cechować własność optymalnej podstruktury. Zazwyczaj obejmuje dwa główne parametry, definiujących podproblemy, pierwszym z których jest liczba elementów znajdujących się w rozpatrywanym problemie, natomiast drugim pewna wartość liczbowa, zmieniająca się w zakresie od 0 do największej stałej występującej w problemie. Możliwa jest także większa ilość parametrów wejściowych, a także bardziej skomplikowany ich dobór.
- **programowanie matematyczne -** grupa algorytmów, w której nie ma jednej idealnej metody rozwiązywania problemów optymalizacyjnych. W zależności od konkretnego problemu, stosuje się różne odmiany programowania matematycznego, jak np. programowanie liniowe, całkowitoliczbowe, zero-jedynkowe, celowe, kwadratowe i inne. Jedną z jego odmian jest także opisane wcześniej programowanie dynamiczne. Znajduje szerokie zastosowanie w teorii decyzji, jak np. przy optymalizacji kosztów produkcji.
- **efektywne algorytmy dedykowane -** metody opracowywane do rozwiązywania konkretnych problemów obliczeniowych, silnie optymalizowane do działania z zadaniami, dla których są tworzone i architektur, na których będą prowadzone obliczenia.

Metody aproksymacyjne

Metody aproksymacyjne obejmują grupę niektórych przedstawicieli algorytmów przybliżonych, stosowanych do rozwiązywania problemów optymalizacyjnych. Istotną cechą tej grupy algorytmów jest możliwość łatwego określenia jakości zwracanego przez nie rozwiązania w stosunku do rozwiązania dokładnego. Cecha ta wynika z faktu, że koszt rozwiązania zwróconego przez algorytm aproksymacyjny jest nie większy (w przypadku problemu minimalizacyjnego) albo nie mniejszy (w przypadku problemu maksymalizacyjnego) od rozwiązania optymalnego pomnożonego przez pewną stałą. W celu wyznaczenia tej informacji, a co za tym idzie, oszacowania jakości danego algorytmu aproksymacyjnego, stosuje się jedną z kilku znanych metod. Warto po krótce przyjrzeć się każdej z nich.

Ograniczenie względne dla algorytmu aproksymacyjnego przy rozwiązywaniu danego problemu, możliwe jest do określenia jako $\rho(n)$, jeśli dla dowolnych danych wejściowych rozmiaru n koszt C otrzymany jako rozwiązanie konstruowane przez ów algorytm, szacuje się z dokładnością do czynnika $\rho(n)$, przez koszt C^* rozwiązania optymalnego *(5):*

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \le \rho(n) \qquad (5)$$

- Dla problemu maksymalizacji $0 < C \le C^*$, a współczynnik C^* / C określa, ile razy koszt rozwiązania optymalnego jest większy od kosztu rozwiązania przybliżonego.
- Dla problemu minimalizacji $0 < C^* \le C$, a współczynnik C / C^* określa, ile razy koszt rozwiązania optymalnego jest większy od kosztu rozwiązania przybliżonego.

Ograniczenie względne algorytmu aproksymacyjnego nigdy nie jest mniejsze niż 1, ponieważ nierówność $C / C^* < 1$ implikuje $C^* / C > 1$. Ograniczeniem względnym algorytmu optymalnego jest 1, a algorytm o dużym ograniczeniu względnym może dać rozwiązanie znacznie gorsze niż optymalne.

Błąd względny dla dowolnych danych wejściowych może zostać zdefiniowany jako *(6)*:

$$\frac{|C - C^*|}{C^*} \tag{6}$$

Błąd względny jest zawsze nieujemny. Dla algorytmu aproksymacyjnego $\varepsilon(n)$ jest ograniczeniem błędu względnego, jeśli *(7)*:

$$\frac{|C - C^*|}{C^*} \leq \varepsilon(n) \tag{7}$$

Dla problemu minimalizacyjnego prawdziwa jest równość *(8):*

$$\varepsilon(n) = \frac{\rho(n) - 1}{\rho(n)} \tag{8}$$

Dla problemu maksymalizacyjnego prawdziwa jest równość *(9):*

$$\varepsilon(n) \leq \rho(n) - 1 \tag{9}$$

Dla wielu problemów skonstruowano algorytmy aproksymacyjne o stałym, niezależnym od n ograniczeniu względnym. Dla innych problemów nie było to jednak możliwe. Ograniczenie względne jest wówczas funkcją rosnącą wraz z rozmiarem danych wejściowych n. Przykładem takiego problemu jest przedstawiony wcześniej TSP.

Schemat aproksymacji. Dla niektórych problemów NP-zupełnych istnieją algorytmy aproksymacyjne, za pomocą których można uzyskiwać coraz mniejsze ograniczenie względne, kosztem dużego czasu obliczeń. Schemat aproksymacji dla problemu optymalizacyjnego, to algorytm aproksymacyjny, który pobiera na wejściu nie tylko dane opisujące problem, ale także wartość $\varepsilon > 0$ tak, że dla każdego ustalonego ε, schemat ten jest algorytmem aproksymacji o ograniczeniu błędu względnego równym ε.

Metody heurystyczne

Powołując się na opinię Judea Perl z [36], metody heurystyczne można zdefiniować jako zbiór kryteriów, metod oraz zasad, znajdujących zastosowanie w podejmowaniu decyzji o tym, która z alternatywnych dróg rozwiązań będzie najlepsza w celu osiągnięcia jakiegoś wyznaczonego celu. Możliwe jest stwierdzenie, że przedstawiają zbiór kompromisów pomiędzy utrzymaniem jak największej prostoty takich metod, a jednocześnie prawidłowym określaniu przez nie właściwych i niewłaściwych wyborów. Heurystyka może być specyficzną regułą, wypracowaną na drodze eksperymentów i znajdującą zastosowanie jako wskazówka w podejmowaniu określonych decyzji. Spośród przykładów takiej reguły, można przytoczyć metodę stosowaną przez plantatorów melonów Kantalupa do określania czy owoc jest już dojrzały. Opiera się ona na wykonaniu delikatnego nacisku na miejsce, gdzie owoc był połączony z rośliną, a następnie sprawdzeniu zapachu w tym punkcie. Jeśli zapach jest taki, jak w środku owocu, najprawdopodobniej jest on dojrzały. Reguła ta nie daje gwarancji wyboru jedynie dojrzałych owoców, jest jednak skuteczna w większości przypadków. Jako kolejny, choć mniej oczywisty przykład zastosowania metody heurystycznej, można wyobrazić sobie mistrza szachowego, który stoi przed wyborem jednego z kilku możliwych ruchów w trakcie trwania rozgrywki. Jako osoba doświadczona w grze, może on łatwo zdecydować o danym ruchu, jako lepszym od innych, ponieważ układ planszy po nim wydaje mu się bardziej korzystny od tych, które powstaną w wyniku z pozostałych rozważanych ruchów. Takie kryterium jest dla mistrza znacznie prostsze do zastosowania niż np. rygorystyczne rozważanie, który z ruchów szybciej doprowadzi do zakończenia rozgrywki przez mat. Fakt, że mistrzowie szachowi nie zawsze wygrywają, wskazuje na zależność, że stosowana przez nich heurystyka nie zawsze gwarantuje wybór najlepszego z dostępnych ruchów. Co więcej, zapytani o szczegółowy opis stosowanej heurystyki, są w stanie udzielić jedynie częściowych i elementarnych odpowiedzi na temat tego, co według nich może być najbardziej efektywne w dowolnej sytuacji. Jednoznacznie wskazuje to na fakt, że dobre heurystyki dostarczają łatwej metody określania, które spośród wielu możliwości powinny być preferowane, oraz że nie dają one gwarancji dokonania najlepszego wyboru za każdym razem, jednak robią to dostatecznie często.

Najbardziej rozbudowane problemy wymagają rozważenia niezliczonej ilości możliwości w celu znalezienia dokładnego rozwiązania. Czas potrzebny na znalezienie go może więc w wielu przypadkach sięgać nieskończoności. Metody heurystyczne odgrywają

istotną rolę w znacznym ograniczeniu ilości potrzebnych do wykonania rozważań, a co za tym idzie, czasu potrzebnego na znalezienie rozwiązania. W kontekście algorytmów optymalizacyjnych, należy zauważyć, metody heurystyczne obejmują najbardziej popularne algorytmy stosowane obecnie w szukanie rozwiązań dla problemów NP-trudnych.

Najbardziej charakterystyczną cechą heurystyk, jest fakt, że są to algorytmy przybliżone, a więc nie dają gwarancji dostarczenia rozwiązania najbardziej dokładnego, jak zostało to już wcześniej opisane w przedstawionych przykładach. Pozwalają jednak na szukanie rozwiązań optymalnych, wystarczająco dobrych, a jednocześnie są w stanie dostarczyć wyniki w zadowalającym czasie trwania obliczeń. W wielu sytuacjach może się jednak okazać, że znalezione rozwiązanie nie jest nawet prawidłowe. Stosuje się je głównie w sytuacjach, gdy algorytm dokładny nie jest znany lub jego zastosowanie jest zbyt kosztowne. Często znajdują też zastosowanie przy znajdowaniu rozwiązania przybliżonego, które później wykorzystywane jest przez algorytm dokładny. W takiej sytuacji rozwiązanie przybliżone może pozwolić na znalezienie rozwiązania dokładnego, jednocześnie bez poświęcania zbyt wielu kosztów - np. czasowych. Grupa tych algorytmów jest obecnie przedmiotem badań nad rozwiązywaniem problemów z wielu dziedzin nauki i przemysłu.

Metaheurystyki

Szczególnym przypadkiem algorytmów heurystycznych są metaheurystyki. Autor [31] definiuje grupę tych algorytmów, jako najbardziej ogólną odmianę optymalizacji stochastycznej, czyli klasy technik wykorzystujących element losowości na potrzeby wyszukiwania jak najbardziej dokładnych rozwiązań trudnych problemów. Określa on obszar zastosowań metod metaheurystycznych jako problemy, na temat pracy z którymi bardzo mało wiadomo w momencie przystępowania do poszukiwania rozwiązania. W szczególności są to problemy słabo zbadane - nie wiadomo jak powinno wyglądać ich rozwiązanie optymalne, nie jest znany jednoznaczny sposób poszukiwania takiego rozwiązania, a przeszukiwanie ekstensywne jest wykluczone ze względu na zbyt dużą przestrzeń możliwych rozwiązań. Kolejną cechą takiego problemu powinna być możliwość łatwego określenia jakości danego rozwiązania, a więc stwierdzenia czy jest ono wystarczająco dobre. Z tego powodu, autor określa tą klasę problemów jako “wiem o nim, kiedy go widzę”. Jako przykład takiego problemu można przytoczyć poszukiwanie optymalnej sekwencji zachowań dla robota grającego w piłkę nożną. Konieczne jest tutaj założenie,

że dostępny jest np. symulator zachowań takiego robota, który pozwala przetestować dowolną sekwencję zachowań robota i wyznaczyć jakość takiej sekwencji. Wiadome jest również jak takie sekwencje zachowań są zbudowane w ogólności. Nie jest za to znana najbardziej optymalna sekwencja, a także metoda na znalezienie takiego zestawu zachowań. Najprostszym możliwym wyjściem będzie tutaj losowe próbkowanie różnych sekwencji tak długo, jak długo pozwala na to ograniczenie czasowe oraz zwrócenie na koniec najlepszego znalezionego. Możliwe jest bardziej optymalne poszukiwanie rozwiązania w tej sytuacji. W przytaczanej pozycji metoda ta opisywana jest jako technika wspinaczki-górskiej. Polega na wybraniu początkowej, losowej sekwencji zachowań, a następnie dokonywaniu w niej losowych zmian pojedynczych zachowań, które wchodzą w skład opisywanej sekwencji i sprawdzaniu jego jakości w kolejnych krokach. Jeśli nowe rozwiązanie jest lepsze niż poprzednie, powinno zostać wykorzystane w kolejnym kroku, a w przeciwnym razie - porzucone. W kolejnych krokach operacje te są powtarzane z wykorzystaniem sekwencji, wybranych w krokach je poprzedzających. Taki sposób postępowania jest kontynuowany tak długo, na jak długo pozwala limit czasu.

Technika wspinaczki-górskiej jest przykładem prostego algorytmu metaheurystycznego. Wykorzystuje ona założenie właściwe dla większości metod heurystycznych o przestrzeni możliwych rozwiązań, mówiące o tym, że podobne rozwiązania zwykle mają do siebie zbliżoną wartość funkcji jakości. Bazując na tym założeniu, można przyjąć, że małe zmiany w rozważanej sekwencji (będącej rozwiązaniem problemu) będą skutkować w małych, pozytywnych w skutkach zmianach w jakości rozwiązania, stopniowo zbliżając się do optymalnego rozwiązania wraz z kolejnymi powtórzeniami. Założenie to jest właściwe dla znakomitej większości algorytmów metaheurystycznych. Większość z nich opiera bowiem swoje działanie na połączeniu techniki wspinaczki-górskiej wraz z przeszukiwaniem losowym.

Zadania najczęściej rozwiązywane przez algorytmy metaheurystyczne stanowią podgrupę problemów nieodwracalnych. Problemy tego rodzaju można opisać jako takie, dla których mając zdefiniowaną funkcję kryterialną f (która przyjmuje jako argument testowane rozwiązanie, a zwraca wartość, określającą jakość tego rozwiązania), nie jest możliwe określenie funkcji odwrotnej $f^{'}$ (która przyjmowałaby zadaną wartość jakości rozwiązania, a zwracała jednoznaczne rozwiązanie, odpowiadające tej wartości). Dla przedstawionego wcześniej przykładu, symulator robota wraz z procedurą testową stanowią funkcję f. Korzystnym byłoby jednak poznanie funkcji odwrotnej $f^{'}$, dzięki której

możliwe stałoby się poznanie zestawu zachowań robota dla określonej optymalnej jakości rozwiązania. Metaheurystyki zaprojektowane są w taki sposób, aby możliwe było pominięcie tego problemu. Wiele klasycznych technik optymalizacyjnych, jak np. metoda gradientu prostego, wykorzystuje liczne założenia na temat funkcji f, jak np. założenie o znanej wartości pierwszej pochodnej tej funkcji. Ideą metod metaheurystycznych jest wykorzystanie znacznie mniejszej liczby tego typu założeń, a jeśli to możliwe, nie wykorzystywanie ich w ogóle. Pozwala to stwierdzić, że są one najbardziej ogólnymi technikami, jakie można wykorzystać, a często okazują się jedynymi możliwymi do zastosowania. Z tego powodu, łatwe zauważyć, że znajdują one zastosowanie w rozwiązywaniu coraz większej liczby istotnych problemów, która stale rośnie.

Algorytmy metaheurystyczne radzą sobie z większością trudności, jakie napotykane są przy stosowaniu innych metod optymalizacyjnych. Mają wyjątkową zdolność łatwej adaptacji i mogą być stosowane przy rozwiązywaniu złożonych nieliniowych i wielowymiarowych problemów inżynieryjnych. Wśród kilku problemów, występujących przy stosowaniu klasycznych algorytmów, a z którymi radzą sobie metody metaheurystyczne, można wymienić następujące [37]:

- Algorytm, który może być efektywny przy rozwiązywaniu danego problemu optymalizacji, może nie być efektywny przy rozwiązywaniu innego problemu.
- Algorytmy nie mogą zostać w sposób efektywny wykorzystane w architekturach równoległych.
- Zbieżność algorytmu do rozwiązania optymalnego zależy od wyboru rozwiązania początkowego.
- Algorytmy nie są efektywne w zastosowaniu do problemów z dyskretną przestrzenią poszukiwań.
- Większość algorytmów ma tendencję do utknięcia w rozwiązaniu suboptymalnym.

Pośród wielu cech odróżniających metody metaheurystyczne od innych algorytmów, wymienić należy [23]:

- Korzystają tylko z funkcji celu, nie zaś z jej pochodnych lub innych pomocniczych informacji.
- Prowadzą poszukiwania, wychodząc nie z pojedynczego punktu, lecz z pewnej ich liczby.
- Nie przetwarzają one bezpośrednio parametrów zadania, lecz ich zakodowaną postać.
- Stosują probabilistyczne a nie deterministyczne reguły wyboru.

2.2. Przykłady algorytmów metaheurystycznych

Jak zostało to wcześniej przedstawione, algorytmy metaheurystyczne są metodami, które są obecnie najbardziej ekstensywnie rozwijane spośród innych stosowanych w znajdowaniu rozwiązań dla problemów optymalizacyjnych. Najbardziej popularne algorytmy tego typu obejmują: symulowane wyżarzanie, sztuczne sieci neuronowe, algorytmy immunologiczne, algorytmy ewolucyjne, algorytmy mrówkowe i wiele innych. Przedmiotem badań niniejszej pracy są algorytmy ewolucyjne, mrówkowe i hybrydowe. Ich najważniejsze właściwości zostaną przedstawione w tym rozdziale.

