Opracowanie algorytmu ewolucyjnego do wyboru modelu normalnego

Raport

Chojecki Przemysław, Przybyłek Paulina
Numery indeksu 298814, 298837

19.06.2022

Raport projektu w ramach przedmiotu
Wstęp do Algorytmów Ewolucyjnych
na wydziale **Matematyki i Nauk Informacyjnych**
Politechniki Warszawskiej

Spis treści

1 Wstęp

Niniejszy dokument jest częścią projektu o charakterze badawczym, na który składają się jeszcze kod rozwiązujący problem oraz dokumentacja wstępna. Repozytorium projektu umieszczone jest na stronie GitHub [4], gdzie znajduje się implementacja rozwiązania oraz najnowsze wersje wszystkich przygotowywanych dokumentów w ramach tego projektu.

1.1 Cel projektu

Celem projektu jest opracowanie *algorytmu ewolucyjnego* (ang. *Ewolutionary Optimization, EO*) dobrze radzącego sobie w szukaniu maksimum funkcji celu z rodziny opisanej w pracy [1]. Autorzy wspomnianej pracy próbowali znajdować maksimum tych funkcji za pomocą algorytmu *Metropolisa-Hastingsa (MH)*, dlatego to z nim zdecydowano porównywać *EO*. Dziedziną funkcji celu jest zbiór permutacji p elementowych. Dziedzina więc jest zbiorem dyskretnym oraz skończonym, jednakże już dla $p = 20$ bardzo dużym i niemożliwym do przejrzenia algorytmem *"Brute Force"*.

1.2 Skrót dokumentacji wstępnej

W dokumentacji wstępnej przedstawiono motywacje, jakie przyświecały autorom przy rozpoczynaniu projektu. Opisano rozważaną rodzinę funkcji celu, a także zawarto szkic rozwiązania, którego uszczegółowienie oraz wnioski z jego zastosowania znajdują się w tymże dokumencie.

2 Opis problemu

Każda funkcja z rodziny rozważanych funkcji celu przyjmuje jako argument permutację, a zwraca dodatnią liczbę rzeczywistą. Rodzina ta jest parametryzowana kilkoma parametrami: p, n, U, δ oraz D, gdzie $p \in \mathbb{N}$ to długość permutacji przyjmowanych jako argument funkcji, $n \in \mathbb{N}$, $\delta \geqslant 3$, U oraz D są kwadratowymi macierzami symetrycznymi wymiaru $p \times p$.

Parametry δ oraz D parametryzują rozkład a priori i zgodnie z koncepcją statystyki Bayesowskiej, nie powinny być optymalizowane, gdyż reprezentują założenia przyjęte przed przystąpieniem do eksperymentów. Pozostałe parametry są szczegółowo opisane w dodatku A.

Warto jednak wspomnieć, że po ustaleniu parametrów p, n, U, δ oraz D funkcja celu jest proporcjonalna do funkcji wiarogodności permutacji przy zaobserwowanych danych. Oznacza to, że znalezienie maksimum tej funkcji jest równoznaczne ze znalezieniem maksimum funkcji wiarogodności, a co za tym idzie, ze znalezieniem estymatora największej wiarogodności tej permutacji. Wybór tej permutacji można utożsamić z **wyborem rozkładu normalnego** modelującego zaobserwowane dane.

Należy zauważyć, że już dla wartości $p = 20$ dziedzina funkcji celu jest niemożliwa do przeszukania ze względu na swoją wielkość. Dlatego algorytmy ewolucyjne wydają się adekwatnym wyborem do rozwiązywania tego problemu.

3 Zmiany względem wstępnej dokumentacji

Dokumentacja wstępna nie podawała szczegółów związanych z adaptacją mutacji. W tym dokumencie uszczegółowiono to zagadnienie i opisano w podsekcji 4.2.1.

Zaimplementowano dodatkowe algorytmy referencyjne w celach porównania się do nich. Są to algorytmy *największego wzrostu* (ang. *Best Growth*, *BG*) oraz *Monte Carlo* (*MC*). Wspomniany algorytm *MH* został jedynie wykorzystany w projekcie przez autorów, jego implementacja nie należała do części wykonanej pracy.

Hiperparametry algorytmu *EO* nie były strojone metodą *"random search"* jak zakładano w dokumentacji wstępnej. Zamiast tego przeprowadzono procedurę strojenia iteracyjnego w eksperymentach 1 oraz 2, która jest opisana w podsekcji 5.3.2.

Poprzednio założono wykonywanie wykresów z użyciem pakietu *ggplot2*. Jednakże wykresy ostatecznie wykonano w bazowym dla języka *R* pakiecie *graphics*.

W dokumentacji wstępnej planowano również, że przeprowadzone zostanie porównanie wyników *EO* z wynikami *MH*, które otrzymane są w pracy [1] w sekcjach 5.1 oraz 5.2. Okazało się jednak po dokładnej analizie, że algorytm analizowany przez autorów w sekcji 5.1 nie jest zaimplementowany w pakiecie *gips*. Dlatego zrezygnowano z porównania z wynikami z sekcji 5.1. Przeprowadzono jednak eksperyment przedstawiony w sekcji 5.2 dokumentu [1] i nadano mu w niniejszym dokumencie nazwę "Eksperyment 4". Opisano go wraz z pozostałymi eksperymentami w rozdziale 5.

4 Opis rozwiązania

Rozdział zawiera podsumowanie części technicznej projektu. Kod do tego projektu powstawał w języku R [5], a dostępny jest na platformie GitHub na stronie [4]. Algorytm *BG* (który również był zaprogramowany w ramach tego projektu, a więcej na jego temat znajduje się w podsekcji 4.3) został przeniesiony do pakietu 'gips', który to z kolei dostępny jest pod adresem [3].

4.1 Co zostało zrobione w projekcie

Część techniczna projektu składa się z trzech głównych części:

- Implementacja algorytmów *MC* oraz *BG* w celach porównania wyników i analizy. Szczegóły dotyczące algorytmów rozwinięto w podsekcji 4.3.
- Implementacja algorytmu ewolucyjnego (*EO*) do maksymalizacji funkcji z rozważanej dziedziny. Został on zaimplementowany zgodnie z opisem w dokumentacji wstępnej z uszczegółowieniem procesu adaptacji mutacji, co zostało przedstawione w podsekcji 4.2.1. Opis działania algorytmu *EO* znajduje się w sekcji 4.2.
- Implementacja skryptów generujących ciągi rozwiązań dla konkretnych algorytmów oraz skrypty generujące wykresy na podstawie wyników z tamtych.

Przez cały okres prac wykorzystywana była funkcja celu opisana w pracy [1], a zaimplementowana w R w pakiecie 'gips' dostępnym pod adresem [3]. W tym samym pakiecie znajduje się wykorzystywana w tym projekcie implementacja algorytmu *MH* (o którym więcej znajduje się w podsekcji 4.3).

