1 Wprowadzenie

W licznej grupie algorytmów populacyjnych, w ostatnim czasie coraz większą rolę odgrywać zaczynają algorytmy wykorzystujące modele probabilistyczne.

Są to najczęściej metody o strukturze bardzo podobnej do struktury algorytmu ewolucyjnego, z tą różnicą, że kolejne pokolenia osobników (tzn. propozycji rozwiązań analizowanego problemu) generowane są na bazie modelu probabilistycznego, tzw. populacji rozwiązań obiecujących, nie zaś jako efekt krzyżowania bądź mutacji osobników z populacji bieżącej.

Populacja rozwiązań obiecujących powstaje z osobników wyłonionych w wyniku klasycznej selekcji (zwykle turniejowej). W populacji takiej pojawiają się osobniki o wyższym od średniego przystosowaniu, a zbudowany na ich podstawie model powinien promować te cechy rozwiązania, które prowadzą do optymalizowanego celu.

Kolejne pokolenie rozwiązań generowane jest w sposób pseudolosowy, ale z uwzględnieniem modelu probabilistycznego. Oznacza to, że w metodach tego typu sposób budowania modelu odpowiada zarówno za samą zbieżność do ekstremum, jak i jej tempo.

Aby w pełni wykorzystać cechy omawianych metod, należy zadbać o taki sposób budowy modelu probabilistycznego, aby przy efektywnej zbieżności nie utracić możliwości właściwego przeszukiwania przestrzeni. Jeśli populacja zbyt mocno będzie wpływa na zmiany modelu w kolejnych iteracjach, to może prowadzić to do szybkiego ujednolicania populacji i niewłaściwej eksploracji przestrzeni. Z drugiej strony, zbyt powolna zmiana modelu będzie sprawiała, że metoda optymalizacyjna w swoim działaniu przypominała będzie przeszukiwanie losowe.

To w jaki sposób budowany będzie model jest kluczowe z punktu widzenia tego typu metod. Pozostałe elementy algorytmu, takie jak np. sukcesja, mają zwykle klasyczną formę (znaną z algorytmów genetycznych) i służą do prowadzenia procesu iteracyjnego.

W prezentowanej pracy przedstawione zostaną dwie metody optymalizacyjne wykorzystujące model probabilistyczny. W metodach tych zakłada się, że przeszukiwaną przestrzenią jest zbiór ciągów binarnych.

Model probabilistyczny będzie odpowiedzialny za prawdopodobieństwo pojawienia się zer lub jedynek w poszczególnych miejscach ciągu.

Obie metody testowane będą na funkcjach, których optimum poszukuje się w zbiorze ciągów binarnych.

2 Przegląd literatury

W pracy zaprezentowane zostaną dwie wersje algorytmów z modelem probabilistycznym *PBIL* (*ang. Population-based incremental learning*) oraz cGA (*ang. Compact Genetic Algorithm*). Obie metody są heurystykami populacyjnymi, które rozważają populację w procesie iteracyjnym.

Kuo, Glover i Dhir w swoim artykule [**?**] sformułowali problem *Max Diversity*, jednak nie rozważali jego optymalizacji w sposób algorytmiczny. Ich rozważania w artykule [**?**] podjęli Gallego, Duarte, Laguna oraz Martí, szukając rozważania przybliżonego przy użyciu *scatter search procedure*.

Martin Pelikan w jednym ze swoich artykułów [**?**] rozważa między innymi *cGA* oraz *PBIL*, które zostały opisane w niniejszej pracy. Każdy z opisanych algorytmów został poddany testom przez optymalizację problemów *trap_n* oraz *3-deceptive*. Głównym celem jego pracy było porównanie rezultatów optymalizacji.

3 PBIL

Pierwszą z prezentowanych w pracy metod jest algorytm wykorzystujący proces uczenia oparty na „obserwacji" populacji bieżącej, w skrócie *PBIL* (*ang. Population-based incremental learning*).

W metodzie tej osobniki należące do kolejnych populacji/pokoleń tworzone są na podstawie wektora $\mathbf{p} = [p_1, p_2, \ldots, p_m]$, którego składowe p_i określają prawdopodobieństwo wystąpienia jedynki na i-tej pozycji generowanego osobnika. Wektor ten pełni rolę modelu probabilistycznego.