2.2.1. Algorytmy ewolucyjne

Algorytmy ewolucyjne należy postrzegać jako przybliżone algorytmy optymalizacyjne, które inspirowane są przez biologiczny proces ewolucji, a dokładnie rzecz biorąc mechanizmy selekcji, reprodukcji i mutacji. Podstawą tej inspiracji jest obserwacja, że biologiczny proces ewolucji opiera się na presji środowiska, wywołującej naturalną selekcję. W jej wyniku tylko najlepiej przystosowane osobniki mają szanse na przetrwanie i zapoczątkowanie nowych coraz to lepszych populacji. Sprowadzając tą biologiczną terminologię do świata algorytmiki, można powiedzieć, że problem, który mamy rozwiązać, gra rolę środowiska, w którym żyje populacja osobników. Każdy osobnik reprezentuje potencjalne (możliwe) rozwiązanie problemu [39]. Nacisk kładziony jest przede wszystkim na zbliżony do biologicznego proces ewolucji - algorytm tworzy stopniowo coraz to lepsze rozwiązania. Ze względu na tą właściwość, można stwierdzić że algorytmy ewolucyjne bardzo dobrze nadają się do rozwiązywania problemów optymalizacyjnych. Stanowią one reprezentatywny przykład technik probabilistycznych, które używają wyboru losowego jako narzędzia do ukierunkowania procesu poszukiwań. Z powodzeniem stosowane są w praktyce inżynierskiej.

Sposób działania

Proces działania algorytmu ewolucyjnego można przedstawić jako kilka następujących po sobie kroków: algorytm ewolucyjny rozpoczyna przeszukiwanie od utworzenia populacji osobników, reprezentujących potencjalne rozwiązania, które są kodowane przez genotypy zawierające informację o osobnikach (potencjalnych rozwiązaniach).

W każdym ewolucyjnym kroku, nazywanym generacją lub pokoleniem, genotypy są dekodowane i ocenianie zgodnie z pewnym z góry przyjętym kryterium jakości nazywanym przystosowaniem (funkcją przystosowania może być na przykład funkcja celu), a następnie przeprowadzana jest selekcja w celu eliminacji osobników ocenionych jako najgorsze i pozostawienia tylko tych najlepszych. W procesie tym, genotypy najlepszej jakości (wykazujące się najlepszym wskaźnikiem przystosowania) podlegają mutacji oraz rekombinacji przeprowadzanej przy pomocy operatora krzyżowania. Procedura selekcji nie ma na celu wprowadzania do populacji nowych osobników - a co za tym idzie znajdowania nowych punktów w przestrzeni rozwiązań. Przeciwną właściwością cechują się krzyżowanie i mutacja. Dzięki zastosowaniu krzyżowania, proces ewolucji może być odpowiednio nakierowywany w stronę obiecujących obszarów przestrzeni rozwiązań. Mutacja pomaga zaś uniknąć zbieżności do lokalnego optimum. Ostatecznym wynikiem działania operatorów krzyżowania i mutacji jest stworzenie nowych rozwiązań w oparciu o które zbudowana zostanie populacja kolejnej generacji. Podobnie jak w przypadku innych heurystyk, warunek zakończenia może stanowić przykładowo pewna określona liczba iteracji, limit czasowy lub osiągnięcie zadowalającego poziomu przystosowania.

Jako formalne sformułowanie sposobu postępowania można przytoczyć: niech $P(t)$ oznacza populację w generacji t. W takim przypadku ogólny schemat działania algorytmu ewolucyjnego jest następujący [21]:

1. Niech $t = 0$
2. Wygeneruj i oceń początkową populację $P(t)$
3. Dopóki warunek stopu nie jest spełniony wykonuj:
   - 3.1. $t = t + 1$
   - 3.2. Wybierz $P(t)$ z $P(t-1)$
   - 3.3. Zmień $P(t)$ stosując operator krzyżowania i mutacji
   - 3.4. Oceń $P(t)$

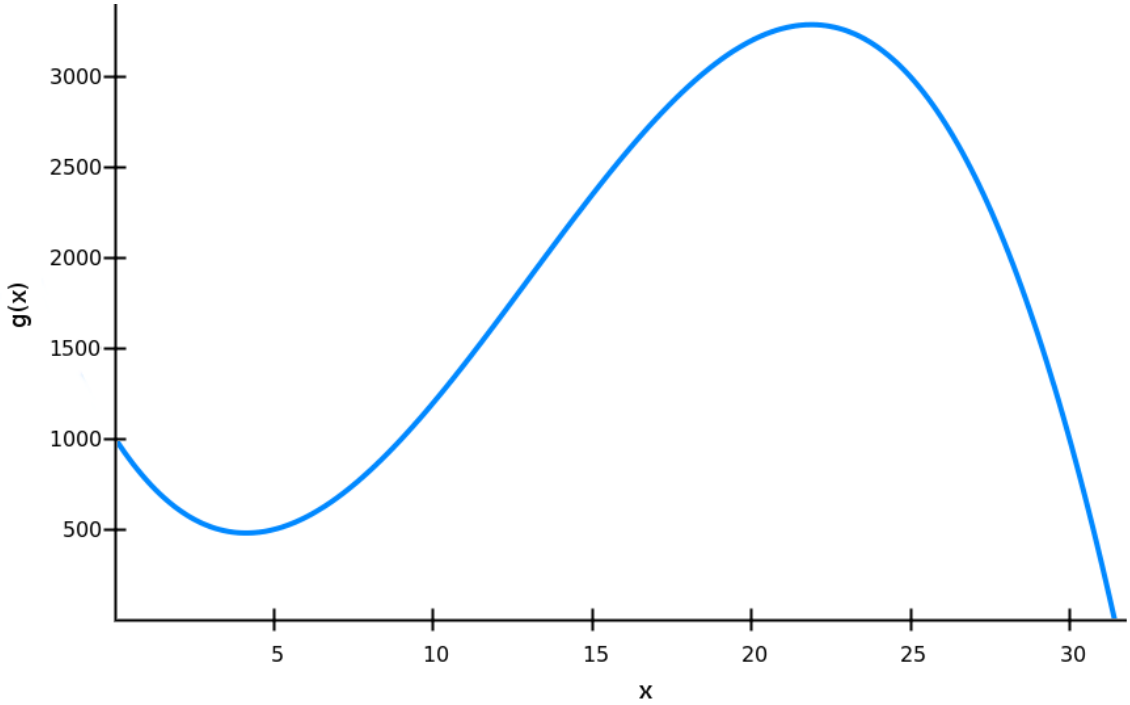
Pierwsze próby wykorzystania darwinowskiej definicji doboru naturalnego w rozwiązywaniu trudnych obliczeniowo problemów można dostrzec już w latach pięćdziesiątych XX-go wieku [22]. Lata sześćdziesiąte to już trzy różne propozycje zastosowania tej idei. Pojawiają się pierwsze propozycje na temat programowania ewolucyjnego, algorytmu genetycznego oraz strategii ewolucyjnych. Kolejne 15 lat to całkowicie niezależny rozwój tych technik. Od początku lat dziewięćdziesiątych zaczynają być one traktowane

jako różne formy jednej metodologii - algorytmów ewolucyjnych. Współcześnie wśród najbardziej rozpoznawanych można wyróżnić następujące typy algorytmów ewolucyjnych: algorytmy genetyczne, programowanie genetyczne, programowanie ewolucyjne oraz strategie ewolucyjne.

Algorytmy genetyczne

Algorytmy genetyczne stanowią to najbardziej rozpoznawana podgrupa algorytmów ewolucyjnych. Zgodnie ze wcześniejszym wprowadzeniem, genotypy reprezentowane są tutaj przez ciągi binarne o ustalonej długości (wśród spotykanych form kodowania należy wymienić również ciągi liczb całkowitych lub rzeczywistych). Ilość osobników w populacji jest stała, a ocena genotypów następuje w każdej generacji. Wymaganymi parametrami wejściowymi są: wielkość populacji, prawdopodobieństwo mutacji oraz warunek zakończenia działania algorytmu. Od samego początku musi być również zdefiniowana funkcja oceny. Celem lepszego przybliżenia sposobu działania algorytmów genetycznych, warto rozpatrzeć następujący przykład [25]:

Celem rozwiązania problemu jest maksymalizacja funkcji $g(x) = -x^3 + 39x^2 - 270x + 1000$ określonej na zbiorze liczb całkowitych z przedziału $[0,31]$. Wykres tej funkcji został przedstawiony na *Rysunku 2.3*. Należy rozpocząć od przyjęcia sposobu kodowania dla zmiennej za pomocą określonego ciągu znaków. Dobrym sposobem kodowania może być tutaj 5-cio pozycyjny ciąg binarny (składający się z 0 i 1). Każdy możliwa permutacja będzie reprezentować wartość zmiennej w systemie dwójkowym. Ciąg 00000 reprezentuje więc wartość 0, ciąg 00001 – wartość 1, a ciąg 11111 – wartość 31. Zgodnie z przyjętymi założeniami, jesteśmy więc w stanie operować na liczbach z zadanego zakresu – od 0 do 31. Działanie algorytmu genetycznego rozpoczyna się z wykorzystaniem losowej, początkowej populacji. Można ją otrzymać poprzez wygenerowanie genotypów w wyniku 5-ciu kolejnych rzutów monetą (1 - orzeł, 0 - reszka) dla każdego z genotypów. Na potrzeby przykładu można przyjąć, że tym sposobem została wygenerowana populacja o rozmiarze $n = 4$ (składająca się z 4 genotypów). Każdy z genotypów poddawany jest ocenie poprzez określenie odpowiadającej mu wartości funkcji celu.



Rysunek 2.3: Wykres $g(x) = -x^3 + 39x^2 - 270x + 1000$ na przedziale $[0, 31]$.

Opracowanie własne.

W dalszej kolejności z początkowej populacji tworzone są nowe, coraz lepsze osobniki, z wykorzystaniem podstawowych operatorów: selekcji, krzyżowania i mutacji. Selekcja stanowi tutaj procedurę, w czasie działania której genotypy są zwielokrotniane w stosunku odpowiadającym wartości funkcji przystosowania (która w tym przypadku odpowiada funkcji celu). Innymi słowy, zapewnia ona, że genotypy o wyższym przystosowaniu mają większe szanse na wprowadzenie swoich potomków do kolejnej populacji. Najczęściej spotykanym rodzajem selekcji jest ta oparta na zasadzie proporcjonalności do przystosowania. Procedurę tą można przyrównać do zastosowania wykalibrowanej tarczy ruletki, w której każdy sektor odpowiada pojedynczemu genotypowi, a jego rozmiar jest proporcjonalny do wartości względnego przystosowania dla danego genotypu. Taki rodzaj selekcji nazywany jest selekcją ruletkową [21]. Dla przedstawionego przykładu, ruletka uruchamiana jest czterokrotnie. Każdy wylosowany w ten sposób genotyp dostarcza swoją kopię, która wykorzystywana jest w puli, stanowiącej pokolenie rodzicielskie dla dalszych operacji genetycznych. Kolejnym krokiem jest przeprowadzenie operacji krzyżowania. Dokonywana jest ona z wykorzystaniem dwóch genotypów, losowo wybieranych z otrzymanej w poprzednim etapie puli. Zmiana polega na wyborze punktu krzyżowania c, odpowiedniego dla obu wybranych genotypów rodzicielskich oraz zamiany

miejscami wszystkich znaków od pozycji $c+1$ do pozycji g włącznie (gdzie za g przyjmowana jest długość genotypu) w obu egzemplarzach genotypów bazowych. Dzięki takiemu zabiegowi, otrzymywane są dwa nowe genotypy potomne. Proces krzyżowania przeprowadzany jest zazwyczaj z wykorzystaniem pewnej ustalonej ilości par rodzicielskich. Kolejnym krokiem jest mutacja, która stanowi operację, przeprowadzaną w sposób losowy dla każdego genu z osobna. Wpływ mutacji na proces ewolucji określany jest przez zadane prawdopodobieństwo, przykładowo jeśli dla przedstawionego przypadku przyjmiemy prawdopodobieństwo mutacji równe 0.05, to można oczekiwać, że dotknie ona 1 genu w całej populacji. Po zastosowaniu wszystkich operatorów spośród selekcji, krzyżowania i mutacji, nowe pokolenie poddawane jest ocenie i następuje zapętlenie procesu, tzn. obliczana jest wartość przystosowania dla genotypów z nowego pokolenia, tak aby możliwe było określenie puli rodzicielskiej dla następnej generacji. Wpływ na jakość wyników otrzymywanych dzięki zastosowaniu technik genetycznych ma zadana wielkość populacji, czas poświęcony na poszukiwanie rozwiązania, dobór sposobu selekcji, rodzaj wybranych operatorów krzyżowania i mutacji oraz ustalone prawdopodobieństwa, z jakimi procesy te są przeprowadzane. Wśród popularnych operatorów krzyżowania i mutacji, można wymienić także wiele innych poza tymi, przedstawionymi do tej pory. Jednym z bardziej popularnych sposób krzyżowania jest np. tzw. krzyżowanie dwupunktowe, przy zastosowaniu którego następuje wybór dwóch losowych punktów, pomiędzy którymi przeprowadzania jest wymiana odpowiednich fragmentów genotypów. Z kolei operacja mutacji może przykładowo przebiegać w oparciu o przemieszanie wartości dwóch losowo wybranych składowych genów. Dla pewnych specyficznych problemów, bardziej efektowne wyniki można uzyskać również przez zastosowanie innych rodzajów selekcji, niż przedstawiona powyżej selekcja ruletkowa. Przykładem może być selekcja turniejowa, która opiera się na wylosowaniu dwóch genotypów i wyborze jako rodzicielskiego, lepszego z nich. Należy również pamiętać, że sposobem reprezentacji mogą być zarówno ciągi binarne, jak i ciągi liczb całkowitych lub rzeczywistych.

Programowanie genetyczne

Programowanie genetyczne stanowi metodykę w dużym stopniu zbliżoną do algorytmów genetycznych. Technika ta wykorzystuje zasady genetyki i darwinowskiej natu-

ralnej selekcji do rozwiązywania zaawansowanych problemów obliczeniowych [28]. Zasadniczą różnicą, jaka występuję pomiędzy tymi dwoma podejściami, jest sposób reprezentacji potencjalnych rozwiązań, a co za tym idzie operacji na nich przeprowadzanych. W przeciwieństwie do przytoczonych algorytmów genetycznych, gdzie reprezentacja ta opiera się na ciągach liczb, tutaj mamy do czynienia z osobnikami kodowanymi za pomocą struktur drzewiastych oraz operatorami działającymi na gałęziach i węzłach tychże drzew.

Strategie ewolucyjne

Wśród strategii ewolucyjnych wymienić można dwie najważniejsze: $(\mu,\beta)ES$ oraz $(\mu+\beta)ES$ gdzie μ rodziców produkuje β potomków, opierając się przy tym na wykorzystaniu operatora krzyżowania i mutacji. W strategii $(\mu,\beta)ES$ najlepszych β potomków przeżywa i zastępuje rodziców. Efektem takiego sposobu postępowania jest brak obecności genotypów rodzicielskich w kolejnej populacji. Całkowicie przeciwnie działa $(\mu+\beta)ES$, gdzie możliwe jest przetrwanie zarówno osobników rodzicielskich jak i potomków. Przy wykorzystaniu $(\mu+\beta)ES$, zastosowanie znajduje tzw. strategia elitarna, opierająca się na kopiowaniu osobnika o najlepszej wartości przystosowania do kolejnej generacji. W obu wymienionych przypadkach przeprowadzane są również procesy krzyżowania i mutacji [36].

EMAS: Evolutionary Multi-Agent System

Interesującym podejściem do prowadzenia obliczeń z wykorzystaniem algorytmów ewolucyjnych jest również koncepcja EMAS [48]. Podejście to proponuje wykorzystanie algorytmów ewolucyjnych w sposób rozproszony. Autorzy opierają swoje rozwiązanie na wykorzystaniu technik wielo-agentowych do uruchamiania obliczeń w architekturach zdecentralizowanych. Każdy z osobników algorytmu genetycznego reprezentowany jest jako agent w terminologii systemu jako całości. Operują one w obrębie "wysp", które wskazują na rozproszony charakter obliczeń. Każdy z nich może wchodzić w interakcję z innymi agentami, ale ponadto dozwolona jest także migracja pomiędzy "wyspami" w celu umożliwienia przepływu informacji w obrębie całości systemu. Innowacyjnym

elementem tego podejścia jest fakt, że typowe dla agentów zachowanie, jakim jest komunikacja z innymi zostało rozszerzone o dodatkowe możliwości, jak tworzenie nowych agentów oraz śmierci (poprzez usunięcie z systemu). Początki koncepcji sięgają końca lat dziewięćdziesiątych [14]. W kolejnych latach system był stopniowo rozwijany, a w ostateczności znalazł zastosowanie dla wielu problemów optymalizacyjnych - zarówno dyskretnych, jak i ciągłych. Selekcja odbywa się tutaj w oparciu o nieodnawialne zasoby, w których posiadaniu jest każdy agent. Ilość tychże zasobów odpowiada jakości reprezentowanego rozwiązania problemu. Zasoby te przyznawane są za dążenie w kierunku rozwiązania lepszej jakości, natomiast zabierane za przeciwne zachowanie. Gdy ilość zasobów powiązanych z danym agentem osiągnie poziom zerowy, jest on usuwany z systemu. W przypadku agentów o dużej ilości zasobów, znacząco wzrasta prawdopodobieństwo wyboru ich w trakcie selekcji jako rodziców dla następnego pokolenia [2].

Przykłady zastosowań

Problemy z jakimi mamy do czynienia w rzeczywistych zastosowaniach, często charakteryzują się dużą złożonością (wiele ograniczeń i celów, które często stoją pomiędzy sobą w sprzeczności) oraz mogą być określane bardzo dużą ilością zmiennych (zarówno dyskretnych, jak i ciągłych). Niejednokrotnie zdarza się również, że są one zmienne w czasie (dynamiczne), w wyniku czego powstaje konieczność szybkiego otrzymywania dobrych rozwiązań.