4.2 Algorytm EO

Do rozwiązania problemu zaproponowano algorytm *EO*, którego szkic działania wygląda następująco:

1. Ustalenie wartości parametrów:
   - ‘max_f_calls‘ - liczba możliwych do wykonania wywołań funkcji celu;
   - ‘pop_size‘ - wielkość populacji;
   - ‘init‘ - rodzaj inicjacji;
   - ‘p_0‘, ‘success_treshold‘, ‘a‘, ‘k_max‘ - parametry mutacji;
   - ‘tournament_part‘ - parametr selekcji.
2. Obliczenie wartości ‘x‘ potrzebnej do adaptacji mutacji, zgodnie z podsekcją 4.2.1.
3. Inicjacja populacji - w zależności od parametru:
   - ‘random‘ - każdy element populacji losowany niezależnie, jednostajnie z całej przestrzeni przeszukiwań;
   - ‘random_close‘ - pierwszy element populacji losowany jednostajnie z przestrzeni przeszukiwań, a pozostałe oddalone od niego o jedną transpozycję;
   - ‘id_close‘ - populacja losowana z rozkładu jednostajnego na transpozycjach, czyli w odległości jeden od permutacji identycznościowej.
4. Ewaluacja populacji początkowej i zapis ich wartości do loga.
5. Pętla główna:
   (a) Mutacja - dla każdego osobnika z populacji losowana jest liczba kroków do wykonania z rozkładu dwumianowego o parametrach ‘k_max‘ oraz ‘p_t‘, gdzie ‘p_t‘ jest parametrem adaptowanym zgodnie z opisem z podsekcji 4.2.1. Oznacza to, że mutacja może spowodować przesunięcie się osobnika o co najwyżej ‘k_max‘, a średnio będzie przesuwać o ‘k_max‘ · ‘p_t‘.
   (b) Ewaluacja zmutowanych osobników i zapis ich wartości do loga.
   (c) Adaptacja wartości ‘p_t‘ zgodnie z opisem w 4.2.1.
   (d) Selekcja turniejowa, gdzie w każdym turnieju bierze udział ‘pop_size‘ · ‘tournament_part‘ zawodników.
   (e) Warunek stopu - jeśli w następnej iteracji nastąpiłoby przekroczenie dozwolonej liczby wywołań funkcji celu, to zakończ pętlę.
6. Zwrócenie loga i zakończenie działania algorytmu.

4.2.1 Adaptacja mutacji

Adaptacja zastosowana w algorytmie *EO* jest mocno zainspirowana regułą 1/5 liczby sukcesów. Działa ona w oparciu o funkcję sigmoid:

$$sigm(x) = \frac{1}{1+e^{-x}},\ sigm^{-1}(y) = ln(\frac{y}{1-y})$$

Adaptowana jest wartość ‘p_t‘, która jest parametrem rozkładu dwumianowego i musi znajdować się w przedziale ‘p_t‘ $\in [0, 1]$.

Początkowe obliczenie wartości x (punkt 2 algorytmu *EO*) odbywa się zgodnie z procedurą:

1. Początkowa wartość ‘p_t‘ ustalana jest przez użytkownika na ‘p_0‘. W eksperymentach w tym dokumencie zawsze ustawiono ‘p_0‘$= 0.5$.
2. Obliczana jest wartość ‘x‘ odpowiadająca tej wartości ‘p_0‘ zgodnie ze wzorem:

$$x = sigm^{-1}(p\_0)$$

Adaptacja wartości ‘p_t‘ (punkt 5c algorytmu *EO*) przebiega zgodnie z następującym schematem:

1. Obliczana jest liczba ‘l‘ tych mutacji w aktualnej iteracji, które poprawiły wartość funkcji celu względem swojego rodzica.
2. Ułamek ‘l‘/‘pop_size‘ porównywany jest z podaną przez użytkownika wartością progową ‘success_treshold‘:
   - Jeśli sukcesów było dużo (‘l‘/‘pop_size‘ $\geqslant$ ‘success_treshold‘), to zwiększano wartość ‘x‘ o wartość parametru ‘a‘:

$$x := x + a$$

   - Jeśli sukcesów było mało (‘l‘/‘pop_size‘ $<$ ‘success_treshold‘), to zmniejszano wartość ‘x‘ o wartość parametru ‘a‘:

$$x := x - a$$

3. Obliczana jest nowa wartość ‘p_t‘:

$$p\_t := sigm(x)$$

4.3 Algorytmy pomocnicze

W celach porównawczych zaimplementowano dwa dodatkowe algorytmy optymalizacyjne:

- Monte Carlo
- oraz największego wzrostu.

Algorytm *Monte Carlo (MC)* jest algorytmem, który iteracyjnie losuje permutacje jednostajnie z całej dziedziny oraz liczy jej wartość funkcji celu. Algorytm *największego wzrostu (BG)* jest algorytmem zachłannym - w jednej iteracji liczy wartość funkcji celu dla wszystkich sąsiadów, a następnie idzie do tego, którego wartość jest największa. *BG* kończy swoje działanie, gdy wszyscy sąsiedzi są gorsi od aktualnego punktu.

5 Eksperymenty numeryczne

W celu przeanalizowania algorytmu *EO* i porównania go z algorytmami *MH*, *MC* oraz *BG*, przeprowadzono szereg eksperymentów obliczeniowych. Każdy eksperyment od 0 do 4 wykonywany był na innej funkcji celu z rozważanej rodziny.

Cele przeprowadzonych eksperymentów numerycznych:

- Eksperyment 0 miał na celu pokazanie, że algorytm *EO* działa.
- Eksperymenty 1 oraz 2 miały za zadanie dobranie jak najlepszych parametrów dla *EO*.
- Eksperyment 3 służył porównaniu algorytmów *MH* oraz *EO* na dużej próbce.
- Eksperyment 4 jest powtórzeniem eksperymentu z pracy [1] z sekcji 5.2 dla algorytmu *MH* oraz porównaniem wyniku z algorytmem *EO*.

5.1 Hipotezy

Sekcja stanowi przypomnienie hipotez z dokumentacji wstępnej pracy.
Postawiono następujące hipotezy:

1. Opracowany algorytm ewolucyjny będzie znajdował satysfakcjonującą wartość funkcji celu szybciej od algorytmu 'Metropolis_Hastings' z pakietu R *gips* ([3]).
2. Algorytm ewolucyjny z większymi mutacjami będzie dawał lepsze wyniki w porównaniu do podstawowego.

Hipotezy te zostały zweryfikowane poprzez wykonane eksperymenty.

5.2 Metodologia

Aby zaprezentować jakość algorytmów zastosowano wykresy ECDF. Na nich na osi OX znajduje się zlogarytmowana liczba wywołań funkcji celu. Natomiast na osi OY najlepsza znaleziona wartość funkcji celu do danego momentu. Funkcja celu została jednak przeskalowana dla każdego eksperymentu tak, aby jej wartości były mniejsze od 1. Było to możliwe dzięki temu, że dane w tych eksperymentach są sztuczne, więc znane jest rozwiązanie. Na wartość 0 została przeskalowana mediana z wartości wylosowanych 5000 permutacji, natomiast wartość 1 oznacza rozwiązanie funkcji. Na wykresie zaznaczono dodatkowo czarną linią wartość funkcji celu dla permutacji identycznościowej. To właśnie z tej permutacji "startują" algorytmy *MH* oraz *BG*.

Do porównania wykorzystano testy statystyczne:

- t-Studenta, powszechnie używany test statystyczny, który porównuje średnie przy założeniu, że dane mają rozkład t-Studenta [6];
- Wilcoxon'a, używany m.in. w benchmarku COCO [2], który porównuje mediany nieparametrycznie, czyli bez założeń o rozkładzie [7] [8].

5.3 Opis eksperymentów

Niniejsza sekcja opisuje jakie eksperymenty zostały zrealizowane w projekcie, w jakim celu oraz jakie wnioski z nich wynikają. We wszystkich eksperymentach przeanalizowano sytuację sztucznych danych wygenerowanych ze znanego rozkładu. Dzięki temu przed wywołaniem algorytmu znana jest permutacja, dla której przyjmowana jest największa wartość.

W eksperymentach od 0 do 3 użyto wielkości danych $n = 20$, a w eksperymencie 4 użyto $n = 200$, aby być zgodnymi z autorami pracy [1].

W eksperymentach od 1 do 3 algorytm *BG* uruchomiono 1 raz, gdyż jest to algorytm deterministyczny. Algorytmy *MC* oraz *MH* wywołane był 100 razy.