Charakterystyczne dla algorytmu *PBIL* jest wykorzystanie do uaktualnienia wektora $\mathbf{p}$ wyłącznie najlepszego osobnika w pokoleniu bieżącym. Oznacza to, że model taki powstaje w oparciu o jednego, najbardziej obiecującego osobnika, oznaczanego $\mathbf{b}$.

Na początku procesu przyjmuje się, że składowe wektora $\mathbf{p}$ mają jednakową wartość, równą $\frac{1}{2}$. Generuje się także populację startową (z rozkładem równomiernym, czyli zgodnie z modelem reprezentowanym przez $\mathbf{p}$) składającą się z ciągów 0–1 długości k.

W kolejnych iteracjach, składowe wektora $\mathbf{p}$ uaktualnia się według wzoru:

$$p_i^{(k+1)} = (1 - \lambda) \cdot p_i^{(k)} + \lambda b_i, \tag{1}$$

gdzie $p_i^{(k)}$ to i-ta składowa wektora $\mathbf{p}$ w pokoleniu k, b_i – składowa bieżącego wektora $\mathbf{b}$, a λ– tzw. współczynnik uczenia.

Osobniki populacji $k+1$ losowane są zawsze z uwzględnieniem aktualnego wektora prawdopodobieństw. W przeciwieństwie do standardowego algorytmu genetycznego, *PBIL* nie zachowuje najlepszego osobnika w populacji, jednak specyfika procedury daje ogromne szanse na jego wylosowanie, gdyż właśnie na jego podstawie modyfikowany jest model probabilistyczny. Losowanie całej populacji, uwzględniające model (reprezentowany przez $\mathbf{p}$)[!Powtórzone w linii 54] daje spore szanse na pojawienie się większej liczby „dobrych" (z punktu widzenia funkcji celu) osobników, zwykle lepszych niż w poprzedniej generacji.

Wartość współczynnika uczenia λ jest parametrem ustalanym na początku procesu iteracyjnego i ma wpływ na jego przebieg. Należy pamiętać, że mała jego wartość spowalnia modyfikację modelu, a zbyt duża może wpływać na zbyt szybkie ujednolicenie populacji. Współczynnik λ powinien być dobrany tak, aby równoważyć zdolność do ukierunkowanej eksploracji z możliwością eksploatacji przestrzeni.

Poniżej przedstawiono schemat metody PBIL.

---

procedure *PBIL:*

1. *Losowanie populacji startowej, zainicjowanie wektora prawdopodobieństw* $\mathbf{p}$, $(p_i = 0.5, \forall i = 1, \ldots, n)$
2. *Ocena osobników, wybór najlepszego wektora (oznaczonego jako* $\boldsymbol{b}$*).*
3. *Modyfikacja składowych wektora prawdopodobieństw* $\mathbf{p}$ *według wzoru*

   $$p_i = (1 - \lambda) \cdot p_i + \lambda \cdot b_i, \; i = 1, 2, \ldots k$$

   gdzie λ *– współczynnik uczenia.*
4. *Wylosowanie nowej populacji zgodnie z modelem z uwzględnieniem aktualnego wektora* $\boldsymbol{p}$
5. *Sprawdzenie warunku zatrzymania, jeśli spełniony – zakończenie algorytmu, w przeciwnym razie powrót do 2.*

---

4 cGA

Kolejnym omawianym w pracy algorytmem jest cGA (*ang. Compact Genetic Algorithm*), metoda będąca modyfikacją AG i wykorzystująca model probabilistyczny.

Podobnie jak w *PBIL*, kolejne pokolenia osobników tworzone są w oparciu o model probabilistyczny. Model budowany jest na bazie o rozwiązania z pokolenia bieżącego, przy czym w jego konstrukcji uwzględniane są zarówno najlepszy, jak i najgorszy osobnik. Rolę modelu ponownie pełni wektor prawdopodobieństw $\mathbf{p}$, którego składowe aktualizowane są według wzoru:

$$p_i = \begin{cases} p_i + \frac{1}{m}, & x_i = 1 \wedge y_i = 0 \\ p_i - \frac{1}{m}, & x_i = 0 \wedge y_i = 1 \quad i = 1, 2, \ldots k \\ p_i, & \text{pozostałe} \end{cases} \tag{2}$$

gdzie $\mathbf{x} = [x_1, \ldots, x_k]$ i $\mathbf{y} = [y_1, \ldots, y_k]$ to odpowiednio najlepszy i najgorszy osobnik w populacji, zaś m to liczebność populacji.