Jednym z bardziej istotnych obszarów praktycznego wykorzystania algorytmów heurystycznych, jest harmonogramowanie zadań (np. procesów przemysłowych). Podgrupa problemów planowania i harmonogramowania, których podstawową rolą jest przydział odpowiednich zasobów do kolekcji zadań, stanowi skomplikowane zagadnienie ze względu na obecność różnego rodzaju ograniczeń i skomplikowanych struktur produktów. Techniki programowania matematycznego, które często są tutaj stosowane, okazują się być odpowiednie wyłącznie dla problemów o małych rozmiarach. Zazwyczaj w tego typu przypadkach cały harmonogram kodowany jest za pomocą pojedynczego genotypu - określa on kolejność wykonywania wszystkich czynności w danym procesie. Zależność ta implikuje wymóg stosowania odpowiednio dobranych operatorów genetycznych - muszą one zapewniać, aby w kolejnych generacja genotypy przechowywały tylko i wyłącznie rozwiązania dopuszczalne (np. spełniające wymóg braku powtórzeń lub zachowania pewnych relacji pomiędzy zadaniami).

Kolejnym ciekawym obszarem wykorzystania algorytmów genetycznych jest również przemysł chemiczny. Zakłady tego typu eksploatują specjalistyczne urządzenia takie jak pompy, destylatory i reaktory chemiczne, dostarczające skomplikowaną sieć wzajemnie sprzężonych strumieni informacji. Przeznaczone są one do przeprowadzania procesu chemicznego, podczas którego surowy materiał jest przekształcany w docelowy produkt. Zastosowanie optymalizacji w tego typu problemach sprowadza się do doboru parametrów operacyjnych w taki sposób, aby możliwe było znalezienie globalnego optimum dla zadanego procesu chemicznego. Typowo stosowane podejścia dla tego typu przypadków nie umożliwiają znajdowania rozwiązań w zadowalająco krótkim czasie. Oparcie się o techniki gradientowe skutkuje zbieganiem do optimów lokalnych i brakiem możliwości ich wykorzystania dla zadań nieróżniczkowalnych, które często tutaj występują. Skutkuje to dużym zainteresowaniem w obszarze zastosowania algorytmów ewolucyjnych, a w szczególności klasycznych algorytmów genetycznych dla przedstawionych problemów. Duża efektywność algorytmów genetycznych dla przytoczonych i podobnych problemów, pomimo występowania wielu zależności, wynika z faktu, że zazwyczaj w genotypie zakodowana jest jedynie minimalna wymagana część problemu. Skutkuje to stosunkowo niewielkim rozmiarem genotypu, co z kolei zwielokrotnia szanse szybkiego otrzymania akceptowalnej propozycji rozwiązania.

Jako kolejny dobry przykład wykorzystania algorytmów mrówkowych w procesach przemysłowych można przytoczyć projekt kontroli fermentacji piwa [13]. Zastosowanie to opiera się na doborze odpowiedniej temperatury w ustalonym okresie czasu.

2.2.2. Algorytmy mrówkowe

Podgrupę algorytmów heurystycznych, jakimi są algorytmy mrówkowe, można zdefiniować jako techniki stosowane przede wszystkim do poszukiwania dróg w grafie. Ze względu na metaheurystyczną charakterystykę, algorytmy mrówkowe nie są algorytmami, pomimo swojej nazwy. Swoje działanie opierają na tzw. zasadzie inteligencji roju. Jeden z pierwszych algorytmów mrówkowych został przedstawiony na łamach pracy doktorskiej Marco Dorigo w 1992. Jego pierwotnym zastosowaniem było poszukiwanie optymalnej drogi w grafie. Pierwsze implementacje nie były jednak zbyt wysoko cenione ze względu na szybką zbieżność do optimum lokalnego. Z czasem możliwe było jednak osiągnięcie bardziej dopracowanych, a co za tym idzie skutecznych wersji tego algo-

rytmu. Stosunkowo szybko doczekał się on uznania w szerszej gamie problemów z dziedzin harmonogramowania, poszukiwania partycji i pokrycia zbiorów, czy nawet wykrywania krawędzi na obrazach rastrowych. Typowo dla technik heurystycznych, algorytmy mrówkowe znajdują zastosowanie we wszystkich dziedzinach, gdzie rozwiązanie przybliżone jest zadowalające bądź zastosowanie algorytmów dokładnych okazuje się być niemożliwe, np. ze względu na zbyt duży koszt obliczeniowy.

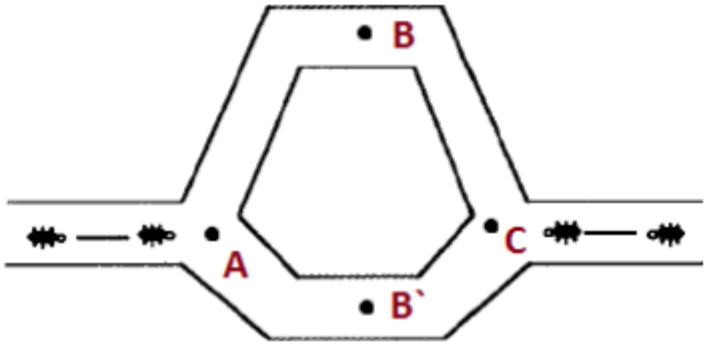
Sposób działania

Główną inspiracją dla algorytmu mrówkowego, było, jak sama nazwa wskazuje, zachowanie mrówek w ich naturalnym środowisku. Zasada działania tego algorytmu opiera się na obserwacji sposobu współpracy mrówek w trakcie procesu poszukiwania pokarmu w otoczeniu mrowiska. Heurystyka tego podejścia nie wykorzystuje więc matematycznego modelu i zależności, ale (podobnie jak w przypadku technik ewolucyjnych) obserwacje procesów zachodzących w naturze [24]. Podstawową obserwację stanowi fakt, że mrówki robotnice opuszczające mrowisko każdego ranka w poszukiwaniu pokarmu nie mają żadnej wiedzy na temat otaczającego ich środowiska. Zmysły, którymi dysponują są bardzo skromne, w związku z czym, dostarczają im wiedzy o bardzo niewielkiej jego części. Pomimo tego, są one w stanie wyżywić całe mrowisko, pracując w sposób zorganizowany. Stanem początkowym jest nowy dzień, który mrówka zaczyna o poranku. Nie ma ona żadnej wiedzy o otaczającym świecie, skutkiem czego swoją wędrówkę rozpoczyna w losowym kierunku. Jeśli po drodze natrafi na pożywienie w swoim najbliższym otoczeniu (w zasięgu swoich skromnych zmysłów), weźmie je i wróci z nim do mrowiska. Ma ona ułatwione zadanie z tym związane dzięki śladzie feromonowym, jaki pozostawia po sobie w trakcie początkowej wędrówki. Co ciekawe, natura pozwoliła mrówkom rozróżniać własny ślad od śladów innych osobników, dzięki czemu unika ona możliwości pomylenia drogi. Przyjmijmy, że druga mrówka opuszcza mrowisko po powrocie pierwszej. Kolejne mrówki, które opuszczają mrowisko po powrocie innych mają również dzięki temu ułatwione zadanie. Co prawda wciąż nie mają one świadomości całego otaczającego świata, ale mogą wykorzystać ślad feromonowy pozostawiony przez swoje prekursorki (wiedząc, że prowadzi on do źródła pożywienia). Dzięki tej właściwości, podejmowanie decyzji o kierunku poruszania się jest dla nich znacznie ułatwione. Im

bardziej intensywny ślad feromonowy, tym większe prawdopodobieństwo wyboru podobnej ścieżki przez kolejne mrówki. Na poziom intensywności feromonu składają się dwa czynniki, oba związane z parowaniem feromonu z upływem czasu [17]:

- Odległość od mrowiska do żerowiska. Im mniejsza, tym krótszą drogę mrówki muszą pokonać, przez co mniejsza ilość feromonu może wyparować.
- Częstotliwość wyboru danej ścieżki przez różne osobniki. Im większa, tym więcej feromonu jest na niej pozostawiane.

W początkowej fazie, wpływ feromonu na podejmowane przez mrówki decyzje jest oczywiście niewielki. Z czasem jednak, każda kolejna mrówka opuszczająca mrowisko, pozostawia swój ślad feromonowy, który sumuje się z już pozostawionym na danej ścieżce. Jak zostało to wyżej wspomniane, poziom intensywności feromonu na ścieżce jest czynnikiem złożonym. Główny wpływ ma na niego fakt parowania, który powoduje jego osłabianie.



Rysunek 2.4: Przykładowe sposoby poruszania się mrówek po ścieżkach [49]

Dobrym przykładem może być uogólnienie tych zasad na sytuację, w której kilka osobników mrówek porusza się po wielu niezależnych ścieżkach [30]. W rzeczywistym świecie mogą istnieć dwie różne drogi, które prowadzą do tego samego źródła pożywienia, jednak jednak z nich (ABC) jest zdecydowanie dłuższa niż druga $(AB'C)$ (*Rysunek 2.4*). Z oczywistych powodów, mrówki które wybiorą dłuższą ścieżkę, pozostawią mniej intensywny ślad dla swoich następczyń ze względu na dłuższy czas podróży powrotnej do mrowiska. Prawdziwego znaczenia fakt ten nabiera przy większej ilości osobników poszukujących ścieżki, ze względu na ułatwienie (bądź jego brak) zadania poszukiwania właściwej ścieżki dla mrówek wyruszających w późniejszym czasie.

Procedura algorytmu

Sposób postępowania mrówek w naturalnym środowisku, przedstawiony w poprzednim rozdziale sugeruje wyraźnie, że nie koniecznie kierują się one poszukiwaniem rozwiązania najlepszego, ale zadowalają się dostatecznie dobrym - takim, które jest w ich zasięgu, a jednocześnie jest w stanie dostarczyć odpowiedniej ilości pożywienia. Znacząca większość osobników z grupy obiera kilka wytartych, ustalonych ścieżek, natomiast tylko pojedyncze decydują się na wybór bardziej niestandardowych możliwości. Podczas rozwiązywania problemów o skomplikowanym modelu matematycznym istotne jest jednak osiągnięcie wysokiej różnorodności w przeszukiwaniu przestrzeni potencjalnych rezultatów. Chcemy, aby sprawdzona została jak największa część przestrzeni, obejmująca potencjalnie wysokie jakościowo rozwiązania. Istotne jest więc tutaj uniknięcie optimów lokalnych [19]. Należy tutaj pamiętać, że algorytm mrówkowy ze swej natury nie zawsze zwraca rozwiązanie najbardziej optymalne. Należy jednak dążyć do osiągnięcia jak najlepszego rezultatu. W ogólności sposób postępowania tej heurystyki można przedstawić następująco (przy założeniu, że t to kolejny numer dnia poszukiwania najlepszego źródła pożywienia przez mrówki):

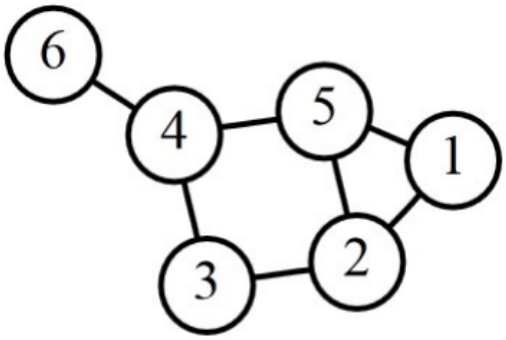
1. Niech $t = 0$
2. Wygeneruj kolonię mrówek
3. Dopóki warunek stopu nie jest spełniony, wykonuj:
   3.1 $t = t + 1$
   3.2 Wygeneruj potencjalne rozwiązania (ścieżki)
   3.3 Zastosuj parowanie feromonów ze ścieżek
   3.4 Nanieś nowe feromony na ścieżki

W kolejnych podrozdziałach zostanie przedstawione bardziej szczegółowe omówienie poszczególnych etapów działania algorytmu mrówkowego.

Generowanie rozwiązania

Jest to najważniejszy etap działania algorytmu mrówkowego. Odzwierciedla on obserwacje związane z rozchodzeniem się mrówek w środowisku wokół mrowiska. W odróżnieniu od zachowania mrówek w naturze, działanie algorytmu odbywa się w sposób iteracyjny. W trakcie trwania każdej z iteracji, każda mrówka z kolonii generuje pełne, pojedyncze możliwe rozwiązanie zadanego problemu. Słowo “generowanie” jest tutaj

istotne ze względu na fakt, że wybór konkretnych ścieżek w dużym stopniu zależy od czynników probabilistycznych i parametrów wejściowych.



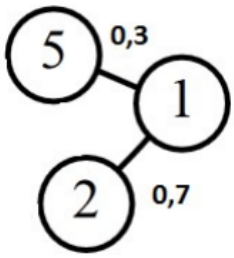
Rysunek 2.5: Przykład grafu stanowiącego podstawę problemu przeszukiwania [49]

Za przykładowy problem można przyjąć graf przedstawiony na *Rysunku 2.5* i poszukiwanie ścieżki pomiędzy węzłami nr 1 i 6. W opisywanej fazie każda z mrówek będzie miała za zadanie pokonanie tej trasy w wybrany przez siebie sposób. Podróż rozpoczyna się zawsze w węźle 1. Pierwszą decyzją jest wybór spomiędzy węzłów 2 lub 5. Prawdopodobieństwo przejścia z węzła i do węzła j dla $k-tej$ mrówki można zdefiniować następująco [18] *(10)*:

$$p_{i,j}^{k}=\begin{cases}\dfrac{(\tau_{ij})^{\alpha}(\sigma_{ij})^{\beta}}{\sum_{h\notin tabu_k}(\tau_{ij})^{\alpha}(\sigma_{ij})^{\beta}}, & \text{gdy j}\in \text{tabu}_k \\ 0, & \text{w przeciwnym przypadku}\end{cases} \qquad (10)$$

Gdzie : τ_{ij} to ilość feromonu na łuku z węzła i do j, a z kolei współczynnik σ_{ij} to wartość funkcji atrakcyjności przejścia z i do j. Jest ona ściśle związana z konkretnym problemem optymalizacyjnym. W przypadku poszukiwania najkrótszej drogi w grafie ważonym, będzie ona przyjmować najwyższe wartości na tych krawędziach, dla których waga jest najniższa. Dzięki temu uzyskiwana jest preferencja wyboru ścieżek, które są krótsze. Współczynniki α,β stanowią o istotności wpływu powyższych dwóch funkcji na podejmowane decyzje. Dzięki temu możliwe jest określenie, czy decyzje powinny być podejmowane bardziej ze względu na wartości feromonów na ścieżkach czy wiedzy dotyczącej dziedziny problemu. Należy tutaj zaznaczyć, że nie istnieje uniwersalny dobór tych parametrów i w dużej mierze zależy on od konkretnego problemu. Zbiór tabuk to

węzły odwiedzone już przez mrówkę w danej iteracji. Zależność ta pozwala uniknąć zapętleń w trasie. Ze względu na przykładowy charakter opisywanego przypadku, nie jest istotne w jaki sposób dobierane są prawdopodobieństwa dla poszczególnych ścieżek (jest to zależne głównie od konkretnego problemu) - ważny jest natomiast fakt, że decyzje podejmowane są z uwzględnieniem wartości feromonowych oraz wiedzy o strukturze grafu. Przyjmijmy rozkład prawdopodobieństwa ścieżek jak na *Rysunku 2.6*. Aby określić do jakiego węzła przejdzie mrówka, musimy wygenerować liczbę losową z przedziału. Wyższa wartość prawdopodobieństwa nie oznacza wcale, że mrówka musi przejść po „bardziej prawdopodobnym" łuku. W tym miejscu objawia się probabilistycznych charakter tej heurystyki. Losowana jest liczba z przedziału $< 0;1 >$ i to ona określa podstawę do podjęcia decyzji o następnym węźle. Na wynik losowania mają wpływ wszystkie opisane do tej pory parametry, dlatego w pewnych sytuacjach może dojść do wyboru ścieżki o mniejszej wartości prawdopodobieństwa i eksploracji rozwiązań do tej pory niesprawdzonych.



Rysunek 2.6: Przykładowy rozkład prawdopodobieństwa obrania ścieżek przez mrówkę [49]

Można przyjąć, że w toku działania algorytmu, mrówka znajdzie się w węźle nr 2. Procedura ta będzie powtarzana dla kolejnych węzłów, aż do wystąpienia warunku końcowego, którym w omawianym przykładzie jest dotarcie do węzła nr 6.

