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/rozwiązań generowane są na bazie modelu probabilistycznego 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, 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 GA) i służą do prowadzenie procesu iteracyjnego.

W prezentowanej pracy przedstawione zostaną dwie metody optymalizacyjne wykorzystujące model probabilistyczny. Są to metody w których zakłada się, że przeszukiwaną przestrzenią jest zbiór ciągów binarnych.

Model probabilistyczny będzie odpowiedzialny za to z jakim prawdopodobieństwem pojawiać mają się w takich ciągach zera lub jedynki.

Obie metody testowane będą na funkcjach, których optimum poszukuje się w zbiorze ciągów binarnych.
In this article we consider the binary strings set.

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.

There will be presented two variations of probabilistic based algorithms *PBIL* (*Population-based incremental learning*)and cGA (*Compact Genetic Algorithm*). Both methods are population-based heuristics, that consider the population in iterative process.

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, tzw. *PBIL* (*ang. Population-based incremental learning*).

First of the presented method is the algorithm using incremetal learning based on current population observation, so called *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.
In this method individuals for the next generation are generated using the following vector $\mathbf{p} = [p_1, p_2, \ldots, p_m]$, which coordinates p_i determine probability of occurrence ones at the i-th position the new individual. That vector

acts as a probabilistic model.

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 $\mathbf{b}$.

The characteristic element of *PBIL* algorithm is upgrading the vector $\mathbf{p}$ based on a fittest solution. It means that such model is generated by one, the most promising solution denoted by $\mathbf{b}$.

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) składającą się z ciągów 0–1 o długości k.

The searching procedure starts with equal coordinates of the vector $\mathbf{p}$ equals $\frac{1}{2}$. There is also generated a starting population consisting of binary strings with length 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. In the next iterations, vector $\mathbf{p}$ coordinates are updated using the formula:

$$p_i^{(k+1)} = (1 - \lambda) \cdot p_i^{(k)} + \lambda b_i, \tag{2}$$

where $p_i^{(k)}$ is a i-th coordinate of vector $\mathbf{p}$ in k-th generation, b_i – i-th coordinate of current $\mathbf{b}$ and λ– learning rate. 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, ale specyfika procedury daje ogromne szanse na jego wylosowanie, gdyż właśnie na jego podstawie modyfikowany jest model probabilistyczny. Losowanie całej populacji, uwzględniające model (reprezentowany przez $\mathbf{p}$) daje spore szanse na pojawienie się większej liczby „dobrych" (z punktu widzenia funkcji celu) osobników, zwykle lepszych niż w poprzedniej generacji.

The k+1-th population individuals are randomly selected based on weight values in the current vector $\mathbf{p}$. Unlike the standard Genetic Algorithm *PBIL* does not remember the best candidate solution, but the procedure specificity gives a strong probability to draw those member, because the probabilistic model is updated using it. Random selection of the new population weighted by $\mathbf{p}$ gives great chance for getting a large amount of better candidate solutions. Mainly the new generation is fitted better than the previous one.

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.

The learning rate λ is initialized at the beginning of the iterative procedure and impact on the proceeding.

Poniżej przedstawiono schemat metody *PBIL*.

The *PBIL* method scheme is shown above.

procedure *PBIL:*

Losowanie populacji startowej, zainicjowanie wektora prawdopodobieństw

1. *Generating start population, initializing probability vector*

$$\mathbf{p},\ (p_i = 0.5, \forall i = 1, \ldots, n)$$

Ocena osobników, wybór najlepszego wektora $\boldsymbol{b}$*.*

2. *Fittness evaluation and selection of the fittest individual.*

Modyfikacja składowych wektora prawdopodobieństw $\mathbf{p}$ *według wzoru*

3. *Updating the probability vector using equation:*

$$p_i = (1 - \lambda) \cdot p_i + \lambda \cdot b_i,$$

gdzie λ *– współczynnik uczenia*

Wylosowanie nowej populacji zgodnie z modelem (z uwzględnieniem aktualnego $\boldsymbol{p}$*)*

4. *Random selection of the new generation using probabilistic model weighted by current* $\boldsymbol{p}$

Sprawdzenie warunku zatrzymania, jeśli spełniony – zakończenie algorytmu, w przeciwnym razie powrót do 2.