Tak jak w poprzedniej metodzie, składowe wektora $\mathbf{p}$ określają prawdopodobieństwo występowania 1 na i-tym miejscu osobnika generowanego do kolejnej populacji. Współczynnik $\frac{1}{m}$ pełni we wzorze (**??**) rolę współczynnika uczenia się i może być zastąpiony dowolną inną, ustaloną wielkością. Dzięki wykorzystaniu najlepszego i najgorszego osobnika z pokolenia bieżącego, algorytm *cGA* pozwala efektywniej tworzyć model probabilistyczny. Procedura

budowy wektora $\mathbf{p}$ sprawia, że szansa wylosowania osobnika zbliżonego do najlepszego rośnie, a najgorszego maleje.

Schemat algorytmu zaprezentowano poniżej w formie pseudokodu.

---

procedure *cGA*

1. *Losowanie populacji startowej, zainicjowanie wektora prawdopodobieństw* $\boldsymbol{p}$ *(wszystkie wartości* $p_i = 0.5, i = 1, \ldots, n$*).*
2. *Ocena osobników, wybór najlepszego i najgorszego.*
3. *Zaktualizowanie wektora prawdopodobieństw według wzoru*

$$p_i = \begin{cases} p_i + \frac{1}{m}, & x_i = 1 \wedge y_i = 0 \\ p_i - \frac{1}{m}, & x_i = 0 \wedge y_i = 1 \quad i = 1, 2, \ldots k. \\ p_i, & \textit{pozostałe} \end{cases}$$

4. *Wygenerowanie nowej populacji z uwzględnieniem prawdopodobieństw* $\boldsymbol{p}$
5. *Sprawdzenie warunku zatrzymania, jeśli spełniony – zakończenie algorytmu, w przeciwnym razie powrót do punktu 2.*

---

5 Opis zadań testowych

W ramach prezentowanej pracy, optymalizacji poddano trzy funkcje testowe. We wszystkich przypadkach przyjmowano, że poszukiwane rozwiązanie jest $k-$elementowym ciągiem binarnym. Założenie to było wymuszone specyfiką omawianych metod optymalizacyjnych.

Każda z testowanych funkcji miała odmienny charakter, aby możliwe było jak najlepsze rozpoznanie zalet i wad prezentowanych heurystyk.

5.1 $trap_n$

Pierwszą funkcją testową była funkcja $trap_n$ dana wzorem:

$$f_{trap_n}(\mathbf{u}) = \begin{cases} n - 1 - u_1, & \text{dla } u_1 < n \\ n, & \text{w pozostałych przypadkach,} \end{cases} \tag{3}$$

gdzie n oznacza rząd funkcji, a u_1 to liczba jedynek występujących z wektorze $\mathbf{u}$.

Zwykle przyjmuje się, że rząd funkcji $trap_n$ jest taki sam jak wymiar zadania, tzn. $n = k$.

Przykładowo, funkcja $trap_5$ wyraża się wzorem:

$$f_{trap_5}(\mathbf{u}) = \begin{cases} 4 - u_1, & \text{dla } u_1 < 5 \\ 5, & \text{w pozostałych przypadkach} \end{cases}$$

i osiąga swoje maksimum globalne o wartości 5 dla $\mathbf{u}_{opt} = (1, 1, 1, 1, 1)$.

W odróżnieniu od klasycznie wykorzystywanej do testów funkcji OneMax, wartości funkcji $trap_n$ nie zależą liniowo od liczby jedynek w wektorze $\mathbf{u}$, co może być dodatkową trudnością w optymalizacji (rys.**??**).

Rysunek 1: Wykres funkcji $trap_5$

5.2 $3 - deceptive$

Drugą testowaną w ramach pracy funkcją była funkcja $3 - deceptive$ zadana wzorem:

$$f_{3deceptive}(\mathbf{u}) = \begin{cases} 0.9, & \text{dla } u_1 = 0 \\ 0.8, & \text{dla } u_1 = 1 \\ 0, & \text{dla } u_1 = 2 \\ 1, & \text{w pozostałych przypadkach} \end{cases} . \tag{4}$$

Podobnie jak w poprzednim przypadku u_1 oznacza liczbę składowych wektora $\mathbf{u}$, które przyjmują wartość 1.