Aktualizacja ścieżki feromonów

Po zakończeniu pełnej iteracji generowania ścieżek, w której mrówki wybrały swoje rozwiązania, następuje aktualizacja wartości feromonów. W przeciwieństwie do świata rzeczywistego, mrówki nie zostawiają śladów w trakcie podróży po węzłach, ale ma to miejsce dopiero po określeniu wybranych tras dla każdego z osobników. Zależność ta wynika z chęci promowania rozwiązań lepszych jakościowo już w trakcie działania

algorytmu. Opiera się to na pozostawianiu większej ilości feromonu na krótszych ścieżkach, dzięki czemu będą one chętniej wybierane przez kolejne mrówki. W efekcie, po zastosowaniu parowania, na tychże ścieżkach pozostanie więcej feromonu niż na innych. Odwzorowywane w ten sposób jest szybsze parowanie feromonów ze ścieżek dłuższych występujące w środowisku naturalnym (ze względu na zachodzenie tego procesu w ciągłej przestrzeni czasu). Formułę opisującą aktualizację feromonu pomiędzy węzłami i oraz j dla mrówki k można przedstawić następująco [18] *(11)*:

$$\tau_{ij}^{k} := (1-\rho)\tau_{ij}^{k} + \tau_{ij}^{k} \qquad (11)$$

$$\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k}, \text{ gdy } node_k \in sol_k \\ 0, \text{ w przeciwnym przypadku} \end{cases}$$

Jest to najbardziej ogólna forma tej zależności, nieprzystosowana pod żaden konkretny problem. W powyższych równaniach parametr ρ określa szybkość parowania feromonu. Współczynnik Q to pewna dobrana dla konkretnego problemu stała (parametr procedury), L_k to koszt rozwiązania związanego z mrówką k, a z kolei sol_k to rozwiązanie związane z mrówką k. W przypadku przykładowego problemu poszukiwania optymalnej ścieżki w grafie, L_k może odpowiadać długości danej ścieżki.

2.2.3. Algorytmy hybrydowe

Wśród popularnie stosowanych technik warto również przytoczyć techniki hybrydowe. W ostatnich latach możliwe jest odnotowanie dużego wzrostu zainteresowania tego typu algorytmami wśród badaczy. Powodem tej sytuacji jest fakt, że pozwalają one osiągać stosunkowo najlepsze wyniki dla większości powszechnie rozważanych problemów optymalizacyjnych. Główną ideą hybrydyzacji jest odpowiednie połączenie różnych technik metaheurystycznych w sposób pozwalający na ich kooperację [12]. Wśród najczęściej łączonych algorytmów wymienić należy techniki gradientowe, symulowane wyżarzanie, przeszukiwanie tabu czy algorytmy ewolucyjne [44]. Częstym zabiegiem jest także przeprowadzanie dodatkowych operacji na danych otrzymywanych z poszcze-

gólnych podsystemów składowych, jak np. generowanie rozwiązań pośrednich na podstawie wybranych rezultatów danego podsystemu [41]. Dobrze zbadaną podgrupą technik hybrydowych, które również dodatkowo wpływają na postać przetwarzanych informacji są algorytmy memetyczne [32]. Swoją popularność algorytmy hybrydowe zawdzięczają w dużej mierze możliwości wykorzystania istniejących rozwiązań w nowy sposób, pozwalający na osiąganie znacznie lepszych rezultatów. Kolejne podrozdziały prezentują szczegółową klasyfikację tych technik.

Klasyfikacja metod hybrydowych

Podejścia hybrydowe mogą zostać sklasyfikowane w następujący sposób, w oparciu o szczegóły związane ze sposobem działania [44]:

- **niskopoziomowe** - podgrupa ta skupia metody, dla których wybrana funkcja danej metaheurystyki może zostać w całości zastąpiona przez inną metaheurystykę.
- **wysokopoziomowe** - w przeciwieństwie do poprzedniego podejścia, w tym przypadku nie występuje ingerencja w wewnętrzne szczegóły działania danej metaheurystyki. Każda z nich działa w sposób niezależny.

Możliwy jest także podział ze względu na sposób wykonywania poszczególnych składowych. Wyróżnia się systemy:

- **sztafetowe** - uruchamiające metaheurstyki w kolejności jedna po drugiej
- **grupowe** - prowadzące obliczenia w sposób równoległy

Ciekawym sposobem klasyfikacji jest także wyróżnienie podgrup łączących obie te cechy [43][44]:

- **hybrydy niskopoziomowe, sztafetowe (ang. *Low-level Relay Hybrid - LRH*)** Podgrupa ta definiowana jest jako hybrydyzacja metaheurystyk z podgrupy dających pojedyncze rozwiązanie, jak np. symulowane wyżarzanie z przeszukiwaniem lokalnym. Koncepcja ta obejmuje zastąpienie wybranego kroku pierwszej ze składowej (w przedstawianym przykładzie symulowanego wyżarzania) przez deterministyczny proces poszukiwania lokalnego. W odróżnieniu od standardowego działania symulowanego wyżarzania, w tym przypadku wprowadzany jest dodatkowy krok, który pozwala łańcuchowi Markova na eksplorację optimów lokalnych. Efekt działania tego dodatkowego kroku poddawany jest testowi, od którego zależy dalszy sposób postępowania w obliczeniach.

- **hybrydy niskopoziomowe, grupowe (ang. *Low-level Teamwork Hybrid - LTH*)**
  Charakteryzuje je obecność dwóch przeciwstawnych celów - eksploracji i eskploatacji. Eksploracja jest konieczna ze względu na potrzebę sprawdzenia wszystkich możliwych rozwiązań z całej przestrzeni w celu znalezienia globalnego optimum o odpowiednio wysokiej jakości. Eksploatacja wprowadza prace nad wybranym rozwiązaniem w celu znalezienia jeszcze lepszego i bardziej dokładnego. Heurystyki oparte o działanie populacji (jak algorytmy ewolucyjne czy mrówkowe) są tutaj bardzo efektywne w kategoriach eksploracji. Znacznie gorzej wypadają natomiast pod względem eksploatacji. Przeciwnie zachowują się natomiast algorytmy przeszukiwania lokalnego (jak symulowane wyżarzanie czy przeszukiwanie tabu). Ich połączenie daje dobre wyniki ze względu na wzajemnie uzupełnianie się - metody populacyjne osiągają wysokiej jakości rezultaty w optymalizacji globalnej (eksploracji), które następnie poprawiane są poprzez przeszukiwanie lokalne (eksploatację).
- **hybrydy wysokopoziomowe, sztafetowe (ang. *High-level Relay Hybrid - HRH)***
  Zgodnie z przedstawioną powyżej charakterystyką poszczególnych metod klasyfikacji, w tym przypadku kilka zaenkapsulowanych algorytmów metaheurystycznych uruchamiane jest w odpowiedniej kolejności, operując na danych wynikowych swoich poprzedników w sekwencji. Często łączonymi metodami są w tym wypadku przeszukiwanie lokalne, którego efekty działania przekazywane są następnie do algorytmu ewolucyjnego, po którym uruchamiane jest z kolei przeszukiwanie tabu. Algorytm ewolucyjny pozwala tutaj na szybką lokalizację obiecujących obszarów spośród rozległych i skomplikowanych przestrzeni rozwiązań. Na tak ograniczonych przestrzeniach, uruchamiany jest algorytm poszukiwania lokalnego, który umożliwia łatwe znalezienie wysokiej jakości rozwiązania problemu. Znacząca jest w tym wypadku obserwacja, że algorytmy ewolucyjne po pewnym czasie przestają zwracać rozwiązania o lepszych wartościach funkcji oceny. Wyjście z takiej sytuacji jest mało prawdopodobne. Hybrydyzacja pozwala rozwiązać ten problem.
- **hybrydy wysokopoziomowe, grupowe (ang. *High-level Teamwork Hybrid - HTH)***

  Główną ideą tej podgrupy metod hybrydowych jest działanie wielu niezależnych metaheurystyk w sposób równoległy oraz ich wzajemna kooperacja w celu znalezienia jak najwyższej jakości optimum globalnego. Zgodnie z intuicją, ostateczny

wynik obliczeń będzie w najgorszym wypadku tak dobry jak dla pojedynczego algorytmu działającego bez jakiejkolwiek współpracy z innymi. Często jednak możliwe jest osiągnięcie lepszych rezultatów poprzez wymianę informacji pomiędzy składowymi w celu wzajemnego wsparcia. Dobrym przykładem takiego działania jest rozproszenie osobników algorytmu genetycznego pomiędzy kilka niezależnie funkcjonujących "wysp". Osobniki mogą migrować pomiędzy poszczególnymi "wyspami", zapewniając wymianę informacji. W takim wypadu, jakość otrzymywanych wyników zależy od wielu czynników, takich jak struktura topologiczna obszaru działania (sieć połączeń pomiędzy "wyspami"), ilość migrujących osobników oraz częstotliwość występowania tego procesu. Możliwe jest przykłodowo zrównoleglenie w 4-wymiarowym hipersześcianie [45].

Algorytmy memetyczne

Techniki memetyczne są jednymi z najchętniej wybieranych przez naukowców metaheurystykami hybrydowymi w ostatnich latach. Z tego powodu można je uznać za najbardziej reprezentatywny przykład tego podejścia. Opierają się one najczęściej o połączenie algorytmów ewolucyjnych z przeszukiwaniem lokalnym. Podstawą ich działania jest wykorzystanie systemów wieloagentowych i szczegółowej wiedzy domenowej na temat rozwiązywanego problemu. Drugi z tych czynników jest szczególnie istotny i nie może być on traktowany w sposób opcjonalny, na co zwraca uwagę autor [16]. Podkreśla on także, że w technikach tych, dane obliczeniowe są przekazywane i w dużym stopniu modyfikowane lub rozszerzane przez poszczególne składowe. Wskazuje także, że techniki memetyczne mogą być postrzegane jako systemy populacji agentów, które współpracują i rywalizują pomiędzy sobą. Realizacją czynnika wiedzy domenowej może być wspomniane przeszukiwanie lokalne, aproksymacja lub w pewnych przypadkach także algorytm dokładny. Pierwsze wzmianki na temat technik memetycznych, które w sposób najbardziej reprezentatywny opisują ideę hybrydyzacji metaheurystycznej notuje się na koncówkę lat osiemdziesiątych [35]. Początkowe implementacje opierały się przede wszystkim o połączenie algorytmu genetycznego i symulowanego wyżarzania. Jedną z podstawowych motywacji dla jego powstania było poszukiwanie sposobu na uniknięcie ograniczeń związanych ze składowymi algorytmami w rozwiązywaniu problemu komiwojażera. Kolejne lata owocują w nowe implementacje, które dostarczają rozwiązań wielu problemów NP-trudnych.

W literaturze klasyfikuje się 3 generacje algorytmów memetycznych [33]. Obecnie stosowane są implementacje wykorzystujące przede wszystkim jedną z dwóch technik przeszukiwania lokalnego: przeszukiwanie Baldwinowskie lub Lamarckinowskie [12]:

- **przeszukiwanie Baldwinowskie** - teoria związana w dużym stopniu z procesem ewolucji, oparta przede wszystkim na założeniu, że w trakcie powstawania nowych generacji, możliwe jest przekazywanie pewnych właściwości do kolejnych pokoleń (poprzez dziedziczenie) [7]. Wykorzystanie tej metody przeszukiwania w technice hybrydowej nie wpływa bezpośrednio na postać genotypu, którego ono dotyczy. Zamiast tego, procedura selekcji genotypów operuje na wartości funkcji oceny uruchamianej na genotypach poprawionych przez ten typ przeszukiwania lokalnego. W ten sposób do kolejnego pokolenia przekazywane są tylko osobniki, mogące dać początek populacji o wyższej jakości.
- **przeszukiwanie Lamarckinowskie -** technika oparta na teorii biologicznej, mówiącej o możliwości dziedziczenia właściwości nabytych w trakcie rozwoju i życia rodziców, przez ich potomków [11]. Przeszukiwanie to uruchamiane jest w ramach operacji mutacji lub krzyżowania w algorytmie ewolucyjnym. Proces jego działania opiera się na wielokrotnych próbach uruchamiania krzyżowania na różnych genotypach, w ostateczności doprowadzając do wyboru możliwości dających najlepsze rezultaty. Z pokolenia na pokolenie wprowadzane jest usprawnienie genotypów poszczególnych osobników, które przekazywane jest w ramach dziedziczenia.

Przykłady zastosowań

Algorytmy hybrydowe cieszą się dużą popularnością wszędzie tam, gdzie powszechnie stosowane techniki metaheurystyczne nie są w stanie poradzić sobie z uzyskiwaniem wystarczająco dobrych wyników lub obliczenia są zbyt kosztowne pod względem czasu obliczeń.

Ciekawym obszarem zastosowań jest tego typu rozwiązań są zadania związane z podziałem grafu [9]. Stanowią one grupę problemów, obejmujących wiele dziedzin współczesnej informatyki, jak eksploracja danych, projektowanie układów VLSI, czy segmentacja obrazu. Zadania te można sklasyfikować jako NP-zupełne, co sprawia że metody aproksymacyjne wydają się być najbardziej stosowne do ich rozwiązywania. Możliwe jest odnotowanie wielu różnych podejść do tego obszaru, jak algorytmy za-

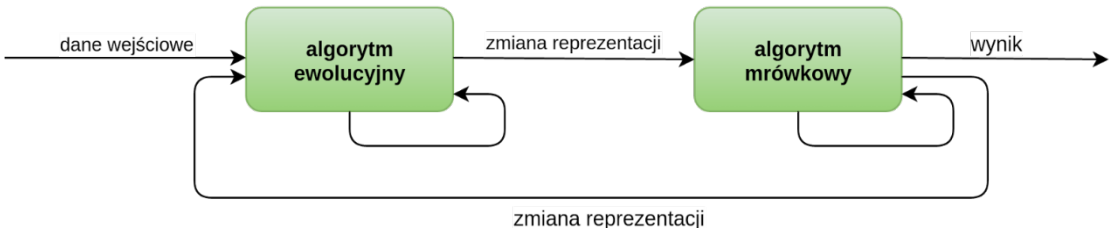
chłanne, mrówkowe, genetyczne, czy przeszukiwanie tabu. Interesujące podejście prezentuje jednak przywołana publikacja. Zauważają oni, że większość znanych technik działa w krótkim czasie, ale dostarcza rozwiązań jedynie średniej jakości, podczas gdy inne znajdują rozwiązania bardzo optymalne, ale kosztem dłuższego czasu obliczeń. Aby uniknąć konieczności wyboru pomiędzy tymi dwiema skrajnościami, przeprowadzają oni przegląd możliwości zastosowania algorytmów hybrydowych w tym obszarze, tak aby być w stanie uzyskać wysokiej jakości wyniki w odpowiednio krótkim czasie.

Szerokim spektrum niebanalnych zagadnień, w których metaheurystyki okazują się być cennym podejściem okazują się być także medycyna oraz biocybernetyka. Świetnymi efektami w klasyfikacji danych medycznych, które często bywają niespójne, niekompletne lub niedokładne, pochwalić mogą się metaheurystyki hybrydowe [6]. Podobny rodzaj problemów często rozwiązywany jest także przez sztuczne sieci neuronowe. Dla lepszych efektów, proces ich uczenia może być wspierany przez inne algorytmy z wykorzystaniem hybrydyzacji [38]. Również tak poważne tematy medyczne jak planowanie dawkowania podczas leczenia raka w oparciu o chemioterapię z wykorzystaniem łączenia różnych rodzajów leków wspierane są przez techniki hybrydowe [46]. Intensywnie eksploatowane są one również przez linie lotnicze, co wprowadza często wielomilionowe oszczędności finansowe [20].

3. Ewolucyjno-mrówkowy system optymalizacji dyskretnej

3.1. Proponowana koncepcja systemu

Obecnie dostępne algorytmy heurystyczne, zarówno z grupy systemów ewolucyjnych, jak i mrówkowych posiadają implementacje wysoce zaawansowane, mocno zoptymalizowane pod obliczenia w popularnych współcześnie architekturach równoległych i rozproszonych. Fakt ten pozwala stwierdzić, że poszukiwanie usprawnień w działaniu każdego z nich może być trudne, czasochłonne i z wysokim prawdopodobieństwem nie przynieść pożądanych efektów. Każdy z nich posiada jednak pewne wady, a więc obszarem, na którym warto się skupić jest możliwość ich ograniczenia lub w miarę możliwości nawet wyeliminowania. Głównym problemem, jaki może wystąpić w obu przypadkach jest szybkie zgubienie różnorodności przeszukiwanych rozwiązań i skupienie się obliczeń na jednym wybranym, niekoniecznie najlepszym z nich. Propozycją rozwiązania tej sytuacji, jaką chciałbym przedstawić jest wzajemne połączenie systemów opartych o oba te algorytmy, w taki sposób, aby możliwe stało się dłuższe, a zarazem bardziej dokładne poszukiwanie odpowiedzi dla zadanego problemu. Podejście takie może zaowocować wzajemną eliminacją problemu braku różnorodności. Idea takiego systemu została przedstawiona na *Rysunku 3.1*.



Rysunek 3.1: Diagram idei rozwiązania postawionego problemu. Opracowanie własne.