5.3.1 Eksperyment 0

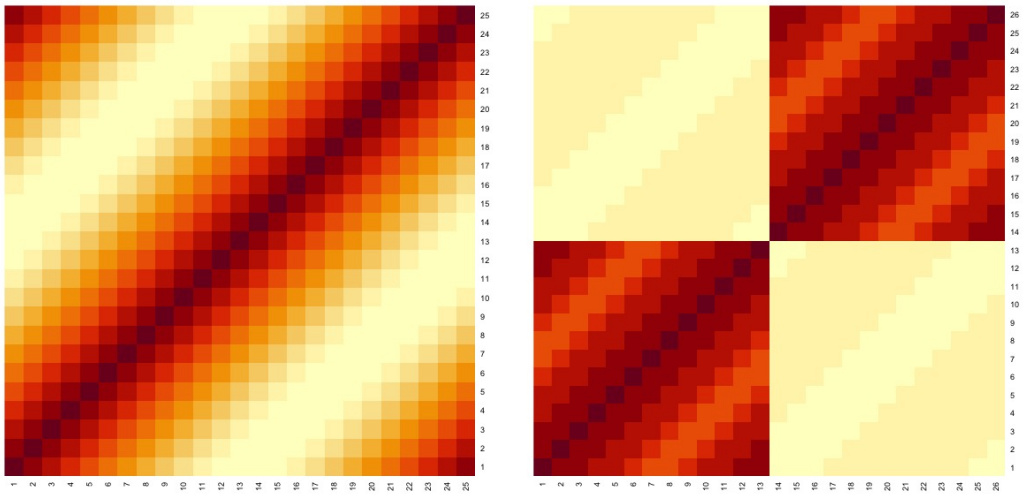
Eksperyment dla małego wymiaru $p = 6$ przeprowadzony w celu sprawdzenia czy algorytm działa i znajduje rozwiązanie. Algorytm *EO* nie był w tym eksperymencie porównywany z żadnym innym algorytmem.

Eksperyment 0 odpowiada pierwszemu eksperymentowi z dokumentacji wstępnej opisanemu w rozdziale "5 Metodologia".

5.3.2 Eksperymenty 1 i 2

Eksperymenty 1 i 2 przeprowadzono w celu dostrojenia wartości parametrów algorytmu *EO*. Wykonane były na podobnym wymiarze (kolejno $p = 25$ oraz $p = 26$), ale na innej macierzy i innej permutacji docelowej. W eksperymencie 1 docelowa permutacja była pojedynczym cyklem $(1, 2, ..., 24, 25)$, natomiast w eksperymencie 2 docelowa permutacja była podwójnym cyklem $(1, 2, ..., 12, 13)(14, 15, ..., 25, 26)$. Z tego powodu podejrzewano, że eksperyment 2 będzie łatwiejszy (łatwiej jest znaleźć dwa mniejsze części rozwiązania niż jedno większe rozwiązanie; zasada dziel i rządź). Na dodatek postarano się, aby ułatwić algorytmowi odróżnienie części $(1, ..., 13)$ od części $(14, ..., 26)$ w taki sposób, że wewnątrz cykli kowariancje były dodatnie, a poza cyklami - ujemne. Obie macierze zobrazowane są na mapie ciepła (ang. heat map) na wykresie 1.

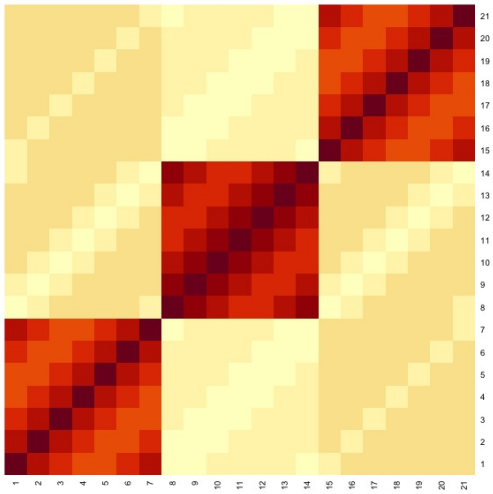
W ramach tych eksperymentów iteracyjnie dostrajano kolejne parametry algorytmu *EO* w kolejności: 'a', 'k_max', 'pop_size', 'tournament_part', 'success_treshold', 'init'. Dostrajanie parametru 'init', w celu dokładniejszego zbadania, wywołano na 10 razy większym budżecie.



Rysunek 1: Heatmapy macierzy kowariancji Σ użytych do generowania danych dla eksperymentów 1 (po lewej) oraz 2 (po prawej). Macierze te są wymiaru kolejno $p = 25$ oraz $p = 2 \cdot 13 = 26$. Można zauważyć, że macierz po lewej ma strukturę globalną, a macierz po prawej ma dwie struktury mniejsze - bardziej lokalne. Z tego powodu można podejrzewać, że struktura macierzy po prawej będzie łatwiejsza do odkrycia przez algorytm.

5.3.3 Eksperyment 3

Eksperyment 3 przeprowadzono w celu wykonania testów statystycznych i sprawdzenia, czy *EO* daje statystycznie istotnie lepsze wyniki od *MH*. Wykonano ten eksperyment na przestrzeni $p = 21$ oraz poszukiwanej macierzy pokazanej na wykresie 2. W tym eksperymencie wykonano większą liczbę powtórzeń algorytmu EO niż w innych eksperymentach (200 powtórzeń).

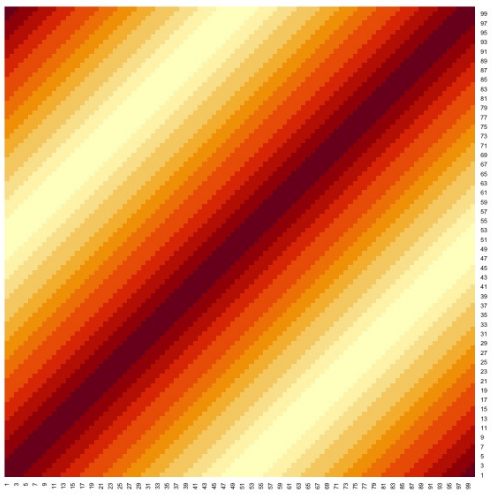


Rysunek 2: Heatmapa macierzy kowariancji Σ użytej do generowania danych dla eksperymentu 3. Macierz ta jest wymiaru $p = 3 \cdot 7 = 21$. Rozwiązaniem dla tej macierzy jest cykl (1,2,...,7)(8,...,14)(15,...,21). Można zauważyć, że macierz ma trzy małe, lokalne struktury. Z tego powodu można podejrzewać, że struktura tej macierzy będzie łatwiejsza do odkrycia przez algorytm od struktur z wykresów 1 oraz 3.

Eksperyment 3 odpowiada drugiemu eksperymentowi z dokumentacji wstępnej opisanemu w rozdziale "5 Metodologia".

5.3.4 Eksperyment 4

Eksperyment 4 zrealizowano w celu sprawdzenia jak na bardzo dużej przestrzeni $p = 100$ poradzą sobie algorytmy *EO* i *MH*. Wykonano ten eksperyment na macierzy pokazanej na wykresie 3. Algorytmy wywołano jedynie raz ze względu na czasochłonność obliczeń.



Rysunek 3: Heatmapa macierzy kowariancji Σ użytej do generowania danych dla eksperymentu 4. Można zauważyć, że macierz ma jedną dużą, globalną strukturę. Na dodatek, macierz ta jest dużego wymiaru ($p = 100$). Z tego powodu można podejrzewać, że struktura tej macierzy będzie najtrudniejsza do odkrycia przez algorytm spośród struktur z eksperymentów 1, 2 oraz 3, a zaprezentowanych na wykresach 1 oraz 2.

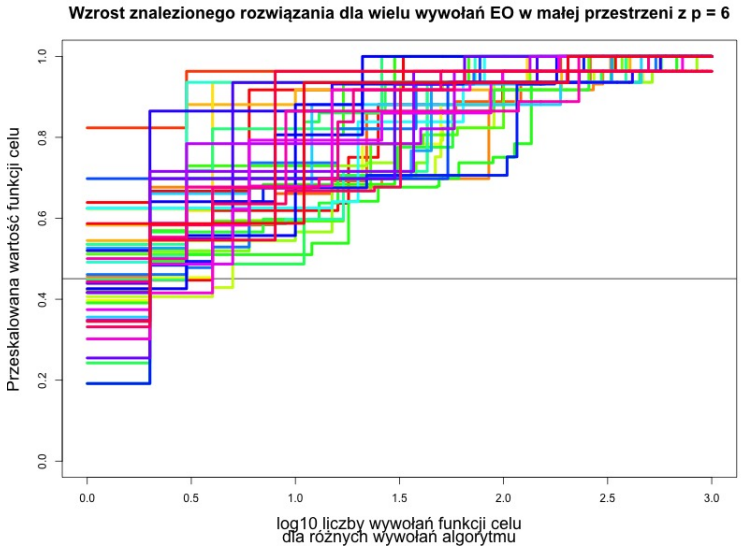
Eksperyment 4 odpowiada trzeciemu eksperymentowi z dokumentacji wstępnej opisanemu w rozdziale "5 Metodologia".