5. *Verification of the termination conditions, if are fullfilled – ending the algorithm, otherwise return to* **??***.*

4 cGA

Kolejnym omawianym w pracy algorytmemjest cGA (ang. Compact Genetic Algorithm), metoda będącą modyfikacja AG i wykorzystujaca model probabilistyczny.
The next discussed algorithm is *cGA* (Compact Genetic Algorithm), procedure being transformation of Genetic Algorithm and using probabilistic model.

Podobnie jak w *PBIL*, kolejne pokolenia osobników tworzone są w oparciu o model probabilistyczny. Model budowany jest w oparciu o rozwiązania z pokolenia bieżącego, przy czym w jego konstrukcji rolę odgrywa zarówno najlepszy, jak i najgorszy osobnik. Rolę modelu ponownie pełni wektor prawdopodobieństw $\mathbf{p}$, którego składowe wyznacza się według wzoru:
Similar to the *PBIL* algorithm new individuals generations are generated using the probabilistic model. This model is created on basis of the current generation results. Updating the probabilities is considering the best and the worse fitted individual. Mentioned model is realized by a probabilities vector $\mathbf{p}$, which coordinates are generated using the following equation:

$$p_i = \begin{cases} p_i + \frac{1}{m}, & x_i = 1 \wedge y_i = 0 \\ p_i - \frac{1}{m}, & x_i = 0 \wedge y_i = 1 \\ p_i, & \text{pozostałe} \end{cases} \tag{3}$$

gdzie $\mathbf{x} = [x_1, \ldots, x_m]$ i $\mathbf{y} = [y_1, \ldots, y_m]$ to odpowiednio najlepszy i najgorszy osobnik w populacji, zaś m to liczebność populacji.
where $\mathbf{x} = [x_1, \ldots, x_m]$ and $\mathbf{y} = [y_1, \ldots, y_m]$ are respectively the best and the worse fitted individuals, value m is the population size.
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.

Similarly to the previous method the vector $\mathbf{p}$ coordinates determine probability of occurrence ones at the i-th position the new individual. Parameter $\frac{1}{m}$ in equation (**??**) method is a learning rate. Due to using the best and the worse individuals the *cGA* algorithm lets us to make the probabilistic model more efficient after the update. Building vector $\mathbf{p}$ procedure increases the chance to select randomly individual close to the fittest one

and decreaces for the worse element.

Schemat algorytmu zaprezentowano poniżej w formie pseudokodu.

Above, in form of pseudocode we can see the algorithm scheme.

---

procedure *cGA*

Losowanie populacji startowej, zainicjowanie wektora prawdopodobieństw $\boldsymbol{p}$ *(wszystkie wartości* $p_i = 0.5, i = 1, \ldots, n$*).*

1. *Generating the start population in random selection weighted by probabiblities vector* $\boldsymbol{p}$ *(all of the values* $p_i = 0.5, i = 1, \ldots, n$*)*

   Ocena osobników, wybór najlepszego i najgorszego (porównanie z najlepszym i najgorszym z poprzedniej populacji).

2. *Checking the fittness of polulation elements and selection the best and the worse fitted individual.*

   Obliczenie wektora prawdopodobieństw według wzoru

3. *Updating the probability vector using equation:*

$$p_i = \begin{cases} p_i + \frac{1}{m}, & x_i = 1 \wedge y_i = 0 \\ p_i - \frac{1}{m}, & x_i = 0 \wedge y_i = 1 \\ p_i, & \textit{pozostałe} \end{cases}$$

   Wygenerowanie nowej populacji z uwzględnieniem prawdopodobieństw $\boldsymbol{p}$

4. *Generating the new population using weights* $\boldsymbol{p}$

   Sprawdzenie warunku zatrzymania, jeśli spelniony – zakończenie algorytmu, w przeciwnym razie powrót do punktu 2.