Jak można zaobserwować, sposób postępowania opiera się na wykorzystaniu dowolnych, dostępnych już wcześniej implementacji algorytmów genetycznego i mrówkowego. Proponowana koncepcja szereguje obliczenia w każdym z nich, jednocześnie przekazując pomiędzy nimi ich wzajemne wyniki działania. Pierwszym etapem jest uruchomienie obliczeń w algorytmie ewolucyjnym. Długość ich trwania (ilość powtórzeń) uzależniona jest od zadanego warunku stopu. Otrzymane wyniki są następnie wykorzystywane do zmiany reprezentacji, odpowiedniej dla operowania na nich przez algorytm mrówkowy – proces ten stanowi jeden z dwóch filarów proponowanej koncepcji. Cel ten osiągany jest w sposób, który pozwala na rozpoczęcie obliczeń w algorytmie mrówkowym z narzuconym ustalonym stanem wejściowym systemu - podobnie do sytuacji, gdyby obliczenia w tym systemie trwały już od jakiegoś czasu. Osiągnięcie takiego stanu możliwe jest przez uruchomienie kilku pierwszych kroków algorytmu ze z góry narzuconymi ścieżkami mrówek (wynikającymi z otrzymanych wcześniej wyników), bez możliwości podejmowania decyzji o trasie przejścia przez algorytm. Dzięki takiemu zabiegowi, feromony na ścieżkach zostają naniesione z odpowiednią intensywnością i możliwe jest rozpoczęcie typowego działania tego podsystemu. Po przekazaniu tak przygotowanych danych do algorytmu mrówkowego, następuje kolejna porcja iteracji, w ilości ponownie uzależnionej od zdefiniowanego dla niego warunku stopu. Zakończenie obliczeń na tym etapie, równoznaczne jest ze sprawdzeniem globalnego warunku stopu. Jeśli jest taka potrzeba, nastąpić powinno ponowne przekształcenie reprezentacji danych, tak aby możliwe było rozpoczęcie całego procesu od początku. W przeciwnym przypadku, wynik dostępny w danym momencie jako najlepszy, zwracany jest jako ostateczny. Obrazowy schemat działania tej procedury przedstawiony jest przez *Pseudokod 3.1*.

```
hybryda(globalny_warunek_stopu, warunek_stopu_alg_ewolucyjnego, warunek_stopu_alg_mrówkowego, problem):
  początkowa_populacja_genotypowa := inicjuj(problem)
  dopóki globalny_warunek_stopu nie jest spełniony wykonuj:
    populacja_genotypowa := algorytm_ewolucyjny(warunek_stopu_alg_ewolucyjnego, początkowa_populacja_genotypowa)
    początkowa_tablica_feromonów := transformuj_reprezentację(populacja_genotypowa)
    tablica_feromonów := algorytm_mrówkowy(warunek_stopu_alg_mrówkowego, początkowa_tablica_feromonów)
    jeżeli globalny_warunek_stopu jest spełniony to:
        zwróć rezultat_o_najmniejszym_koszcie(tablica_feromonów)
    w przeciwnym razie:
        początkowa_populacja_genotypowa := transformuj_reprezentację(tablica_feromonów)
```

Pseudokod 3.1: Idea działania hybrydy ewolucyjno-mrówkowej.

Wspomniana zmiana kodowania informacji pomiędzy algorytmami mrówkowym, a ewolucyjnym stanowi drugi z filarów hybrydyzacji. W ramach niniejszej pracy, zaproponowane zostaną trzy sposoby na osiągnięcie tego celu. Architektura systemu pozwala jednak na łatwe zastosowanie również innych implementacji.

Pierwsza z proponowanych metod opiera swoje działanie na celowym zaburzaniu ścieżki, jaką podążyłaby mrówka w normalnych warunkach. W celu wygenerowania docelowego zestawu permutacji, po zakończeniu obliczeń uruchamiany jest dodatkowy krok algorytmu mrówkowego. Mrówki wybierają ścieżki w typowy dla siebie sposób, w oparciu o intensywność feromonów. Przy każdym z wyborów dalszego odcinka ścieżki, dokonywana jest jednak randomizacja binarna, tożsama z rzutem monetą. W zależności od jej wyniku, wybór ten jest akceptowany, bądź nie. W drugim przypadku, ścieżka taka wykluczana jest z możliwych połączeń w danym kroku i następuje ponowny wybór połączenia w zależności od jego atrakcyjności. Taki sposób postępowania powtarzany jest aż do momentu wygenerowania odpowiedniej ilości permutacji, które stanowić będą podstawę nowej populacji osobników dla algorytmu ewolucyjnego. Warto zauważyć, że żadne z przejść mrówek po ścieżkach na tym etapie nie ma wpływu na intensywność feromonów. Nie są one zwiększane, ani nie następuje parowanie. Idea działania tego sposobu zmiany reprezentacji została przedstawiona przez *Pseudokod 3.2*.

```
transformuj_reprezentację_przez_rzut_monetą(tablica_feromonów):
    i := pobierz_skonfigurowaną_wielkość_populacji()
    wielkość_problemu := pobierz_wielkość_problemu(tablica_feromonów)
    system_mrówkowy := inicjuj()
    populacja_genotypów = inicjuj_pustą_populację()
    dopóki i > 0 wykonuj:
        j := wielkość_problemu
        genotyp := inicjuj_pusty_genotyp()
        dopóki j > 0 wykonuj:
            ścieżka := system_mrówkowy.krok_bez_zmiany_wartości_feromonów(tablica_feromonów)
            jeżeli losuj(0,1) == 1 to:
                ścieżka := system_mrówkowy.krok_bez_zmiany_wartości_feromonu(tablica_feromonów)
            genotyp := genotyp.rozszerz(ścieżka.cel.id)
            j := j-1
        populacja_genotypów.rozszerz(genotyp)
        i := i-1
    zwróć populacja_genotypów
```

Pseudokod 3.2: Idea działania transformacji przez rzut monetą.

Druga z proponowanych metod przekształceń opiera się przede wszystkim na parowaniu feromonów ze ścieżek, podobnym do tego, które występuje w kolejnych krokach wykonywania algorytmu mrówkowego, z zastrzeżeniem, że w tym wypadku odbywa się to w bardziej nietypowy, kontrolowany sposób. Podobnie jak w poprzednim przypadku, również i tutaj wygenerowanie nowych ścieżek, stanowiących podstawę dla populacji genotypowych, opiera się na kilku dodatkowych przejściach mrówek po ścieżkach. Wpływ na wybór połączeń przez algorytm odbywa się tym razem jednak poprzez odparowywanie części feromonów z danego połączenia po każdym jego wybraniu. Zabieg taki skutkuje zmianą jego poziomu atrakcyjności dla mrówek, które będą je rozważać w kolejnych krokach. Warto zauważyć, że podobnie jak przy pierwszej metodzie, intensywność feromonów na pozostałych ścieżkach nie jest w żaden sposób zmieniana. Rozwiązanie takie sprawia, że wraz z wykonywaniem kolejnych kroków, otrzymywane w wyniku ścieżki zaczynają się co raz bardziej różnić względem wcześniej wybranych. Dzięki tej właściwości, wzrasta różnorodność genotypów w poszczególnych populacjach algorytmu ewolucyjnego, co w efekcie prowadzi do możliwości dłuższego prowadzenia procesu ewolucji. Metoda ta zobrazowana jest przez *Pseudokod 3.3*.

```
transformuj_reprezentację_przez_parowanie_feromonów(tablica_feromonów):
    i := pobierz_skonfigurowaną_wielkość_populacji()
    wielkość_problemu := pobierz_wielkość_problemu(tablica_feromonów)
    system_mrówkowy := inicjuj()
    populacja_genotypów = inicjuj_pustą_populację()
    dopóki i > 0 wykonuj:
        j := wielkość_problemu
        genotyp := inicjuj_pusty_genotyp()
        dopóki j > 0 wykonuj:
            ścieżka := system_mrówkowy.krok_bez_zmiany_wartości_feromonów(tablica_feromonów)
            genotyp := genotyp.rozszerz(ścieżka.cel.id)
            tablica_feromonów := system_mrówkowy.odparuj_feromon_ze_ścieżki(tablica_feromonów, ścieżka.id)
            j := j-1
        populacja_genotypów.rozszerz(genotyp)
        i := i-1
    zwróć populacja_genotypów
```

Pseudokod 3.3: Idea działania transformacji przez parowanie feromonów.

Ostatni zaproponowany sposób postępowania przy mapowaniu reprezentacji ponownie bazuje na ograniczeniu wyboru połączeń do tylko niektórych z dostępnych. Tym

razem opiera się to jednak na ograniczeniu możliwości wyboru do kilku pozycji posiadających najbardziej intensywne wartości feromonów. W każdym kroku, dostępne połączenia są sortowane według atrakcyjności, a następnie pula, ograniczana jest do kilku pierwszych elementów z tak przygotowanego zbioru. Ponownie osiągane jest w ten sposób zróżnicowanie wybranych ścieżek w stosunku do tych, które byłyby wyłaniane w normalnym przebiegu działania systemu mrówkowego. Aby zapewnić odpowiedni sposób zróżnicowania pomiędzy wyborami w kolejnych krokach, również i tutaj następuje stopniowe wyparowywanie feromonów z wyselekcjonowanych połączeń. Unikane jest w ten sposób ograniczanie wyboru połączeń w poszczególnych etapach do tych samych pul. Opisany sposób postępowania przedstawiony został jako *Pseudokod 3.4*.

```
transformuj_reprezentację_przez_dobór_zachłanny(tablica_feromonów):
    i := pobierz_skonfigurowaną_wielkość_populacji()
    wielkość_problemu := pobierz_wielkość_problemu(tablica_feromonów)
    system_mrówkowy := inicjuj()
    populacja_genotypów = inicjuj_pustą_populację()
    dopóki i > 0 wykonuj:
        j := wielkość_problemu
        genotyp := inicjuj_pusty_genotyp()
        dopóki j > 0 wykonuj:
            tablica_feromonów_odfiltrowana := odfiltruj_ścieżki_o_niskich_wartościach_feromonów(tablica_feromonów)
            ścieżka := system_mrówkowy.krok_bez_zmiany_wartości_feromonów(tablica_feromonów_odfiltrowana)
            genotyp := genotyp.rozszerz(ścieżka.cel.id)
            tablica_feromonów := system_mrówkowy.odparuj_feromon_ze_ścieżki(tablica_feromonów, ścieżka.id)
            j := j-1
        populacja_genotypów.rozszerz(genotyp)
        i := i-1
    zwróć populacja_genotypów
```

Pseudokod 3.4: Idea działania transformacji przez dobór zachłanny.

3.2. Architektura programowa prototypowego rozwiązania

Przedstawiona koncepcja proponowanego rozwiązania została zaimplementowana w aplikacji napisanej w języku Python. Język ten został wybrany ze względu na możliwość szybkiego i sprawnego prototypowania wielu koncepcji, których rozważanie jest przedmiotem niniejszej pracy. Istotną częścią omawianej implementacji jest platforma PyAge, implementująca koncepcję zaproponowaną przez autorów [25]. Rozwiązanie to zostało wybrane jako jedna z podstaw dla implementacji omawianego systemu m.in. ze

względu na szerokie możliwości zastosowania w obliczeniach heurystycznych. Wykorzystanie go pozwoliło na sprawne sprawdzenie możliwości wykorzystania algorytmów ewolucyjnych w omawianej koncepcji.

Drugim istotnym filarem, który pomógł zbudować architekturę hybrydyzującą była platforma przeprowadzająca obliczenia w oparciu o algorytmy mrówkowe, zaproponowana i szczegółowo opisana przez autorów [42]. System ten skupia się wokół nowoczesnych technik działania systemów mrówkowych, jak rozwiązania socjo-kognitywne. Autorzy pracy skupiają się na porównaniu działania tego typu innowacyjnych podejść do obliczeń w algorytmach mrówkowych w stosunku do klasycznych rozwiązań. Platforma ta daje więc solidne podstawy do badań nad możliwościami efektywnego wykorzystania systemów optymalizacji opartych o kolonie mrówek.

Oba przedstawione rozwiązania również opierają się na implementacji w języku Python, co pozwoliło na ich sprawną integrację programową, a w efekcie umożliwiło autorowi skupić się na aspekcie koncepcyjnym rozwiązania, zamiast na problemach technicznych z tym związanych. Główną częścią implementacji prototypu było wprowadzenie części łączących obie wymienione platformy i pozwalających na ich swobodną współpracę. Podstawowym problemem, jaki pojawiał się w tym obszarze były znaczące różnice w sposobie reprezentacji problemu, który wykorzystywany był przez każdą z platform. Największą częścią prowadzonych prac było więc wprowadzenie koncepcji wzajemnego mapowania tych reprezentacji. W celach testowych zostały zaproponowane trzy niezależne metody takiego mapowania. Umożliwiło to wyłonienie koncepcji wprowadzającej najlepsze efekty z punktu widzenia poprawy wyników otrzymywanych z tradycyjnych (funkcjonujących samodzielnie) rozwiązań.

3.3. Platforma PyAge

Szczegółowe informacje dotyczące platformy obliczeniowej PyAge można znaleźć w publikacji opisującej jej koncepcję [25], a także w jej dokumentacji technicznej [1]. Warto jednak po krótce przybliżyć ideę, jaka przyświecała autorom oraz zwrócić uwagę, dlaczego daje ona dobre podstawy do wykorzystania w obliczeniach dotyczących problemów optymalizacji dyskretnej, a w szczególności w obliczeniach związanych z algorytmami ewolucyjnymi, które są jedną z podstaw proponowanego rozwiązania.

Jest to system opierający się na lekkich, reużywalnych, komponentach, które dają się w łatwy sposób zrównoleglać. Sprawia to, że rozwiązanie takie staje się bardzo dobrą

podstawą dla ewolucyjnych systemów wielo-agentowych. Autorzy zwracają również uwagę na możliwość zastosowania go w systemach wspierających podejmowanie decyzji. Implementacja opiera się na wykorzystaniu środowiska luźno powiązanych komponentów, co sprawia, że implementacja każdego z nich może być w łatwy sposób konfigurowana i podmieniana, bez istotnego wpływu na sposób działania pozostałych części systemu. Pozwala to na wykorzystanie algorytmów obliczeniowych, których dotyczy platforma do zastosowań związanych z rozwiązywaniem zróżnicowanych problemów wejściowych.

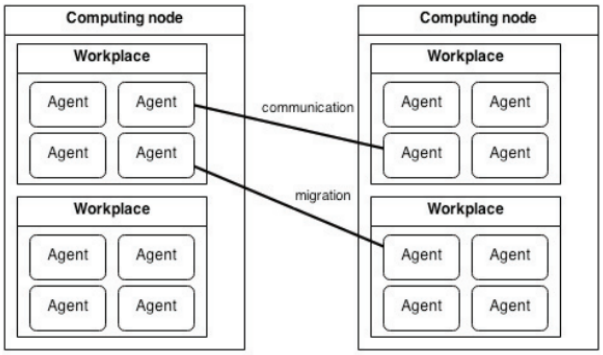
Dzięki oparciu poszczególnych komponentów na obiektach języka Python, definiujących określone funkcjonalności, wprowadzenie nowych komponentów odbywa się poprzez implementację podobnej klasy obiektu, z takim samym zestawem metod. Nie ma tutaj potrzeby implementacji konkretnego interfejsu, czy też dziedziczenia po klasach bazowych, gdyż zgodnie z założeniami języka Python, każdy obiekt określany jest poprzez zbiór swoich metod. Podejście takie pozawala uwolnić się od konieczności utrzymywania skomplikowanej hierarchii klas, jednocześnie wciąż pozwalając na wykorzystanie technik programowania obiektowego, jak np. polimorfizm.

Zastosowanie konkretnych implementacji komponentów definiuje się poprzez wykorzystanie stosownego pliku konfiguracji, wykorzystywanego przez system jako jeden z parametrów wejściowych podczas uruchamiania. Poprzez tą konfigurację i określane w niej komponenty, definiowane są ilości agentów/osobników, ich parametry, sposoby migracji, mutacji, krzyżowania, a także sposób kodowania problemu i reprezentacji go w postaci genotypu.

Symulacje uruchamiane w oparciu o platformę PyAge oparte są na reaktywnym przetwarzaniu na podstawie wywoływanych zdarzeń. W metodologii tej, operacje jakie zachodzą w trakcie działania, modelowane są (jak wskazuje nazwa) poprzez ciąg następujących po sobie zdarzeń. Podejście takie pozwala na znacznie łatwiejszą organizację kodu aplikacji, pozwalając na jego łatwiejszy rozwój i utrzymanie. Ponadto, systemy oparte o zdarzenia uznawane są za bardziej wydajne niż te, które przeprowadzają symulacje w sposób ciągły. Dużą zaletą jest w tym wypadku brak konieczności stosowania skomplikowanych mechanizmów synchronizacji wątków [15].

W konsekwencji zastosowania architektury wieloagentowej, podstawową jednostką, na której operuje platforma, jest *agent*. Każdy z nich operuje wewnątrz ściśle określonego środowiska - *workplace*. Środowiska te określają sposób działania agentów, poprzez wywoływanie ich poszczególnych metod i dostarczanych przez nich usług w odpowiedniej

kolejności. System jako całość, jest odpowiedzialny za tworzenie nowych agentów, kontrolowanie ich funkcjonowania i dostarczanie wymaganych, zależnych usług. Architektura takiego rozwiązania została przedstawiona na rysunku *Rysunku 3.2*.



Rysunek 3.2: Architektura agentowa platformy PyAge [26]

Jak wynika z diagramu, poszczególne agenty operują wewnątrz wspominanych środowisk (*workplace)*, które z kolei są ściśle związane z odseparowanymi od siebie węzłami (*computation node)*. Warto zwrócić uwagę, że agenty nie są sztywno przypisane do konkretnych środowisk - możliwa jest komunikacja pomiędzy wszystkimi agentami, a także ich migracja pomiędzy środowiskami, a nawet węzłami. Logika związana z obliczeniami, dotyczącymi konkretnego agenta definiowana jest w jego metodzie *step*. Każde ze środowisk wywołuje ją kolejno na wszystkich związanych z nim agentach.

Podstawą do uruchomienia własnej, niestandardowej konfiguracji, rozwiązującej konkretny problem jest wprowadzenie specyficznego dla niego sposobu kodowania. W przypadku omawianego problemu komiwojażera, reprezentacją taką jest genotyp opierający się na permutacji numerów miast, określającej kolejność ich odwiedzania oraz koszt pokonania takiej trasy (*fitness*).