5.4 Wyniki

Sekcja zawiera opis wyników z przeprowadzonych eksperymentów oraz wnioski, które z nich wynikają. Eksperymenty wykonane były w kolejności opisanej w tej sekcji, dlatego też np. parametry użyte w eksperymencie 3 są tymi, które zostały wybrane w eksperymencie 2.

5.4.1 Eksperyment 0

Na wykresie ECDF 4 pokazano 20 krótkich przebiegów działania algorytmu *EO*. Widać, że wszystkie one poprawiały się w czasie i ostatecznie trafiły do najlepszego, bądź prawie najlepszego punktu. Na podstawie tego wykresu uznano, że algorytm *EO* został zaimplementowany poprawnie i że działa dla tej funkcji celu.

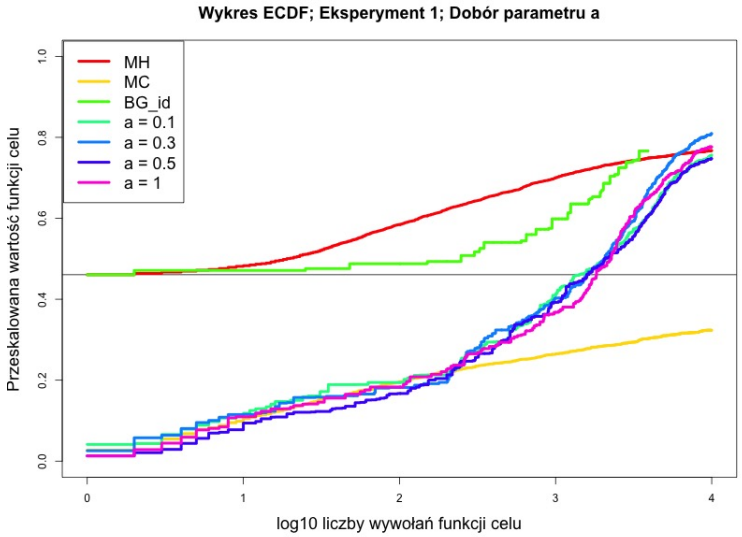


Rysunek 4: Zwiększanie jakości znalezionego rozwiązania na łatwej funkcji celu. Można zauważyć, że jakość rozwiązania rośnie w czasie, a po $10^3 = 1000$ wywołaniach osiąga albo maksimum, albo niewiele brakuje do tej wartości.

5.4.2 Eksperyment 1

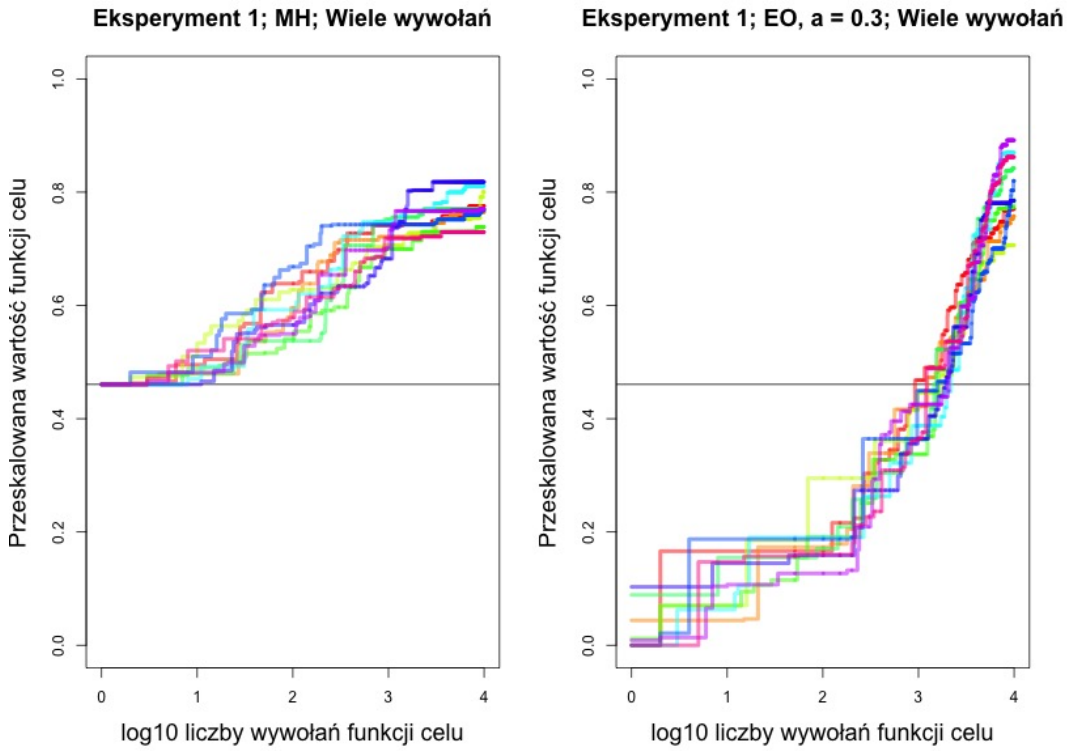
Początkowo w tym eksperymencie użyto wartości parametrów ‘pop_size‘ = 100, ‘success_treshold‘ = 0 ‘a‘ = 1, ‘k_max‘ = 5, ‘tournament_part‘ = 0.5, ‘init‘ = ”random”. Z kolejnymi analizami w ramach tego eksperymentu zmieniano jednak wybrane parametry w zależności od tego, które poradziły sobie najlepiej.

Na wykresie ECDF 5 pokazano przebieg średniego najlepszego znalezionego rozwiązania w czasie dla algorytmów *MH*, *BG*, *MC* oraz kilku wersji *EO* z różnymi wartościami parametru **‘a‘**.



Rysunek 5: Porównanie uśrednionych przebiegów algorytmu EO o różnych wartościach parametru ‘a‘ z przebiegami algorytmów MH, MC oraz BG. Średnio EO z wartością parametru ‘a‘=0.3 daje najlepszy wynik, choć wydaje się on bliski MH i pozostałym przebiegom.

Widać na nim, że wersja z wartością parametru **'a'=0.3** jest średnio najlepsza po 10^4 krokach, choć różnice wydają się niewielkie. Należy sprawdzić, czy te "niewielkie" na pierwszy rzut oka różnice między średnimi mają swoje potwierdzenie w rozrzucie wyników. W tym celu narysowano wykres 6, na którym ukazano przebiegi wszystkich trajektorii dla algorytmu *MH* oraz *EO* z wartością parametru 'a'=0.3. Można na nim zauważyć, że rozrzut wyników algorytmu *EO* jest duży w porównaniu z różnicami w średnich na wykresie 5.