Jest to funkcja posiadająca jedno minimum globalne oraz dwa, niewiele różniące się co do wartości, maxima lokalne. Dodatkowo, jeśli tylko długość k wektora $\mathbf{u}$ jest większa od 3, maximum globalne o wartości 1 osiągane jest w wielu punktach. Sytuacja taka utrudnia optymalizację, gdyż nawet znacznie różniące się rozwiązania mają dokładnie taką samą jakość.

Rysunek 2: Wykres funkcji $3deceptive$

5.3 $MaxDiversity$

Ostatnią i równocześnie najciekawszą funkcją wykorzystaną do testów była funkcja $MaxDiversity$. W tym przypadku optymalizacja polega na znalezieniu w $k-$elementowym zbiorze X (elementy zbioru należy dowolnie ponumerować), $m-$elementowego podzbioru A, do którego należą punkty, których suma wzajemnych odległości jest największa. Danymi wejściowymi są tutaj zbiór punktów X oraz m czyli liczba punktów, z których złożony ma być szukany podzbiór. W zadaniu $MaxDiversity$ rozwiązania poszukuje się w postaci wektora o k składowych, z których m ma wartość jeden (jedynka na pozycji i oznacza, że i-ty punkt zbioru X należy do A).

Przykładowo, przyjmując za X zbiór wierzchołków kwadratu jednostkowego i szukając 2-elementowego podzbioru A spełniającego powyższe założenia, otrzymać powinno się parę przeciwległych wierzchołków kwadratu, których wzajemna odległość wynosi $\sqrt{2}$. Tak postawione zadanie ma oczywiście dwa równoważne rozwiązania optymalne.

Funkcja MD ($MaxDiversity$), funkcja celu której maximum poszukujemy, przyjmuje postać:

$$\mathrm{MD}(A, X) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d(\mathbf{x}_i, \mathbf{x}_j),$$

gdzie $A = \{\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_m\}$ to m-elementowy podzbiór zbioru X, a $d(x_i, x_j)$ to odległość pomiędzy punktami $\mathbf{x}_i$ oraz $\mathbf{x}_j$ należącymi do zbioru X.

Na potrzeby pracy przyjęto standardową definicję odległości – odległość Euklidesową:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2},$$

gdzie $\mathbf{x} = (x_1, x_2, \dots, x_n)$, $\mathbf{y} = (y_1, y_2, \dots, y_n)$, ($n$ - wymiar przestrzeni X).

6 Wyniki testów

Celem testów numerycznych była ocena metod PBIL oraz cGA, ich porównanie oraz ewentualnie znalezienie sposobu doboru parametrów.

Do rozwiązania każdego z zadań testowych wykorzystano obie metody, przy czy metodę PBIL testowano dodatkowo dla różnych współczynników uczenia. Zmieniana była także liczebność populacji.

W pojedyńczym teście wykonywano 100 eksperymentów, przy czym eksperyment rozumie się jako pojedyńczą pętlę iteracyjną prowadzoną do uzyskania rozwiązania dokładnego, ale nie dłużej niż przez 100 iteracji. Za wynik testu przyjmowano średnią (ze 100 eksperymentów) liczbę iteracji koniecznych do uzyskania rozwiązania optymalnego.

6.1 Funkcja $Trap_n$

W ramach testów, poszukiwano maximum funkcji $trap_n$ dla różnych wartości n. W ogólnym przypadku maksimum globalne funkcji $trap_n$ wynosi n i jest osiągane w $\mathbf{u}^{max} = \underbrace{(1, 1, \ldots, 1)}_{n}$, tzn.

$$f^{max}_{trap_n} \underbrace{(1, 1, \ldots, 1)}_{n} = n.$$

Uzyskane rezultaty zamieszczono w tabelach poniżej.

W nielicznych przypadkach, liczba 100 kroków iteracyjnych nie wystarczyła do wyznaczenia maksimum globalnego. Wówczas w tabeli zamieszczono dodatkowo (w nawiasie) informacje o średnim błędzie rozwiązania w 100 doświadczeniach.