Drugim elementem specyficznym dla konkretnego problemu jest zdefiniowanie zestawu tzw. *operatorów*. Zestaw taki składa się z czterech komponentów, odpowiedzialnych za: selekcję, krzyżowanie, mutację oraz ocenę. W opisywanej implementacji zastosowana została selekcja turniejowa. Opiera ona swoje działanie na wybraniu do kolejnej populacji osobników, posiadających genotypy o jak najlepszym wskaźniku funkcji przystosowania (*fitness*). Krzyżowanie odbywa się na zasadzie łączenia ze sobą genotypów dwóch osobników (permutacji numerów odwiedzanych miast) w taki sposób, aby w miarę możliwości zachowana była kolejność występowania ich w genotypach, z których pochodzą. Wprowadzona implementacja mutacji wykorzystuje zamiany losowych

par liczb miejscami w obrębie poszczególnych genotypów. Najbardziej istotnym elementem tego zbioru komponentów jest ewaluator. Dokonuje on oceny każdego z indywidualnych genotypów poprzez obliczenie kosztu przebycia reprezentowanej przez niego ścieżki. Wyznaczenie tego kosztu odbywa się z wykorzystaniem tablicy odległości, reprezentującej wagi krawędzi w omawianym wcześniej grafie. Określenie wag krawędzi wynika z odległości między punktami w układzie współrzędnych. Im mniejszy koszt przebycia reprezentowanej przez genotyp ścieżki, tym lepiej jest on oceniany przez ewaluator, a tym samym wzrasta prawdopodobieństwo jego wyboru jako źródła kolejnego pokolenia w algorytmie.

Przykładem jej interesującego zastosowania jest system symulujący zachowanie otwornic, zaproponowany przez autorów [26]. Projekt, którego wspomniana praca dotyczy, skupia się na dostarczeniu stosownych komponentów dla platformy PyAge w celu przeprowadzenia symulacji w środowisku agentowym. Autorzy zwracają uwagę na dużą elastyczność platformy w stosunku do innych implementacji środowisk agentowych (jak np. JADE [3]). Wspomniany system, dowodzi dużej elastyczności i uniwersalności zastosowań PyAge.

3.4. System mrówkowy

Prototypowa implementacja wykorzystana do przeprowadzenia pomiarów opiera się na algorytmie mrówkowym w podstawowej wersji, wykorzystywanym przy badaniach nad nowoczesnymi rozwiązaniami socjo-kognitywnymi [42]. Wprowadzona implementacja także i tego podsystemu wykorzystuje luźno powiązane komponenty i generyczne elementy, umożliwiając jej łatwe rozszerzanie i podmienianie implementacji poszczególnych części składowych. Taka architektura rozwiązania dała możliwość szybkiego wprowadzenia niestandardowych składników, niezbędnych do osiągnięcia założonego celu. Podsystem ten nastawiony jest ściśle na rozwiązywanie problemów, nad którymi można prowadzić obliczenia algorytmami mrówkowymi, jednocześnie nie nakładając ograniczeń związanych z konkretnymi problemami. Zastosowanie go do rozwiązywania problemu komiwojażera było związane z wprowadzeniem dedykowanych mu komponentów pojedynczej mrówki oraz całej kolonii. Reprezentacja problemu, na której operuje ten algorytm znacząco różni się od tej, którą wykorzystuje platforma PyAge w algorytmach ewolucyjnych. Zgodnie ze sposobem działania algorytmów mrówkowych,

mamy tutaj do czynienia z tablicą feromonów. Reprezentowana jest ona poprzez abstrakcję mapy ścieżek, po których poruszają się mrówki oraz intensywności feromonów z nimi związanych. Każda ścieżka reprezentuje krawędź grafu, a wagi poszczególnych krawędzi odpowiadają kosztom ich przebycia.

Na każdym etapie działania algorytmu, mrówka podejmuje decyzję o wyborze danej ścieżki na podstawie jej atrakcyjności. Konieczne jest więc obliczenie atrakcyjności wszystkich dostępnych z danego węzła ścieżek. Wybór konkretnej ścieżki w opisywanej implementacji uzależniony jest zarówno od wartości feromonu danej ścieżki, jak i odległości jej odpowiadającej. Prawdopodobieństwo wyboru ścieżki n_{ij} oraz odpowiadającej jej krawędzi e_{ij}, można więc zdefiniować poprzez następującą estymację *(12):*

$$n_{ij} = \frac{(\sum e_{ik,k\in V^{(i)}\setminus\{i\}} \text{cost}(e_{ik}))^{\alpha}}{\text{cost}(e_{ij})^{\beta}} \qquad (12)$$

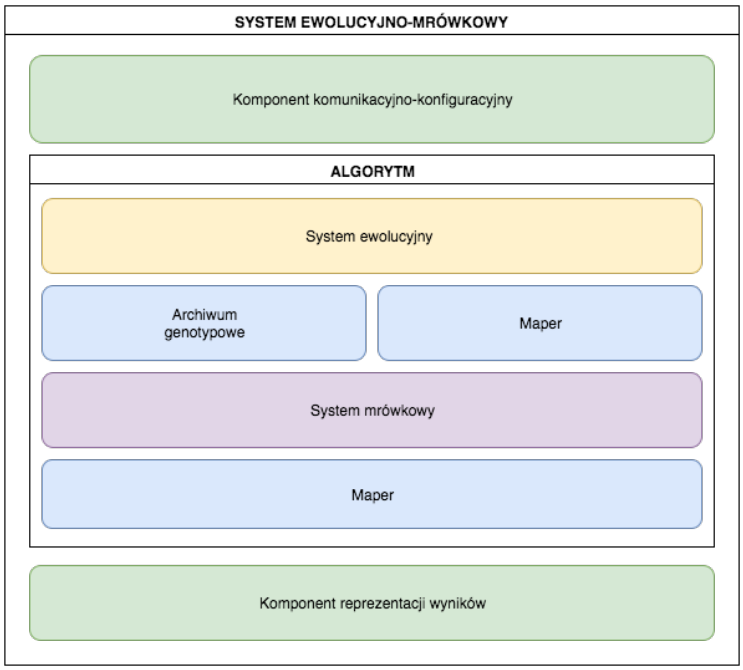
Implementacja uwzględnia domyślne wartości parametrów wpływu feromonów = 5.0 oraz odległości = 6.0. Zbiór wierzchołków $V(i)$ nie obejmuje tych, które zostały już odwiedzone, aby dojść do węzła i [42].

Warto podkreślić również, żę działanie implementacji algorytmu zostało oparte na wzorcu odwiedzający (*vistior*), co pozwoliło na odseparowanie logiki związanej z koncepcją algorytmu od obiektowej reprezentacji abstrakcji problemu.

3.5. Aspekty implementacyjne systemu

Implementacja systemu hybrydowego opiera się na wielu komponentach, które ściśle ze sobą współpracują i mogą być łatwo podmieniane. Szczegółowy obraz tego rozwiązania przedstawia *Rysunek 3.3*. Punktem wstępnym działania jest komponet komunikacyjno-konfiguracyjny. W prototypowej aplikacji wykorzystano do tego celu bibliotekę PyCLI [5]. Zastosowanie go pozwoliło na wprowadzenie przyjaznego dla użytkownika interfejsu linii komend, ułatwiającego definiowanie i poszukiwanie odpowiednich wartości dla wszystkich dostępnych stopni swobody aplikacji, jak również poprzez wywoływanie usług innych komponentów, uruchamianie odpowiednich rodzajów eksperymentów i obrazowanie ich rezultatów. Kolejnym istotnym elementem jest komponent obli-

czeniowy (*Algorytm*). Składa się on z kilku mniejszych ogniw, które związane są z dostarczanymi przez nie funkcjonalnościami. Jego głównym zadaniem jest przeprowadzanie eksperymentów według zadanej konfiguracji i dostarczanie ich wyników. Wśród modułów składowych komponentu obliczeniowego należy wymienić system ewolucyjny, system mrówkowy, a także elementy pozwalające na hybrydyzację ich działania. Zgodnie z przedstawioną wcześniej koncepcją rozwiązania, proces obliczeń składa się z kilku następujących po sobie faz, spośród których każda wykorzystuje inne elementy tego komponentu.



Rysunek 3.3: Schemat komponentów składających się na implementację. Opracowanie własne.

Pierwszą z nich jest uruchomienie obliczeń na algorytmie ewolucyjnym w oparciu o platformę PyAge. Długość trwania obliczeń w tej fazie określana jest przez warunek stopu związany z ilością wykonywanych kroków. Kolejna faza to przekształcenie wyników otrzymanych po zadanej liczbie epok obliczeniowych algorytmu ewolucyjnego na reprezentację algorytmu mrówkowego. Istotnym czynnikiem jest tutaj konieczność zachowania otrzymanej różnorodności i liczności rozwiązań. Jednocześnie wysokim priorytetem jest wykorzystanie zbioru genotypów wytworzonych podczas działania algorytmu ewolucyjnego, które będą w stanie zapewnić jak najlepsze wyniki. Dzięki takiemu podejściu, możliwe jest otrzymanie wysokiej jakości podstaw do wykorzystania z dalszymi obliczeniami w kolejnej fazie, przy zastosowaniu algorytmu mrówkowego. Aby

osiągnąć ten cel, implementacja rozwiązania hybrydowego została wzbogacona o komponent stanowiący pewien rodzaj bufora genotypów (*Archiwum genotypowe*). Rozwiązanie to zostało wprowadzone z uwagi na możliwość skupienia się różnorodności genotypów w bardzo wąskim zakresie, który niekoniecznie byłby w stanie dostarczyć wystarczająco dobrego rozwiązania problemu. Wprowadzenie tego komponentu, umożliwiło więc łatwe zachowanie zarówno różnorodności, jak i wysokiej jakości genotypów w zbiorze bazowym do przemapowania. Jego działanie opiera się na gromadzeniu zadanej ilości genotypów w trakcie działania algorytmu ewolucyjnego. Zgromadzone genotypy służą następnie jako podstawa do wspomnianego wcześniej procesu mapowania. W celu zachowania wszystkich wymaganych cech zbioru genotypów, decyzja o dodaniu nowego elementu podejmowana jest na podstawie jakości reprezentowanego przez niego rozwiązania, a także jego różnorodności na tle permutacji zgromadzonych wcześniej. Kryterium jakości osiągane jest tutaj dzięki przechowywaniu struktury posortowanej, która pozwala łatwo określić, czy dana wartość funkcji przystosowania może być lepsza od najgorszej z dotychczas przechowywanych. Ponadto ocenie poddawane jest kryterium różnorodności. W tym przypadku, klasyfikacja odbywa się z wykorzystaniem metryki Hamminga. Dzięki temu możliwe jest łatwe stwierdzenie jak duże zróżnicowanie permutacji występuje między badanym genotypem, a zbuforowanym zbiorem. W momencie gdy proponowany genotyp okaże się mieć wystarczająco dobrą wartość funkcji przystosowania oraz spełniać warunki różnorodności, ze zgromadzonego zbioru zostaje usunięty genotyp najniższej jakości, a na jego miejsce wstawiany jest badany kandydat, po czym następuje ponowne przesortowanie struktury, tak aby możliwe było sprawne sklasyfikowanie kolejnej badanej permutacji. Istotnym jest, aby oba warunki były spełnione. Nawet jeśli genotyp spełni warunek jakościowy, ale nie będzie zapewniał wystarczającej różnorodności, nie zostanie on wzięty pod uwagę. Warto zauważyć także, że w celu zapewnienia odpowiedniej efektywności działania systemu, gromadzenie genotypów jest wstrzymywane jeśli przez odpowiednio długi czas nie ma możliwości dokonania podmiany w zbuforowanym zbiorze. Sytuacja taka jednocześnie wskazuje na skupienie się genotypów w populacji wokół wąskiej grupy rozwiązań. Na podstawie zgromadzonych w takim archiwum genotypów możliwe jest przeprowadzenie zmiany reprezentacji informacji zgrodzanych w trakcie działa algorytmu ewolucyjnego na odpowiednią dla algorytmu mrówkowego, zgodnie z propozycją przedstawioną w początkowej części tego rodziału. Za proces ten odpowiada komponent *Mapera* z *Rysunku 3.3*. Następna faza obliczeń opiera się na uruchomieniu algorytmu mrówkowego z warunkiem stopu ponownie opierającym

się na zadanej ilości kroków. Wynik otrzymywany z tej fazy działania może zostać wykorzystany w dwojaki sposób. Jeśli globalny warunek stopu całego systemu, na to pozwala, reprezentacja tablicy feromonowej zostanie przekształcona ponownie na kodowanie do postaci populacji genotypów i nastąpi powrót do pierwszej fazy. W przeciwnym przypadku, zwracany jest końcowy wynik. Metody mapowania tablicy feromonowej na reprezentację genotypową również są jednym z przedmiotów badań niniejszej pracy, które szczegółowo przedstawione zostały w kolejnym rozdziale. Za działania z tym związane ponownie odpowiedzialny jest komponent *Mapera* (*Rysunek 3.3*).

Ostatni element związany z implementacją stanowi moduł wprowadzający możliwość czytelnego i jednoznacznego sposobu interpretacji otrzymywanych wyników (*Komponent reprezentacji wyników*). Komponent ten generuje pliki ze stosownymi wykresami na podstawie zadanych danych wynikowych, w zależności od przeprowadzanego eksperymentu, w oparciu o rezultaty uzyskane w trakcie działania *Algorytmu*. W prototypowej implementacji został on oparty na bibliotece Matplotlib [4] dla języka Python, która udostępnia wysokopoziomowe API programowe, dające szerokie możliwości na polu dostosowywania graficznej reprezentacji generowanych wykresów.

4. Badania eksperymentalne

4.1. Metodyka

Niniejszy rozdział przedstawia szereg eksperymentów, jakie zostały podjęte w celu sprawdzenia efektów działania proponowanego rozwiązania. W szczególności, ich celem było wykazanie, że możliwe jest uzyskanie lepszych wyników końcowych dla znanych problemów obliczeniowych, niż w przypadku zastosowania tradycyjnych heurystyk, jak algorytm ewolucyjny czy mrówkowy. Wśród eksperymentów, jakie zostały podjęte, wymienić należy:

- sprawdzenie działania wszystkich zaproponowanych metod mapowania sposobu reprezentacji problemu,
- weryfikację najlepszej z metod mapowania sposobu reprezentacji problemu, dla kilku wybranych, różnych zadań obliczeniowych
- porównanie działania z innymi, znanymi heurystykami, przy zastosowaniu najelpszej z metod mapowania sposoby reprezentacji problemu
- pomiar różnorodności badanych rozwiązań dla proponowanego prototypu implementacji

Jako referencyjny problem, wybrany został opisany szczegółowo w rodziale *2.1.2.* problem komiwojażera. Rozdział ten szczegółowo opisuje również, dlaczego problem problem komiwojażera może być uznawany za reprezentatywny przykład problemu optymalizacyjnego. W celu możliwości łatwej weryfikacji otrzymywanych wyników, do przeprowadzenia eksperymentów wybrany został benchmark TSPLib [50]. Obejmuje on szereg różnej wielkości zadań obliczeniowych, stanowiących przykłady problemu komiwojażera. Znane są również optymalne rozwiązania dla zadań, które są tam zaproponowane. Z tego powodu, stanowi on bardzo dobry zbiór referencyjny, który w łatwy sposób pozwala na ocenę jakości rozwiązania. Dla każdego z eksperymentów, obliczenia przeprowadzone zostały z 10-krotnym powtórzeniem, a następnie uśrednione, co pozwoliło na wykazanie powtarzalności otrzymywanych rezultatów. Wyniki przedstawione na wykresach pozwalają lepiej zaobserwować tą właściwość. Aby możliwe było pełne przepro-

wadzenie obliczeń, konieczne było wyspecyfikowanie szeregu stopni swobody, związanych z działaniem obu algorytmów składowych (ewolucyjnego oraz mrówkowego). *Tabela 4.1* przedstawia informacje na temat dobranych wartości.

Algorytm ewolucyjny	
Współczynnik mutacji	3.0
Ilość iteracji w pojedynczym kroku hybrydy	1000
Algorytm mrówkowy	
Współczynik wpływu feromonu na wybór ścieżki Alpha	5.0
Wsppółczynnik wpływu heurystyki na wybór ścieżki Beta	6.0
Współczynnik zanikania Rho	0.8
Współczynnik pozostawiania feromonu Q	6.0
Współczynnik parowania fermonu	0.9
Ilość iteracji w pojedynczym kroku hybrydy	20

Tabela 4.1: Dobrane wartości stopni swobody dla propozycji algorytmu hybrydowego. Opracowanie własne.

Dobór konkretnych wartości dla przedstawionych współczynników wynika z doświadczalnej weryfikacji działania algorytmów składowych oraz wyboru wartości, które pozwalają na otrzymywanie przez nie możliwie najlepszych rezultatów.