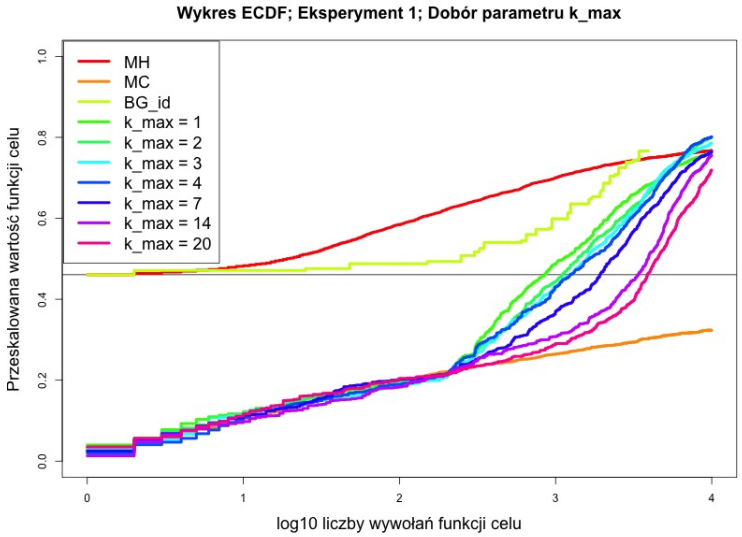


Rysunek 6: Porównanie najlepszego rozwiązania w danej chwili dla wszystkich przebiegów algorytmów MH oraz EO z parametrem 'a'=0.3. Na wykresie tym widać, że rozproszenie wartości wyniku po 10^4 wywołaniach funkcji celu jest duże. Nie można więc wnioskować o lepszości algorytmu EO z parametrem 'a'=0.3 nad algorytmem MH, mimo lepszej dla niego średniej.

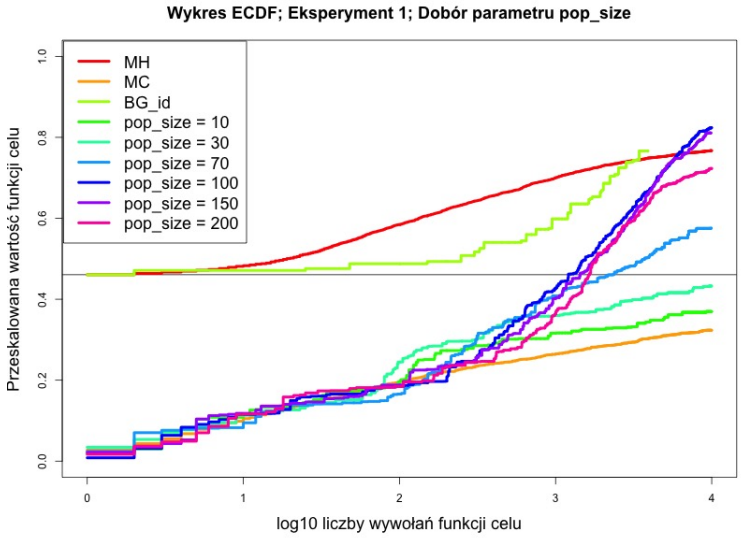
Podobne wykresy rozproszenia sporządzono dla wszystkich pozostałych eksperymentów w tym raporcie i zawsze rozrzut był podobnej wielkości, czyli około 0.2 w skali Y. Z tego powodu w pozostałej części raportu nie zawarto tych wykresów, a jedynie wykresy średnich wyników.

Na wykresie ECDF 7 pokazano przebieg średniego najlepszego znalezionego rozwiązania w czasie dla algorytmów *MH*, *BG*, *MC* oraz kilku wersji *EO* z różnymi wartościami parametru **'k_max'**. Można zauważyć, że wersja z wartością parametru **'k_max'=7** jest średnio najlepsza po 10^4 krokach. Różnice w znalezionych rozwiązaniach są jednak niewielkie w porównaniu z rozrzutem.

Na wykresie ECDF 8 pokazano przebieg średniego najlepszego znalezionego rozwiązania w czasie dla algorytmów *MH*, *BG*, *MC* oraz kilku wersji *EO* z różnymi wartościami parametru **'pop_size'**. Widać na nim, że wersja z wartością parametru **'pop_size' = 100** jest średnio najlepsza po 10^4 krokach, ale bardzo bliska wartości średniej dla 'pop_size' = 150. Jednakże dla pozostałych wartości średnia jest zdecydowanie gorsza.

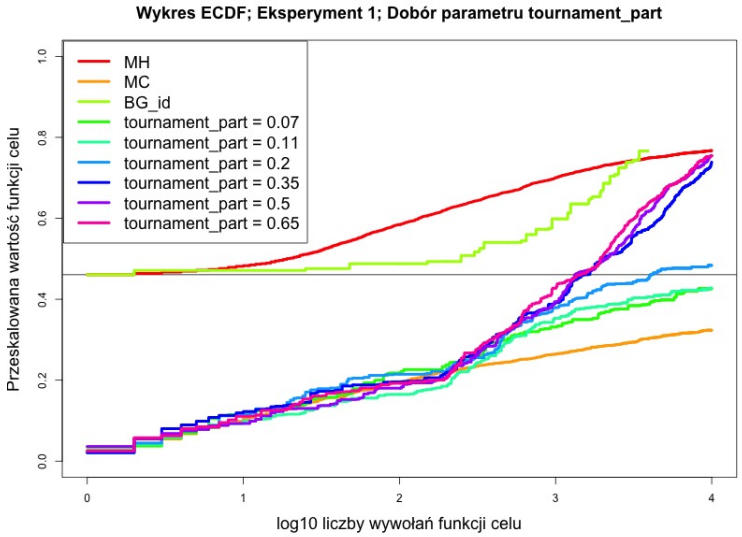


Rysunek 7: Porównanie uśrednionych przebiegów algorytmu EO o różnych wartościach parametru ‘k_max‘ z przebiegami algorytmów MH, MC oraz BG. Widać na nim, że średnio EO z wartością parametru ‘k_max‘=7 daje najlepszy wynik, choć jest on bliski MH i pozostałym przebiegom.



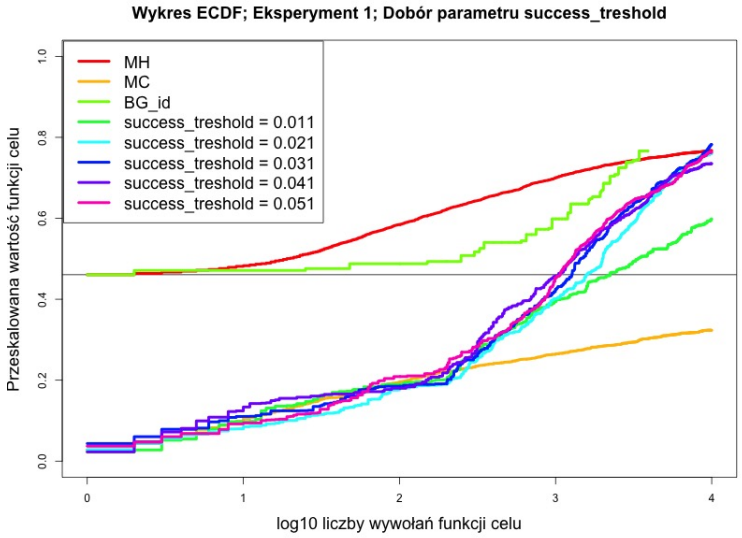
Rysunek 8: Porównanie uśrednionych przebiegów algorytmu EO o różnych wartościach parametru ‘pop_size‘ z uśrednieniami przebiegów algorytmów MH, MC oraz BG. Widać na nim, że średnio EO z wartością parametru ‘pop_size=100‘ daje najlepszy wynik, choć wydaje się on bliski MH. Podobna jakość rozwiązania osiągana jest dla wartości ‘pop_size‘ = 150, a pozostałe testowane wartości są znacząco gorsze.

Na wykresie ECDF 9 pokazano przebieg średniego najlepszego znalezionego rozwiązania w czasie dla algorytmów *MH*, *BG*, *MC* oraz kilku wersji *EO* z różnymi wartościami parametru **‘tournament_part‘**. Widać na nim, że wersja z wartością parametru **‘tournament_part‘=0.65 lub 0.5 lub 0.35** jest średnio najlepsza po 10^4 krokach, oraz bardzo podobna do wyniku *MH*. Pozostałe przebiegi dają znacząco gorsze wyniki.