Wspomniany błąd występował, gdy po przeprowadzeniu 100 kroków nie została osiągnięta największa znana wartość. Jako błąd przyjmowana była odległość od najlepszego wyniku znalezionego w dotychczasowych eksperymentach. W przypadku osiągnięcia ekstremum w mniejszej niż 100 liczbie kroków, wartość błędu przyjmowana była jako 0. W tabelach wyróżniono także konfiguracje, dla których przy ustalonej liczebności populacji algorytm znalazł dokładne rozwiązanie w (średnio) najmniejszej liczbie iteracji.

Tabela 1: Wyniki testów - funkcja $trap_5$

	Ilość iteracji (błąd)					
	cGA	*PBIL*				
Liczebność populacji		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	**2.97**	24.62(0.58)	6.03(0.04)	3.94	5.94	6.77
20	1.77	1.63	**1.59**	1.88	2.02	2.6
50	**1.16**	1.2	1.18	1.21	1.32	1.21

Tabela 2: Wyniki testów - funkcja $trap_6$

Liczebność populacji	*Ilość iteracji (błąd)*					
	cGA	*PBIL*				
		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	**3.55**	47.91(1.29)	13.76(0.3)	6.76(0.03)	10.6	12.46(0.01)
20	2.65	3.84(0.06)	**2.26**	2.6	3.05	3.44
50	1.71	**1.42**	1.46	1.53	1.63	1.71
100	1.24	1.22	1.25	**1.15**	1.19	1.19

Tabela 3: Wyniki testów - funkcja $trap_7$

Liczebność populacji	*Ilość iteracji (błąd)*					
	cGA	*PBIL*				
		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	**4.86**	63.61(1.84)	34.72(0.92)	19.53(0.39)	14.99	25.91(0.02)
20	**3.08**	15.71(0.41)	4.(0.03)	3.76	5.61	7.89
50	2.42	2.81(0.03)	**2.06**	2.18	2.59	3.08
100	1.61	1.43	**1.42**	1.44	1.65	1.76
200	1.22	**1.16**	1.26	1.19	1.26	1.2

Tabela 4: Wyniki testów - funkcja $trap_{10}$

Liczebność populacji	*Ilość iteracji (błąd)*					
	cGA	*PBIL*				
		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	**5.42**	141.76(3.6)	71.36(2.59)	56.78(1.8)	42.21(0.06)	78.45(0.97)
20	**6.18**	84.58(1.97)	30.72(0.92)	12.78(0.18)	20.21	40.79(0.13)
50	6.27	37.64(0.77)	7.78(0.12)	**4.79**	12.49	21.14(0.01)
100	5.76	7.94(0.12)	**2.86**	3.85	7.42	10.82
200	4.78	4.78(0.06)	**2.22**	2.79	4.7	4.88
500	2.49	1.79	**1.66**	1.69	2.16	2.28

Podsumowanie: Z przeprowadzonych testów wynika, że w przypadku funkcji $trap_n$ metoda *cGA* jest bardziej efektywna od metody *PBIL*.

Niezależnie od rzędu funkcji n, compact Genetic Algorithm dobrze sobie radzi z optymalizacją, nawet przy wykorzystaniu mało licznych populacji. O właściwym działaniu świadczy dodatkowo fakt, że wraz ze wzrostem wielkości populacji maleje liczba iteracji koniecznych do uzyskania poprawnego wyniku.

Wykorzystanie metody PBIL jest już bardziej kłopotliwe. W tym wypadku działanie algorytmu wiąże się dodatkowo z koniecznością doboru współczynnika uczenia λ. Jak wynika z powyższych zestawień, niewłaściwy dobór tego parametru może negatywnie wpłynąć na wydajność metody. Testy zdają się wskazywać, że najbezpieczniejszą wartością λ jest 0.01, gdyż dla tej wartości ryzyko uzyskania niedokładnego wyniku było najmniejsze.

6.2 Funkcja 3 – *deceptive*

Funkcja *3-deceptive* osiąga maksimum globalne o wartości 1, jeśli co najmniej 3 ze składowych wektora **u** są jedynkami.