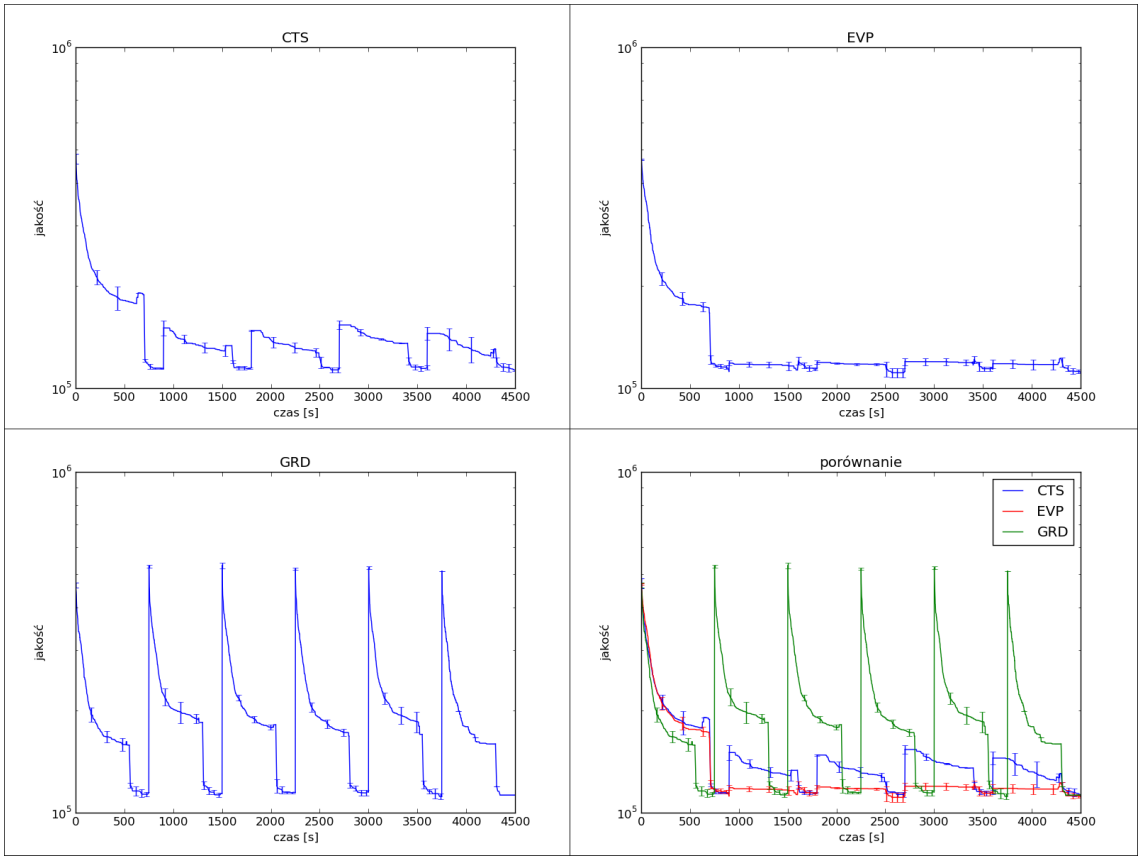
4.2. Porównanie zaproponowanych metod mapowania

Pierwsza grupa przeprowadzonych eksperymentów obejmuje weryfikację opisanych w rodziale *3.1.* trzech sposobów mapowania pomiędzy reprezentacjami problemu w poszczególnych algorytmach składowych. Dla ułatwienia wizualizacji rezultatów, przyjęte zostały następujące oznaczenia: mapowanie w oparciu o rzut monetą – *CTS*, mapowanie z wykorzystaniem parowania – *EVP* oraz mapowanie w oparciu o dobór zachłanny – *GRD*. Jako problem referencyjny, wybrany został zbiór *pr76* z biblioteki TSPLib [50]. Jako warunek końca, wybrany ustawiono limit czasu *4500s*. *Tabela 4.2* zawiera końcowe wyniki jakości rozwiązania dla wykorzystania każdego z opisanych sposobów mapowania, natomiast *Rysunek 4.1* obrazuje zachowanie systemu w czasie. Jako jakość rozwiązania, przyjmowany był koszt przebycia permutacji proponowanej przez

system w danym momencie czasowym (długość ścieżki). Wynikiem końcowym jest najniższy koszt zaproponowany w całym czasie działania. Otrzymywane rezultaty były powtarzalne.

Algorytm	Wynik
Referencyjny	108159.00
CTS	114187.00
EVP	112794.00
GRD	112990.00

Tabela 4.2: Rezultaty końcowe jakości rozwiązania dla trzech sposobów mapowania reprezentacji problemu. Opracowanie własne.



Rysunek 4.1: Porównanie działania w czasie zaproponowanych sposobów mapowania reprezentacji problemu. Opracowanie własne.

Można dostrzec, iż metoda *EVP* umożliwia zachowanie największej ilości wypracowanych wcześniej informacji w stosunku do pozostałych – przy jej zastosowaniu, występuje

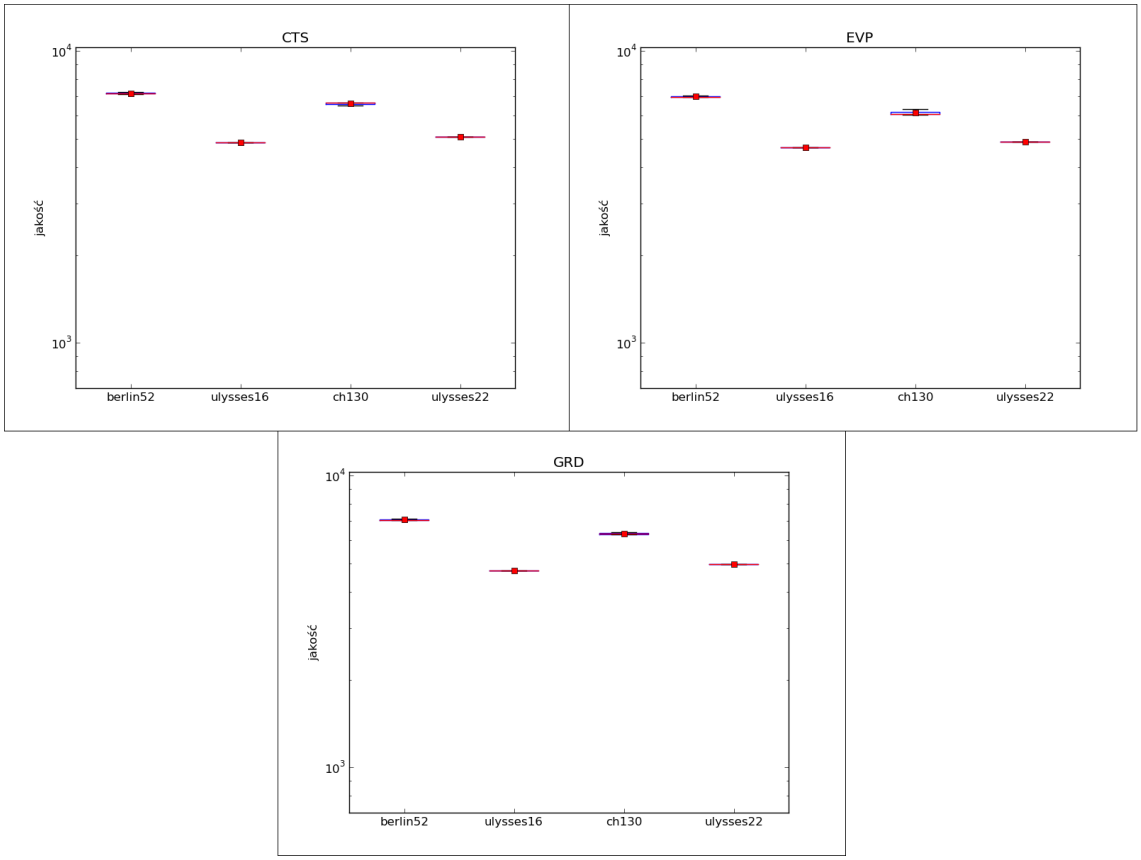
bowiem najmniejsza utrata jakości wyniku, w stosunku do pozostałych. Otrzymane rezultaty wraz z obserwacją dotyczącą zachowywania ilość informacji pozwalają stwierdzić, że technika ta może zostać wykorzystana w najbardziej efektywny sposób. Każda z badanych metod umożliwia otrzymywanie ostatecznych wyników zbliżonych do referencyjnego.

4.3. Porównanie efektów działania dla różnych problemów

Kolejny eksperyment obejmuje porównanie działania poszczególnych metod mapowania reprezentacji przy wykorzystaniu różnych problemów wejściowych. Jego celem jest przede wszystkim sprawdzenie działania proponowanego rozwiązania hybrydowego w działaniu z różnymi problemami wejściowymi. Na tej podstawie wybrana zostanie również metoda mapowania, która posłuży za wybraną metodę mapowania dla dalszych weryfikacji. Podobnie jak przy poprzednim doświadczeniu, również i tutaj przyjęta została następująco nomenklatura: mapowanie w oparciu o rzut monetą – *CTS*, mapowanie z wykorzystaniem parowania – *EVP* oraz mapowanie w oparciu o dobór zachłanny – *GRD*. Rozważane są następujące zbiory wejściowe z biblioteki TSPLib [50]: *ulysses16, ulysses22, berlin52, ch130*. Jako warunek końca, wybrany został limit czasu *10000s*. *Tabela 4.3* zawiera końcowe wyniki jakości rozwiązania dla wykorzystania każdego z opisanych sposobów mapowania, natomiast *Rysunek 4.2* obrazuje zachowanie systemu w czasie. Podobnie jak w przypadku poprzedniego eksperymentu, jako rezultat końcowy, uznawany jest najniższy koszt związany ze ścieżkami proponowanymi przez system w trakcie jego działania.

Algorytm	ulysses16	ulysses22	berlin52	ch130
Referencyjny	6859.00	7013.00	7542.00	6110.00
CTS	6859.00	7073.52	7731.45	6410.85
EVP	6859.00	7056.20	7651.20	6245.80
GRD	6859.00	7062.83	7680.60	6275.00

Tabela 4.3: Rezultaty końcowe jakości rozwiązania dla trzech sposobów mapowania reprezentacji problemu. Porównanie dla kilku różnych problemów. Opracowanie własne.



Rysunek 4.2: Porównanie działania w czasie zaproponowanych sposobów mapowania reprezentacji problemu. Opracowanie własne.

Wyniki przeprowadzonych doświadczeń potwierdzają wnioski wysunięte dzięki poprzedniemu eksperymentowi – osiągnięcie najlepszych rezultatów możliwe jest w oparciu o wykorzystanie metody mapowania *EVP*. Metoda ta posłuży więc do przeprowadzenia kolejnych eksperymentów z wykorzystaniem proponowanej techniki hybrydowej. Niezależnie od wybranego problemu wejściowego, możliwe jest także osiągnięcie wyników jakościowych zbliżonych do algorytmu referencyjnego.

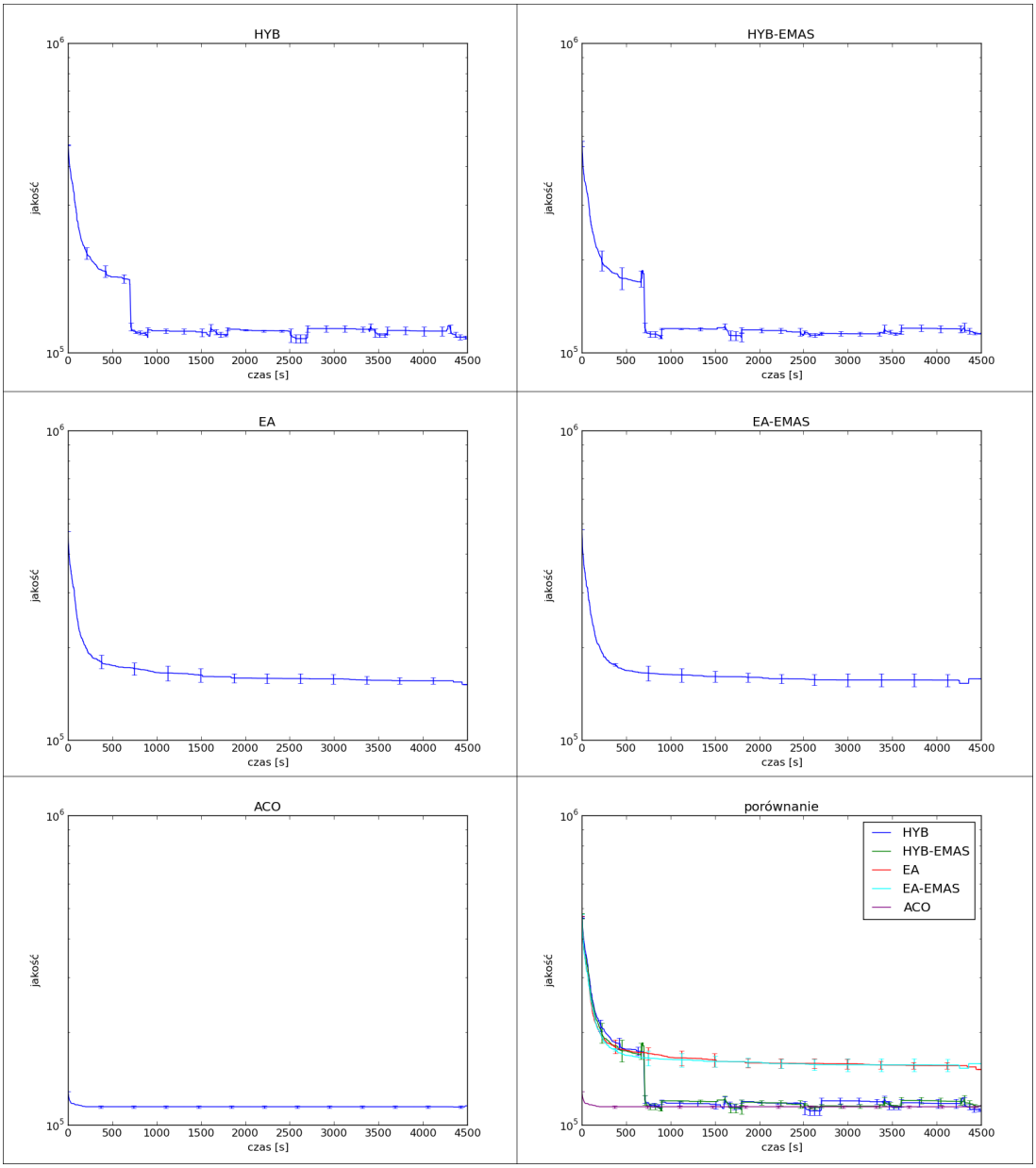
4.4. Porównanie z innymi heurystykami

W celu uzyskania pełnego obrazu na temat efektów działania algorytmu hybrydowego, przeprowadzony został również test porównawczy, obrazujący zestawienie wyników działania proponowanego rozwiązania z innymi znanymi metaheurystykami. Do porównania wybrane zostały algorytmy: ewolucyjny - *EA*, mrówkowy – *ACO*, hybrydowy - *HYB* (w oparciu o wybraną wcześniej technikę mapowania reprezentacji *EVP*), a także dodatkowo wykorzystano implementację algorytmu ewolucyjnego opartą o technikę

EMAS [2][48] – *EA-EMAS*. Implementacja ta została także wykorzystana do porównania jako składowy element algorytmu hybrydowego *HYB-EMAS*. Jako problem referencyjny, ponownie wybrany został zbiór *pr76* z biblioteki TSPLib [50]. Jako warunek końca, ustawiono limit czasu *4500s*. *Tabela 4.4* zawiera końcowe wyniki jakości rozwiązania dla wykorzystania każdej z opisywanych metaheurystyk, natomiast *Rysunek 4.3* obrazuje zachowanie systemu w czasie. Zgodnie z konwencją przyjętą we wcześniejszych badaniach, również i tym razem jako jakość rozwiązania, przyjmowany był koszt przebycia permutacji proponowanej przez system w danym momencie czasowym (długość ścieżki). Wynikiem końcowym jest najniższy koszt zaproponowany w całym czasie działania. Otrzymywane rezultaty były powtarzalne.

Algorytm	Wynik
Referencyjny	108159.00
HYB	112794.00
HYB-EMAS	112586.50
EA	148695.43
EA-EMAS	147149.71
ACO	114223.23

Tabela 4.4: Rezultaty końcowe dla obliczeń z wykorzystaniem różnych metaheurystyk. Opracowanie własne.



Rysunek 4.3: Porównanie działania w czasie kilku metaheurystyk. Opracowanie własne.

Uzyskane rezultaty, pozwalają stwierdzić, że proponowana technika hybrydowa posiada pewną przewagę w jakości finalnego rozwiązania nad pozostałymi algorytmami. Nie jest jednak możliwe uzyskanie za jej pomocą rezultatów w czasie krótszym, niż ma to miejsce w przypadku pozostałych algorytmów. Wykorzystanie techniki EMAS [2][48] dostarcza rozwiązań nieznacznie lepszych od standardowej implementacji zarówno w przypadku algorytmu ewolucyjnego, jak i propozycji hybrydowej. Wszystkie algorytmy poddane testowi, pozwalają uzyskać końcowe wyniki bliskie algorytmowi referencyjnemu (za wyjątkiem algorytmu genetycznego, którego rezultaty działania nieznacznie odbiegają od pozosotałych).