Rysunek 9: Porównanie uśrednionych przebiegów algorytmu EO o różnych wartościach parametru 'tournament_part' z uśrednieniami przebiegów algorytmów MH, MC oraz BG. Widać na nim, że średnio EO z wartością parametru 'tournament_part'=0.65 lub 0.5, lub 0.35 dają najlepsze wyniki, choć wydają się one bliskie MH. Pozostałe wartości parametru 'tournament_part' dają znacząco gorsze wyniki.

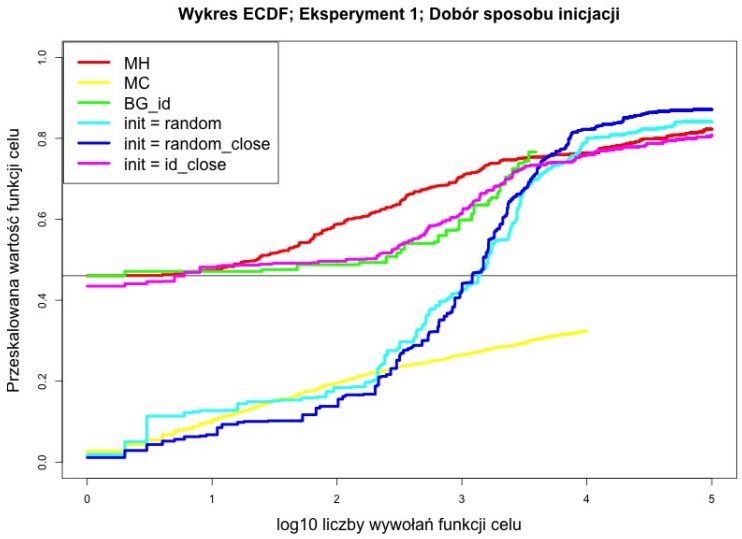
Na wykresie ECDF 10 pokazano przebieg średniego najlepszego znalezionego rozwiązania w czasie dla algorytmów *MH*, *BG*, *MC* oraz kilku wersji *EO* z różnymi wartościami parametru **'success_treshold'**. Widać na nim, że wersja z wartością parametru **'success_treshold'=0.011** jest średnio najgorsza po 10^4 krokach, a pozostałe są równie dobre.



Rysunek 10: Porównanie uśrednionych przebiegów algorytmu EO o różnych wartościach parametru 'success_treshold' z uśrednieniami przebiegów algorytmów MH, MC oraz BG. Widać na nim, że średnio EO z wartością parametru 'success_treshold'=0.011 daje najgorszy wynik, a pozostałe wartości tego parametru dają wyniki bliskie wartością MH.

Na koniec eksperymentu 1 zbadano wpływ parametru **'init'**. Na pierwszy rzut oka wydaje się on największy, gdyż wartość 'id_close' powoduje, że od razu startuje się z w miarę dobrego rozwiązania. Ten test przeprowadzono na większym budżecie 10^5 zamiast 10^4, aby zniwelować wpływ "lepszego początku" dla inicjacji typu "id_close".

Na wykresie ECDF 11 pokazano przebieg średniego najlepszego znalezionego rozwiązania w czasie dla algorytmów *MH*, *BG*, *MC* oraz kilku wersji *EO* z różnymi wartościami parametru **'init'**. Widać na nim, że po 10^5 wywołań funkcji celu niema dużych różnic między wynikami algorytmów.



Rysunek 11: Porównanie uśrednionych przebiegów algorytmu EO o różnych wartościach parametru 'init' z uśrednieniami przebiegów algorytmów MH, MC oraz BG. Widać na nim, że średnio po 10^5 iteracji EO daje podobne wyniki niezależnie od sposobu inicjacji.

Podsumowanie eksperymentu

W tym eksperymencie wyszło, że istotnymi parametrami dla jakości algorytmu *EO* są 'pop_size' oraz 'tournament_part'. Okazało się, że rodzaj inicjacji nie jest zbyt ważny ostatecznie, choć w początkowej fazie 'init'="id_close" znajduje najlepsze rozwiązania.

Ostatecznie do następnego eksperymentu wybrano następujące parametry: 'a' = 0.3, 'success_treshold' = 0.031, 'k_max' = 7, 'tournament_part' = 0.5, 'init' = "random_close", 'pop_size' = 100.

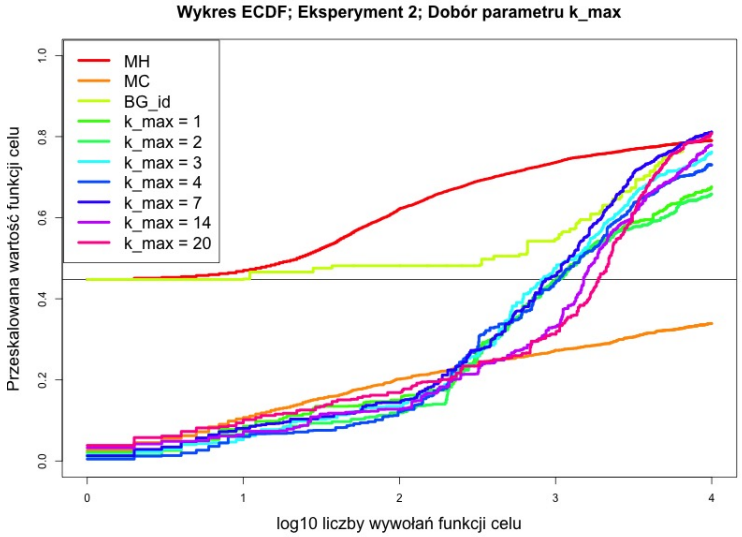
5.4.3 Eksperyment 2

Jako, że dostosowanie parametrów w eksperymencie 1 odbyło się na (w opinii autorów tego raportu) trudnej macierzy kowariancji, to powtórzono dobór parametrów na trochę łatwiejszej. Eksperyment 2 sprowadza się do tego samego, co eksperyment 1.

Aby jednak nie powtarzać całego opisu (gdyż wtenczas ważne wnioski umknęłyby w ogromnym opisie) wybrano do opisania te przebiegi, które przyniosły najciekawsze rezultaty.

Na wykresie ECDF 12 pokazano przebieg średniego najlepszego znalezionego rozwiązania w czasie dla algorytmów *MH*, *BG*, *MC* oraz kilku wersji *EO* z różnymi wartościami parametru **'k_max'**. Widać na nim, że wersja z wartością parametru **'k_max'=7** jest

średnio najlepsza po 10^4 krokach. Należy zwrócić uwagę, że w porównaniu z 7 teraz lepsze wyniki są osiągane dla większych wartości tego parametru. Intuicyjnie oznacza to, że większa mutacja była w tym problemie pożądana, co zgadza się z początkową intuicją, że problem ten jest prostszy niż ten omawiany w eksperymencie 1.



Rysunek 12: Porównanie uśrednionych przebiegów algorytmu EO o różnych wartościach parametru ‘k_max‘ z przebiegami algorytmów MH, MC oraz BG. Widać na nim, że średnio EO z wartością parametru ‘k_max‘=7 daje najlepszy wynik, choć wydaje się on bliski MH i pozostałym przebiegom.

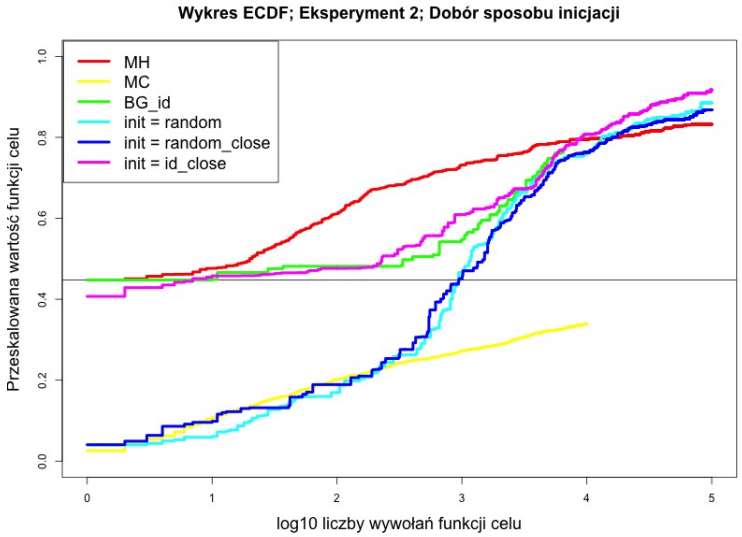
Na wykresie ECDF 13 pokazano przebieg średniego najlepszego znalezionego rozwiązania w czasie dla algorytmów *MH*, *BG*, *MC* oraz kilku wersji *EO* z różnymi wartościami parametru **‘init‘**. Widać na nim, że po 10^5 wywołań funkcji celu algorytm *EO* daje ciut lepsze wyniki dla ‘init‘=”id_close”. Warto jednak zwrócić uwagę, że teraz kolejność jakości średniego rozwiązania po 10^5 jest odwrotna od analogicznego testu przedstawionego na wykresie 11. Tym razem najlepszy okazał się ‘init‘=”id_close”.