W pracy rozważono 3 warianty funkcji, określonej ogólnym wzorem (**??**):

W wariancie pierwszym, rozwiązania poszukiwano w zbiorze wektorów długości 3, co implikuje istnienie dokładnie jednego ekstremum globalnego. W pozostałych przypadkach funkcja *3-deceptive* przyjmuje optimum w kilku różnych punktach przestrzeni.

Dokładniej mówiąc, dla wektorów n-elementowych,

$$f^{max}_{3deceptive}(\mathbf{u}) = 1,$$

dla każdego $\mathbf{u} \in A$, gdzie A–zbiór wektorów z co najmniej trzema jedynkami.

Interpretacja wyników przedstawionych w tabelach jest analogiczna jak w podrozdziale **??**.

Wariant 1 - przestrzeń wektorów długości 3

Tabela 5: Wyniki testów – funkcja 3 – *deceptive*; przestrzeń wektorów dł.3

	Ilość iteracji (błąd)					
	cGA	*PBIL*				
Liczebność populacji		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	1.8	6.26 (0.01*)	**1.36**	1.84	1.77	1.89
20	**0.92**	1.01	0.97	0.93	1.07	0.94
50	0.87	0.87	**0.84**	0.86	0.88	0.88
100	0.89	0.84	0.86	0.88	0.91	**0.83**
200	0.87	0.86	**0.85**	**0.85**	0.86	0.89
500	**0.82**	0.91	0.86	**0.82**	0.93	0.84

* – algorytm *PBIL* dla optymalizowanej funkcji, $\lambda = 0.5$ i bardzo małej populacji (5 osobników) zwrócił niedokładną wartość.

Podsumowanie Optymalizacja funkcji 3 – *deceptive* w wariancie 1 dla obu metod przebiegała bez większych problemów. W znakomitej większości przypadków wyniki uzyskiwano po zaledwie kilku krokach. Jest to z pewnością konsekwencja specyfiki przeszukiwanej przestrzeni (wektory o trzech składowych) i związanej z tym dużej szansy wylosowania rozwiązania optymalnego lub bliskiego optymalnemu już w populacji startowej.

Jedynym testem, w którym optymalizacja przebiegała mniej sprawnie był przypadek w którym wykorzystano PBIL ze stosunkowo dużym współczynnikiem uczenia ($\lambda = 0.5$) i mało liczną populację. Wynika to z pewnością z ujednolicenia populacji wynikającej ze zbyt szybkiego zdominowania populacji przez jednego osobnika, spowodowanego dużą wartością λ.

W przypadku tak sformułowanego zadania, nie ma znacznej różnicy między działaniem algorytmów *cGA* a *PBIL*.

Wariant 2 - przestrzeń wektorów długości 5

Podsumowanie: Podobnie jak w poprzednim wariancie, w testach uzyskano porównywalnie dobre rezultaty w przypadku stosowania obu algorytmów. Nie zaobserwowano nawet wpływu wielkości współczynnika uczenia się na uzyskanie wyniku. W wielu eksperymentach na dokładny/optymalny wynik trafiono w iteracji zerowej. W obliczeniach nie rozróżniano różnych rozwiązań, dla których funkcja celu przyjmuje taką samą wartość.

Wektory 10-cio elementowe

Podsumowanie: Metoda działa lepiej im dłuży jest wektor i populacja jest bardziej liczna. Błąd występuje w pojedynczym przypadku, gdy badamy małą populację i przyjmiemy w algorytmie stosunkowo wysoki współczynnik

Tabela 6: Wyniki testów - funkcja 3 – *deceptive*; przestrzeń wektorów dł.5

	Ilość iteracji (błąd)					
	CGA	*PBIL*				
Liczebność populacji		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	**0.41**	0.61	0.47	0.53	0.55	0.57
20	0.51	0.43	**0.40**	**0.40**	0.49	0.61
50	0.53	0.58	**0.47**	0.48	0.59	**0.47**
100	0.50	0.52	0.5	0.48	**0.43**	0.48
200	**0.48**	0.61	0.51	0.54	0.5	0.54
500	0.52	0.47	**0.46**	0.58	0.55	0.5