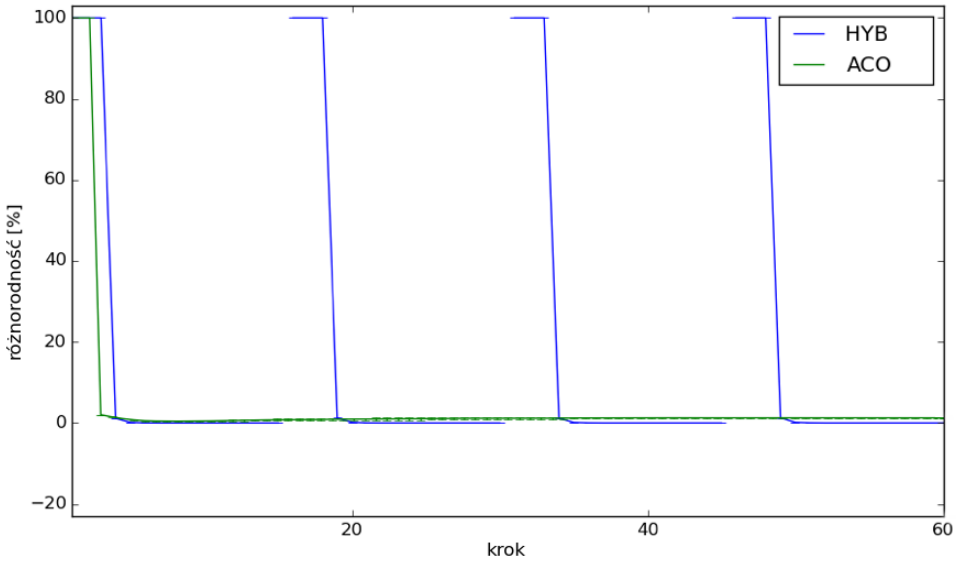
4.5. Pomiar różnorodności

Jako dodatkowe doświadczenie, przeprowadzone zostało także sprawdzenie poziomu różnornodności eksplorowanych rozwiązań. Przez pojęcie różnorodności, rozumiana jest tutaj powtarzalność permutacji rozważanych w trakcie przeszukiwania. Pomiar różnorodności w przestrzeni dyskretnej jest oczywiście zadaniem trudnym – wskazują na to badacze w [52], określając poszukiwanie najbliższego podłańcucha (jak np. kodowanie możliwych rozwiązań TSP) jako problem NP-trudny. Ciekawą propozycję takiego badania dla algorytmu mrówkowego przedstawiają jednak autorzy [51], opierając się na zawartości tablicy feromonów. W przypadku tym, wysoce istotne jest, aby przez cały czas jego działania, zachowywany był wysoki poziom różnorodności, tak aby uniknąć nadmiernego skupienia eksploracji wokół lokalnego optimum i utracenia możliwości sprawdzenia innych możliwości w ramach dostępnych zasobów (np. czasowych). W odniesieniu do charakterystyki działania tej techniki, autorzy proponują określenie różnorodności poprzez badanie faktu pozostawienia feromonów przez mrówki na poszczególnych ścieżkach, bądź jego braku. Wskazują oni, że wiążąc poszczególne ścieżki z możliwymi rozwiązaniami oraz obecność na nich feromonu z intensywnością eksploracji danego rozwiązania (ścieżki), możliwe jest określenie, jak duży jest poziom skupienia przeszukiwania w lokalnych optimach. Na tej podstawie, proponowany jest więc pomiar tej wielkości poprzez obliczenie stosunku ilości ścieżek, na których znajduje się naniesiony feromon, do ilości tych, które tej właściwości nie posiadają – im większa jest wartość tej miary, z tym większą różnorodnością eksploracji mamy do czynienia. Metryka ta jest wielkością procentową i może być zdefiniowana poprzez następujący model *(13):*

$$100\% \cdot \frac{\#\{e_{ij} : \pi_{ij} > 0\}}{\#\{e_{ij}\}}, \forall i, j \qquad (13)$$

Gdzie e_{ij} to ścieżka łącząca miasta i, j, a π_{ij} określa ilość feromonu na tej ścieżce. W przypadku proponowanego w tej pracy rozwiązania, pomiar wykonywany jest w trakcie działania jego składowej związanej z algorytmem mrówkowym. Do uzyskania wyników ponownie wykorzystano zbiór referencyjny *pr76* biblioteki TSPLib [50]. Spraw-

dzony został poziom różnorodności dla 60 kroków wykonanych przez składową algorytmu mrówkowego. Rezultaty porównywane są z algorytmem mrówkowym uruchamianym w sposób niezależny. Wyniki przedstawia *Rysunek 4.4*.



Rysunek 4.4: Rezultat pomiaru różnorodności rozwiązań dla proponowego koncepcyjnego rozwiązania. Opracowanie własne.

Jak można zaobserwować, w początkowej fazie poziom różnorodności jest bardzo wysoki (zaraz po wykonaniu mapowania reprezentacji z algorytmu ewolucyjnego), a wraz z czasem trwania obliczeń, maleje (co jest skutkiem zbliżania się do rozwiązania końcowego). Co więcej, proces ten ma miejsce w każdej iteracji algorytmu hybrydowego jako całości, a więc możliwe jest stwierdzenie, że zastosowanie hybrydyzacji i dodatkowe procesowanie w poszczególnych iteracjach przez algorytm ewolucyjny, pozwalają na odbudowę mierzonej różnorności, podczas gdy dla algorytmu mrówkowego działającego niezależnie, zmniejsza się ona wraz z czasem działania (wykonywaniem kolejnych kroków).

5. Podsumowanie i kierunki dalszego rozwoju

Podsumowanie

W ramach przeprowadzonych prac, przedstawiony został problem związany z dyskretnymi problemami optymalizacyjnymi, a także szeroki przegląd metod związanych z ich rozwiązywaniem. Przeprowadzony został szeroki i szczegółowy przegląd dotychczasowych osiągnięć współczesnej nauki na tym polu. Najbardziej istotną częścią tego dokumentu jest jednak przedstawiona koncepcja rozwiązania hybrydowego, która łączy ze sobą znane dotąd i dobrze zbadane algorytmy ewolucyjne i mrówkowe. Opracowany został system, który pozwala sprawdzić, jak odpowiednie połączenie tego typu metaheurystyk może wpłynąć na otrzymywane wyniki. Prace nad prototypowym rozwiązaniem obejmowały przede wszystkim implementacje trzech niezależnych metod mapowania reprezentacji problemu pomiędzy tymi dwoma algorytmami – zbiorem genotypów oraz tablicą feromonów. Wykonany został szereg doświadczeń i eksperymentów, które pozwoliły na wyciągnięcie wniosków, na temat opisywanej koncepcji. Na podstawie otrzymanych wyników, można stwierdzić, że proponowany system pozwala na poprawę rezultatów, które osiągane są z wykorzystaniem w sposób pojedynczy, używanych technik składowych. Porównanie wyników do algorytmu referencyjnego [50] pozwala na stwierdzenie, że choć ich jakość jest lepsza niż w przypadku algorytmów składowych, to wciąż możliwe jest osiągniecie jeszcze lepszych rezultatów. Zauważyć należy także, że zastosowanie hybrydyzacji zwiększa jednak czas oczekiwania na wyniki z pierwszej iteracji algorytmu. Należy także zaznaczyć, żę dzięki zastosowaniu takiego podejścia, możliwe jest osiągnięcie pozytywnego wpływu na poziom różnorodności (zdefiniowanej w rozdziale 4.5) przetwarzanych potencjalnych rozwiązań w przestrzeni. Ogólny rezultat przeprowadzonych prac należy uznać za wskazujący na wysoki potencjał proponowanej koncepcji i dający możliwość prowadzenia dodatkowych badań w tym obszarze.

Kierunki dalszego rozwoju

Jak zostało to wspomniane we wcześniejszym podrozdziale, choć udało się osiągnąć wyniki zbliżone do otrzymywanych z algorytmu referencyjnego, to mimo wszystko wciąż istnieje możliwość prowadzenia dalszych badań w obszarze rozwoju hybrydowych technik metaheurystycznych. Kierunkami optymalizacji, nad jakimi można w dalszym ciągu pracować są przede wszystkim szukanie rozwiązań, które pozwolą na przyśpieszenie w otrzymywaniu rezultatów, a także dalsze zwiększanie ich jakości. Duże pole możliwości w tym obszarze dostarczają współczesne architektury równoległe i rozproszone. W szczególności, istotnego przyspieszenia może dostarczyć próba implementacji opisywanej koncepcji z wykorzystaniem architektury GPGPU, która wprowadzi wysokie zrównoleglenie prowadzonych obliczeń. Należy jednak pamiętać, że może to wymagać pozyskania całkowicie odrębnej implementacji prototypu, ściśle dostosowanej pod kątem obliczeń na tego typu maszynach. Dużą rolę może odegrać także poszukiwanie i sprawdzanie innych metod mapowania reprezentacji problemu, jakie wykorzystywane są przez algorytmy składowe. Może to wprowadzić istotne zmiany jakościowe w otrzymywanych rezultatach.

Spis rysunków i tabel

Bibliografia

[1] Biblioteka PyAge - dokumentacja techniczna w repozytorium GitHub. https://github.com/maciek123/pyage/wiki. (dostęp 20.05.2016).

[2] Evolutionary Multi-Agent System. https://www.age.agh.edu.pl/agent-based-computing/emas-2. (dostęp 20.05.2016).

[3] Oficjalna stron frameworka języka Java - JADE. http://jade.tilab.com. (dostęp 20.05.2016).

[4] Oficjalna strona internetowa biblioteki języka Python – Matplotlib. http: //matplotlib.org. (dostęp 20.05.2016).

[5] Oficjalna strona internetowa biblioteki języka Python – PyCLI. https:// pythonhosted.org/pyCLI. (dostęp 20.05.2016).

[6] Sarab AlMuhaideb and Mohamed El Bachir Menai. Hcolonies: a new hybrid metaheuristic for medical data classification. Applied Intelligence, 41(1):282–298, 2014.

[7] James Baldwin. A new factor in evolution. American Naturalist, 30:441–451, 1896.

[8] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. J. ACM, 9(1):61–63, January 1962.

[9] Una Benlic and Jin-Kao Hao. Hybrid Metaheuristics, chapter Hybrid Metaheuristics for the Graph Partitioning Problem, 157–185. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[10] P. L. Hammer, E. L. Johnson, B. H. Korte, (Eds.): Annals of Discrete Mathematics 5 - Discrete Optimization II, Amsterdam et al. 1979, 113-138.

[11] Richard W. Burkhardt. Lamarck, evolution, and the inheritance of acquired characters. Genetics, 194(4):793–805, 2013.

[12] Aleksander Byrski. Agent-based metaheuristics in search and optimisation. Wydawnictwa AGH, 2013.

[13] G. E. Carrillo-Ureta, P. D. Roberts, and V. M. Becerra. Genetic algorithms for optimal control of beer fermentation. In Intelligent Control, 2001. (ISIC '01). 2001 IEEE International Symposium, 391–396, 2001.

[14] Krzysztof Cetnarowicz. Problems of the evolutionary development of the multi-agent world. Proceedings of the First International Workshop: Decentralized Multi-Agent Systems DIMAS'95, 113–123, 1995.

[15] Byoung Choi. Modeling and simulation of discrete-event systems. John Wiley & Sons Inc, Hoboken, New Jersey, 2013.

[16] Richard Dawkins. The selfish gene. Oxford University Press, Oxford New York, 1989.

[17] Marco Dorigo. Ant colony optimization. MIT Press, Cambridge, Mass, 2004.

[18] Marco Dorigo. Ant colony optimization and swarm intelligence : 4th international workshop, ANTS 2004, Brussels, Belgium, September 5-8, 2004 : proceedings. Springer, Berlin New York, 2004.

[19] Marco Dorigo and Gianni Di Caro. New ideas in optimization. chapter The Ant Colony Optimization Meta-heuristic, 11–32. McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

[20] Matthias Ehrgott and Xavier Gandibleux. Hybrid Metaheuristics: An Emerging Approach to Optimization, chapter Hybrid Metaheuristics for Multi-objective Combinatorial Optimization, 221–259. Springer Ber- lin Heidelberg, Berlin, Heidelberg, 2008.

[21] Ewa Figielska. Algorytmy ewolucyjne i ich zastosowania. Zeszyt naukowy nr 1, 81–92, 2006.

[22] David Fogel. Evolutionary computation : the fossil record. IEEE Press, New York, 1998.

[23] David Goldberg. Algorytmy genetyczne i ich zastosowania. Wydawnictwa Naukowo-Techniczne, Warszawa, 1998.

[24] Akira Hara, Syuhei Matsushima, Takumi Ichimura, and Tetsuyuki Takahama. Ant colony optimization using exploratory ants for constructing partial solutions. In IEEE Congress on Evolutionary Computation, 1–7, July 2010. 2

[25] Maciej Kaziród, Wojciech Korczyński, and Aleksander Byrski. Agent- oriented computing platform in Python. In Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014 IEEE/WIC/ACM Internatio- nal Joint Conferences on, volume 3, 365–372, Aug 2014.

[26] Maciej Kaziród, Wojciech Korczyński, Elias Fernández, Aleksander Byrski, Marek Kisiel-Dorohinicki, Paweł Topa, Jarosław Tyszka, and Maciej Komo- siński. Agent-oriented foraminifera habitat simulation. Procedia

Computer Science, 51:1062 – 1071, 2015. International Conference On Computational Science, {ICCS} 2015 Computational Science at the Gates of Nature.

[27] Michał Kowol, Aleksander Byrski, and Marek Kisiel-Dorohinicki. Agent-based evolutionary computing for difficult discrete problems. Procedia Computer Science, 29:1039 – 1047, 2014. 2014 International Conference on Computational Science.

[28] John Koza. Genetic programming : on the programming of computers by means of natural selection. MIT Press, Cambridge, Mass, 1992.

[29] Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. European Journal of Operational Research, 59(2):231 – 247, 1992.

[30] Guan-Chun Luh and Chun-Yi Lin. Structural topology optimization using ant colony optimization algorithm. Applied Soft Computing, 9(4):1343 – 1353, 2009.

[31] Sean Luke. Essentials of metaheuristics. https://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf, 2013. (dostęp 20.05.2016).

[32] Pablo Moscato. A gentle introduction to memetic algorithms. In Handbook of Metaheuristics, 105–144. Kluwer Academic Publishers, 2003.

[33] Quang Huy e, Yew Soon Ong, and Meng Hiot Lim. Non-genetic transmission of memes by diffusion. In Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO '08, 1017–1024, New York, NY, USA, 2008. ACM.

[34] R. Gary Parker, Ronald L. Rardin. Discrete optimization. Computer science and scientific computing. Academic Press, 1988.

[35] Michael Norman, Pablo Moscato, and La Plata. A competitive-cooperative approach to complex combinatorial search, 1991.

[36] V. Oduguwa, A. Tiwari oraz R. Roy. Evolutionary computing in manufacturing industry: an overview of recent applications. Applied Soft Computing, 5(3):281 – 299, 2005. Application Reviews.

[37] Judea Pearl. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[38] Yiannis G. Petalas, Konstantinos E. Parsopoulos, and Michael N. Vrahatis. Memetic particle swarm optimization. Annals OR, 156:99–127, 2007. 3

[39] Carlos Andrés Peña-Reyes and Moshe Sipper. Evolutionary computation in medicine: an overview. Artificial Intelligence in Medicine, 19(1):1 – 23, 2000. Evolutionary Computation in Medicine.

[40] Vreda Pieterse and Paul E. Black. Algorithms and theory of computation handbook. http://www.nist.gov/dads/HTML/optimization.html, 1999. (dostęp 20.05.2016).

[41] Mauricio G.C. Resende and Renato F. Werneck. A hybrid heuristic for the p-median problem. Journal of Heuristics, 10(1):59–88, 2004.

[42] Mateusz Sękara, Michał Kowalski, Aleksander Byrski, Bipin Indurkhya, Marek Kisiel-Dorohinicki, Dana Samson, and Tom Lenaerts. Multi- pheromone ant colony optimization for socio-cognitive simulation purposes. Procedia Computer Science, 51:954 – 963, 2015. International Conference On Computational Science, {ICCS} 2015 Computational Science at the Ga- tes of Nature.

[43] Talbi. Metaheuristics from design to implementation. John Wiley & Sons, Hoboken, N.J, 2009.

[44] E.-G. Talbi. A taxonomy of hybrid metaheuristics. Journal of Heuristics, 8(5):541–564, 2002.

[45] Reiko Tanese. Parallel genetic algorithms for a hypercube. In Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application, 177–183, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.

[46] Sui-Man Tse, Yong Liang, Kwong-Sak Leung, Kin-Hong Lee, and Shu-Kam Tony Mok. Multi-drug cancer chemotherapy scheduling by a new memetic optimization algorithm. In 2005 IEEE Congress on Evolutionary Computation, volume 1, 699–706 Vol.1, 2005.

[47] Jerzy Wałaszek. Algorytmy i struktury danych - problem komiwojażera. http://eduinf.waw.pl/inf/alg/001_search/0140.php. (dostęp 20.05.2016).

[48] Aleksander Byrski, Rafał Dreżewski, Leszek Siwik, and Marek Kisiel Dorohinicki.Evolutionary multi-agent systems. The Knowledge Engineering Review, 30:171–186, 3 2015.

[49] Jakub Zajkowski. Algorytmy mrówkowe – wprowadzenie. http: //www.ioz.pwr.wroc.pl/pracownicy/kuchta/referat.pdf. (dostęp 20.05.2016).

[50] TSPLib. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95. (dostęp 20.05.2016).

[51] Ewelina Świderska, Jakub Łasisz, Aleksander Byrski, Tom Lenaerts, Dana Samson, Bipin Indurkhya, Ann Nowé, and Marek Kisiel-Dorohinicki. Applications of

Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I, chapter Measuring Diversity of Socio-Cognitively Inspired ACO Search, 393–408. Springer International Publishing, Cham, 2016.

[52] Ming Li, Bin Ma, Lusheng Wang. On the closest string and substring problems. J. ACM, 49(2):157–171, 2002.