Podsumowanie eksperymentu

Dla eksperymentu 2 najlepsze rezultaty dawały większe wartości parametru ‘k_max‘ niż w analogicznej sytuacji dla eksperymentu 1. Zgadza się to z intuicję, że eksperyment 2 jest prostszy od eksperymentu 1.

Spostrzeżono odwrotną zależność od sposobu inicjacji niż w eksperymencie 1. Dalsze badania nad tym parametrem przeprowadzono w eksperymencie 3 w podsekcji 5.4.4.

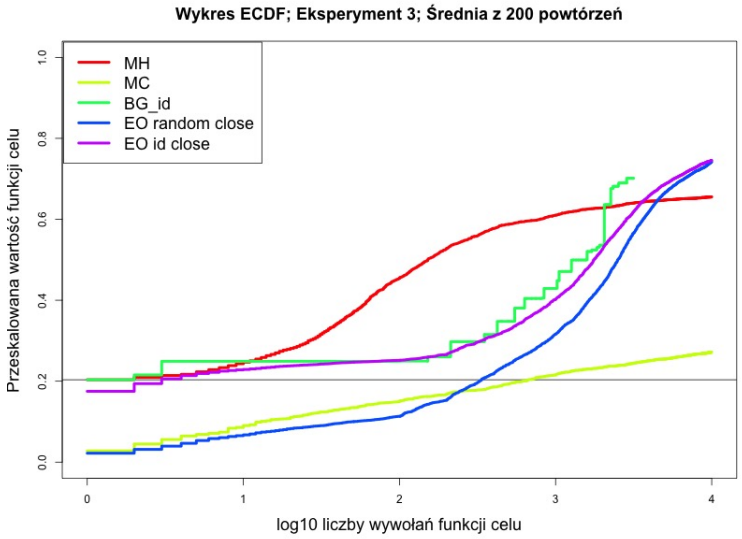
Ostatecznie do następnego eksperymentu wybrano następujące parametry: ‘a‘ = 0.3, ‘pop_size‘ = 100, ‘success_treshold‘ = 0.031, ‘k_max‘ = 7, ‘tournament_part‘ = 0.35, oraz do przetestowania: ‘init‘ = ”random_close” lub ‘init‘ = ”id_close”.



Rysunek 13: Porównanie uśrednionych przebiegów algorytmu EO o różnych wartościach parametru 'init' z przebiegami algorytmów MH, MC oraz BG. Widać na nim, że średnio po 10^5 iteracji EO daje ciut lepsze wyniki dla 'init'="id_close".

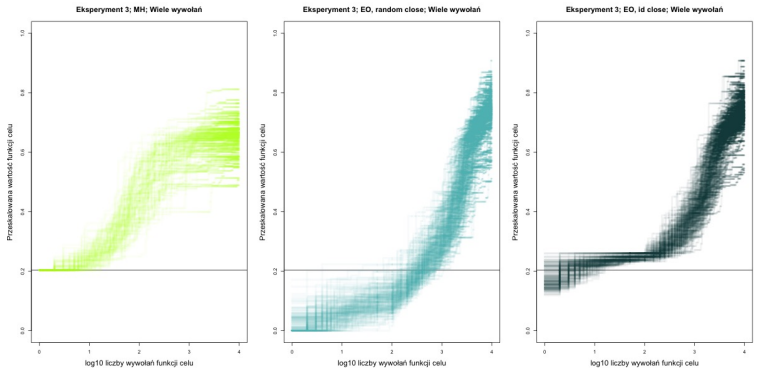
5.4.4 Eksperyment 3

Na wykresie ECDF 14 pokazano porównanie średnich wywołań algorytmu *MH* (średnia ze 100 przebiegów) oraz *EO* (średnia z 200 przebiegów) z odpowiednio inicjacją typu "random close" oraz "id close". Pokazano na nim, że obie te wersje algorytmu przez długi czas średnio były gorsze od algorytmu *MH*, ale pod koniec przegoniły go w jakości rozwiązania. Porównując jednak wersje inicjacji między sobą, wydaje się, że zbiegły średnio do bardzo bliskich wartości.



Rysunek 14: Średnia z dużej liczby przebiegów optymalizacji algorytmami MH oraz EO. Można zauważyć konsystencje - na małych budżetach algorytm MH radzi sobie lepiej, ale na większych algorytm EO radzi sobie lepiej i ostatecznie niema znaczenia, by był wystartowany z permutacji identycznościowej czy losowej.

Na wykresie ECDF 15 pokazano wszystkie trajektorie, których średnie pokazane były na poprzednio omawianym wykresie 14. Można zauważyć, nawet biorąc pod uwagę rozproszenie wyników, algorytm *EO* poradził sobie po 10^5 lepiej niż algorytm *MH*.



Rysunek 15: Wszystkie trajektorie optymalizacji algorytmami MH oraz EO. Na wykresie tym widać, że średnie przewyższanie algorytmu MH przez algorytm EO, które było pokazane na wykresie 14, NIE jest przypadkowe, a konsystentne. Jednakże nie można zauważyć ostatecznie różnic między sprawdzonymi metodami inicjacji algorytmu EO.

Podobne konkluzje wynieść można z testów statystycznych, ktrórych wyniki przedstawia tabela 1.

	EO random_close	**EO id_close**
MH	p-val $< 2.2 \cdot 10^{-16}$ (EO jest lepsze)	p-val $< 2.2 \cdot 10^{-16}$ (EO jest lepsze)
EO random_close	-	$\text{p-val}_{t-test} = 0.56$ $\text{p-val}_{Wilcox} = 0.75$

Tabela 1: Testy statystyczne t-Studenta na równość średnich oraz Wilcoxon'a na równość median. Oba one dla porównania MH z EO wyszły zdecydowanie ku odrzucenia hipotezy o równości. Należy więc przyjąć wniosek, że EO jest lepszy. Porównanie ze sobą różnych wersji EO nie daje statystycznie istotnych wniosków. Po 10^5 kroków, efekty działania tych dwóch algorytmów są podobne zarówno patrząc na średnią, jak i medianę.

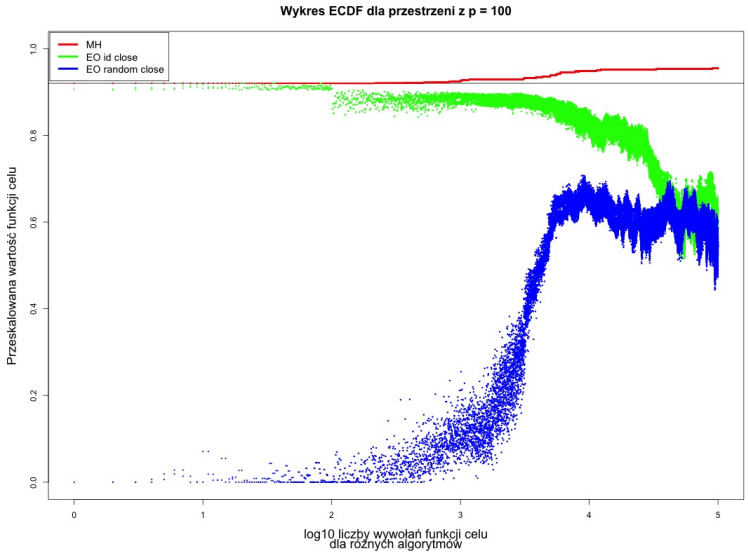
5.4.5 Eksperyment 4

Eksperyment 4 przeprowadzono na trudnej funkcji celu. Szukano permutacji jednocyklicznej w dużej przestrzeni wymiaru $p = 100$.