Tabela 7: Wyniki testów - funkcja 3 – *deceptive*; przestrzeń wektorów dł.10

	Ilość iteracji (błąd)					
	CGA	*PBIL*				
Liczebność populacji		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	0.11	**0.02**	0.07	0.05	0.1	0.04
20	0.06	**0.03**	0.05	0.06	0.07	0.05
50	0.06	0.05	**0.02**	0.08	**0.02**	0.07
100	0.13	0.07	0.02	**0.0**	0.03	0.1
200	0.05	0.06	**0.04**	0.07	0.08	0.02
500	0.04	**0.03**	0.06	0.06	0.04	0.04

uczenia się ($\lambda = 0.5$). Dla problemu *3-deceptive* i wektora n-elementowego wystarczy wybrać populację 20 elementową i dowolny współczynnik uczenia, gdyż dla każdego z przyjętych parametrów czas wyznaczania maksimum nie przekracza jednej iteracji zarówno dla *cGA*, jak i *PBIL*. Dla wektorów długości powyżej 10, liczba iteracji jest bliska zeru, ponieważ prawdopodobieństwo wylosowania na starcie wektora z co najmniej trzema jedynkami jest bardzo duże i rośnie wraz ze wzrostem długości wektora. Moc zbioru rozwiązań A dla wektora długości n wynosi bowiem

$$\|A\| = \sum_{i=3}^{n} \binom{n}{i},$$

co daje prawdopodobienstwo wylosowania ekstremum równe

$$P = \frac{\sum_{i=3}^{n} \binom{n}{i}}{2^n} \underset{n\to\infty}{\longrightarrow} = 1$$

6.3 Funkcja *MaxDiversity*

Minimalnie przeredagowane wprowadzenie

W przypadku funkcji *MaxDiversity*, zadanie polega na znalezieniu m-elementowego podzbioru A danego zbioru X (o mocy k). Rozwiązanie reprezentowne jest przez k-elementowy wektor binarny, przy czym jedynka na i-tej pozycji oznacza, że i-ty element zbioru X należy do podzbioru A.

Podobnie jak w poprzednich testach, za kryterium zatrzymania algorytmu przyjęto znalezienie rozwiązania optymalnego, bądź wykonanie maksymalnej dopuszczalnej liczby iteracji (wówczas za rozwiązanie przyjmuje się najlepszy uzyskany wynik).

Ze względu na odmienną naturę problemu, w tabelach (poza średnią liczbą iteracji niezbędnych do rozwiązania) zamieszczono dodatkowo błąd rozwiązania w skali procentowej. Wielkość tą należy rozumieć jako odsetek poprawnie wyznaczonych rozwiązań w 100 doświadczeniach.

6.3.1 X – zbiór wierzchołków kwadratu jednostkowego (w przestrzeni 2-wymiarowej)

W zadaniu szukamy n-elementowego podzbioru A wierzchołków kwadratu $X = \{(0,0),(0,1),(1,0),(1,1)\}$, tak, aby suma odległość pomiędzy punktami należącymi do A była największa.

W ramach testu rozwiązywano zadanie w którym szukano pary punktów ($m = 2$) ze zbioru X, których odległość jest maksymalna.

W tym wypadku rozwiązanie optymalne to zbiór **A** taki, że

$$\mathrm{MD}^{max}(A, X) = \sqrt{2},$$

gdzie $A = \{(0,0), (1,1)\}$ lub $A = \{(1,0),(0,1)\}$. Rozwiązania te reprezentowane są odpowiednio przez wektory $\mathbf{u} = (1,0,0,1)$ oraz $\mathbf{u} = (0,1,1,0)$.

Tabela 8: Wyniki testów na *MaxDiversity* dla wierzchołków kwadratu ($m = 2$)

Liczebność populacji	*Ilość iteracji*									
	CGA	*PBIL*								
		$\lambda = 0.5$		$\lambda = 0.2$		$\lambda = 0.1$		$\lambda = 0.01$		$\lambda = 0.0001$	
3	**2.03**	2.26	*95%*	2.14	*99%*	2.09	*99%*	2.00	*98%*	2.18	*95%*
5	1.89	1.96	*99%*	1.88	*100%*	**1.82**	*100%*	1.84	*100%*	**1.82**	*100%*
20	**1.62**	1.73	*100%*	1.69	*100%*	1.64	*100%*	1.66	*100%*	1.72	*100%*
50	1.68	**1.62**	*100%*	1.65	*100%*	1.73	*100%*	1.7	*100%*	1.7	*100%*
100	1.68	**1.60**	*100%*	1.62	*100%*	1.72	*100%*	1.66	*100%*	1.61	*100%*

Podsumowanie: W przypadku tego zadania oba algorytmy zadziałały poprawnie bez problemu znajdując jeden ze zbiorów, będąc rozwiązaniem dokładnym. Dobór parametrów algorytmów, tj. liczebności populacji i współczynnika uczenia się, nie miał większego znaczenia, choć zgodnie z oczekiwaniami bardzo niska liczebność populacji nie gwarantowała znalezienia rozwiązania.

6.3.2 X - zbiór 10 losowo wybranych punktów w kuli jednostkowej o środku w punkcie (0,0)

Rysunek 3: Kula jednostkowa z punktami zbioru X

W zadaniu przyjęto, że $X = \{(0.5, -0.5), (0.4, 0.1), (-0.9, -0.1), (0.1, 0.12), (-0.32, 0.14), (-0.1, 0.58), (0.911, 0.2), (-0.77, 0.58), (0.14, -0.85), (-0.14, -0.13)\}$ (rys.**??**).
Zadanie polegało na znalezieniu w zbiorze X 3-elementowego podzbioru A.

Dla tak sformułowanego problemu rozwiązanie dokładne to $A = \{(0.911, 0.2), (-0.77, 0.58), (0.14, -0.85)\}$, dla którego $MD^{max}(X, A) \simeq 4.721$ (rys. ???). Rozwiązanie to reprezentowane jest przez wektor $\mathbf{u}^{max} = (0, 0, 0, 0, 0, 0, 1, 1, 1, 0)$.

Rysunek 4: Maksymalna odległość

W tabeli poniżej zaprezentowano wyniki uzyskane w eksperymentach numerycznych:

Tabela 9: Wyniki testów na $MaxDiversity$ dla punktów kuli ($m = 3$)

Liczebność populacji	*Ilość iteracji*											
	cGA		*PBIL*									
			$\lambda = 0.5$		$\lambda = 0.2$		$\lambda = 0.1$		$\lambda = 0.01$		$\lambda = 0.0001$	
3	28.5	*77%*	77.41	*24%*	39.07	*68 %*	23.26	*92%*	**27.99**	*99%*	33.55	*93%*
5	11.78	*94%*	65.25	*40%*	26.57	*81%*	**14.25**	*100%*	17.31	*100%*	25.59	*98%*
20	5.51	*100%*	12.43	*91 %*	**4.55**	*100%*	5.5	*100%*	7.17	*100%*	6.63	*100%*
50	3.72	*100%*	**2.84**	*100%*	3.2	*100%*	3.37	*100%*	4.12	*100%*	4.12	*100%*
100	2.79	*100%*	2.54	*100%*	**2.46**	*100%*	2.68	*100%*	2.67	*100%*	2.91	*100%*

Podsumowanie: Optymalizacja ostatniej funkcji testowej, zarówno przy zastosowaniu cGa jak i PBIL przebiegała w zadowalający sposób. Przy odpowiednio licznej populacji, za każdym razem udało się znaleźć ekstremum globalne. Mniejsza liczba osobników wymaga niższych wartości współczynnika uczenia się, w przypadku stosowania PBIL. Warto pamiętać, że niekorzystna konfiguracja parametrów obniża wiarygodność wyniku.

Porównując czas pracy algorytmów, należy pamiętać, że *cGA* zwraca rezultat w krótszym czasie niż *PBIL*. Zarówno wysoki, jak i bardzo niski wskaźnik uczenia się algorytmu *PBIL* zmniejsza efektywność pracy algorytmu. Z testów wynika, że najbardziej optymalną wartością współczynnika uczenia się jest 0.1. Wówczas rezultaty uzyskane przy stosowaniu *cGA* i *PBIL* są zbliżone.

Literatura

[1] C.-C. Kuo, F. Glover, and K.S. Dhir. Analyzing and modeling the maximum diversity problem by zero-one programming. *Decission Sciences*, page 1171–1185, November 1993.

[2] M.Gallego, A. Duarte, M. Laguna, and R. Marti. Hybrid heuristics for the maximum diversity problem. *Springer Science+Business Media*, 44:411–426, December 2007.

[3] M. Pelikan. *Bayesian optimization algorithm: From single level to hierarchy.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, 2002.