Na wykresie 16 pokazano zapis loga dla 3 algorytmów szukających maksimum tej funkcji.

Można zauważyć, że algorytm *EO* zupełnie nie poradził sobie z zadaniem. Przez długi czas błądził w podobnych wartościach funkcji celu i nie potrafił poprawiać się ponad wartość przeskalowaną do 0.7. Gdy zaczął na większych wartościach, to spadł ostatecznie do tych mniejszych.

Takie zachowanie byłoby spodziewane w sytuacji, gdy funkcja celu ma dużo tak zwanych "igieł", czyli miejsc o dobrej jakości funkcji przystosowania, które są otoczone słabymi. Być może tak właśnie jest w tym przypadku? Odpowiedź na to pytanie wymagałaby analizy teoretycznej problemu, która nie jest obiektem badań tego raportu.



Rysunek 16: Wartości funkcji celu obliczanych dla algorytmów szukających maksimum trudnej funkcji celu.

6 Podsumowanie

Zaprojektowany algorytm ewolucyjny w badanych przestrzeniach o wymiarze $p < 30$, oraz w miarę łatwych funkcjach celu, działa statystycznie istotnie lepiej niż algorytm *MH*. Oznacza to pozytywne rozstrzygnięcie hipotezy 1.

Okazuje się jednak, że dla "trudnych" funkcji poprawa ta jest niewielka i traci na istotności statystycznej. Ewidentnie więc okazuje się, że dostrojone parametry dla małego wymiaru nie nadają się do optymalizacji w dużym wymiarze.

Siłą algorytmu *MH* jest to, że nie posiada on żadnych parametrów, które trzeba by dostrajać i, jak można zauważyć w wynikach, zadziałał on bardzo dobrze we wszystkich eksperymentach.

Hipoteza 2 również okazała się prawdziwa, algorytm *EO* działał lepiej dla wartości ‘k_max‘ > 1 niż dla ‘k_max‘ $= 1$.

Cel projektu został całkowicie osiągnięty dla przestrzeni niskowymiarowych ($p < 30$). Niestety, ale nie udało się to dla zbadanej funkcji w wymiarze $p = 100$. Być może dostrojenie parametrów algorytmu *EO* do większego wymiaru spowodowałoby polepszenie jego wyników względem *MH*.

6.1 Możliwości dalszego rozwoju

Dokument ten jest jednym z początkowych prac dotyczących omawianej rodziny funkcji celu. Z tego powodu przeprowadzona analiza jest pierwszą próbą użycia algorytmu *EO* do optymalizacji wspomnianej dziedziny. Aby dalej rozwijać algorytm *EO* do tego celu można zająć się następującymi aspektami:

- Więcej eksperymentów - dostosowanie parametrów *EO* powinno być na podstawie wielu funkcji celu (w COCO bbob [2] jest ich 19), a nie tak niewielkiej liczbie jak w tym raporcie.
- Dłuższe eksperymenty - na budżecie 10^4, używanym w tym raporcie, wiele przebiegów wykazywało jeszcze dużą tendencję ku wzrostom - nie zbiegły jeszcze do

docelowych wartości.

- Więcej przebiegów - wykresy średnich przebiegów mogłyby być uśrednieniem większej liczby przebiegów.
- Dodanie operatora krzyżowania do algorytmu *EO*.
- Dodanie innego operatora mutacji do algorytmu *EO*, np. opartego o dodawanie pojedynczych cykli odpowiedniej długości, a nie dowolnych permutacji, jak analizowano w tym raporcie.

Literatura

[1] Piotr Graczyk, Hideyuki Ishi, Bartosz Kołodziejek, and Hélène Massam. Model selection in the space of gaussian models invariant by symmetry. *arXiv*, 04 2020. `https://www.researchgate.net/publication/340500495_Model_selection_in_the_space_of_Gaussian_models_invariant_by_symmetry`.

[2] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36:114–144, 2021.

[3] Chojecki Przemysław. Strona GitHub z kodem pakietu 'gips'. `https://github.com/PrzeChoj/gips/`. [Online; accessed 15-06-2022].

[4] Chojecki Przemysław. Strona GitHub z kodem tego projektu. `https://github.com/PrzeChoj/WAE/`. [Online; accessed 15-06-2022].

[5] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020.

[6] Wikipedyści. Strona Wikipedia dla testu t-studenta. `https://en.wikipedia.org/wiki/Student%27s_t-test`. [Online; accessed 15-06-2022].

[7] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[8] Frank Wilcoxon. Some rapid approximate statistical procedures. american cyanamid co. *Pearl River, NY, USA*, 1964.

A Dodatek - Opis rodziny funkcji celu

Z punktu widzenia głównej części dokumentacji wstępnej, funkcja celu jest czarną skrzynką. W tym dodatku opisano, skąd wzięła się rodzina rozważanych funkcji celu. Krótko wytłumaczono, czemu znalezienie argumentu z jak największą wartością tej funkcji jest istotne dla zadania uczenia maszynowego.

Skąd się wzięła funkcja celu?

Zakładamy, że wektory losowe są kolumnowe.

Załóżmy, że $Z \sim \mathcal{N}_p(0, \Sigma)$ jest wektorem losowym długości p o rozkładzie normalnym. Załóżmy, że $(Z_1, Z_2, ..., Z_n)$ jest ciągiem n zmiennych losowych iid o takim samym rozkładzie jak Z.

Jeśli $n \geqslant p$, to istnieje estymator największej wiarogodności Σ i wynosi on:

$$U = \frac{1}{n} \sum_{i=1}^{n} Z_i \cdot Z_i^T$$

Jeśli jednak $n < p$, to nie istnieje estymator największej wiarogodności Σ. Innymi słowy, nie da się sensownie estymować macierzy kowariancji.

Można jednak założyć coś więcej o macierzy Σ. Jeśli założone zostanie na przykład, że przenumerowanie wierszy i kolumn $1 \to 2, 2 \to 3, ..., p-1 \to p, p \to 1$ nie zmienia macierzy, to można ją estymować, posiadając tylko jedną obserwację, $n = 1$. Widać jednak, że jest to bardzo mocne założenie.

Jeśli przenumerowanie wierszy i kolumn zgodnie z permutacją σ nie zmienia macierzy Σ, to znaczy $\Sigma = \sigma^{-1} \cdot \Sigma \cdot \sigma$, to mówimy, że „$\Sigma$ jest niezmiennicza względem permutacji σ".

W wielu praktycznych zadaniach modelowania można założyć niezmienniczość macierzy kowariancji względem jakiejś permutacji. Na przykład, jeśli pomiary dwóch kolumn wykonywane były tym samym urządzeniem. Trudno jest jednak ekspercko zdecydować o tego typu założeniach.

Autorzy artykułu [1] podali pod numerem (41) wzór na funkcję, która jest proporcjonalna do prawdopodobieństwa a posteriori, że dana obserwacja została wygenerowana przez rozkład niezmienniczy względem permutacji c.

Jeśli dla konkretnej obserwacji $(Z_1, ..., Z_n)$ znalezione zostanie c, dla którego prawdopodobieństwo to będzie największe spośród wszystkich permutacji, to znalezione c będzie estymatorem największej wiarogodności prawdziwej permutacji, z której pochodzi dany rozkład. Sensownie więc byłoby założyć, że obserwacje te pochodzą z tego rozkładu i można dzięki temu założeniu estymować macierz Σ nawet w sytuacjach, gdy $n < p$.