5. *Verification of the termination conditions, if are fullfilled – ending the algorithm, otherwise return to* **??**.

---

5 Test tasks description/ 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.

In the following thesis there were optimalizing threee kinds of test function. In every single case was assumed that the solution is a binary string length of k. The assumption was forced by specificity of the optimalization methods.

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

Each of tested problems had a different character to the fully recognition of advantages and disadvantages presented heuristics.

5.1 $trap_n$

Pierwszą funkcją testową była funkcja $trap_n$ dana wzorem:

First test function was the $trap_n$ function given by the formula:

$$f_{trap_n}(\mathbf{u}) = \begin{cases} n - 1 - u_1, & \text{dla } u_1 < n \\ n, & \text{w pozostałych przypadkach} \end{cases}, \tag{4}$$

$$f_{trap_n}(\mathbf{u}) = \begin{cases} n - 1 - u_1, & \text{dla } u_1 < n \\ n, & \text{otherwise} \end{cases}, \tag{5}$$

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

where n is the function rand and u_1 is the number of ones in vector $\mathbf{u}$.

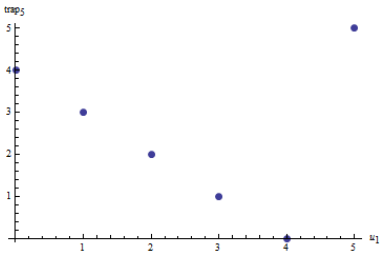
Zwykle przyjmuje się, że rząd funkcji $trap_n$ jest taki sam jak wymiar zadania, tzn. $k = n$
Usually we consider functions $trap_n$ rank equal to the task dimension ($k = n$).

Przykładowo, funkcja $trap_5$ wyraża się wzorem:
For example, fuction $trap_n$ is given by the formula:

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

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

and reaches global maximum equal to 5 for $\mathbf{u}_{opt} = (1, 1, 1, 1, 1)$.
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.**??**).
In oposite to the classic tested function *OneMax*, values of the $trap_n$ function do not depend on the number of ones in vextor $\mathbf{u}$, what makes the problem difficult to optimalize (fig. **??**).



Rysunek 1: Function $trap_n$ plot. / *Wykres funkcji* $trap_5$

5.2 $3 - deceptive$

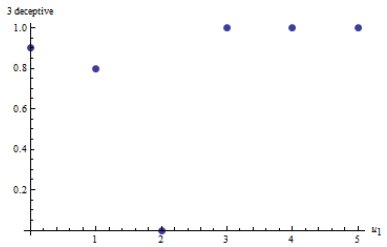
Drugą testowaną w ramach pracy funkcją była funkcja $3 - deceptive$ zadana wzorem:
Second of the tested function was the $3 - deceptive$ function given by the formula:

$$f_{3deceptive}(\mathbf{u}) = \begin{cases} 0.9, & \text{dla/for } u_1 = 0 \\ 0.8, & \text{dla/for } u_1 = 1 \\ 0, & \text{dla/for } u_1 = 2 \\ 1, & \text{w pozostałych przypadkach /otherwise} \end{cases}. \quad (6)$$

Podobnie jak w poprzednim przypadku u_1 oznacza liczbę składowych wektora $\mathbf{u}$, które przyjmują wartość 1.
As in the previous case u_1 denotes the number of ones in vector $\mathbf{u}$.
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. This function has one global minimum and two subtly valued local maxima. Above and beyond if the lenghth k of the vector $\mathbf{u}$ is greater than 3, the global maximum with value 1 is reached in many points. That situation makes the optimalization difficult because of the plurality of solutions reaching this same value.

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, $m - -$elementowego podzbioru A, do którego należą punkty, których suma wzajemnych odległości jest największa.
The last and in parallel most interesting function used for test was the $MaxDiversity$ function. In this case the optimalization problem is to find in $k - -$element set X an $m - -$element subset A where belong points which distance among each other is the greatest.
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, ze i-ty



Rysunek 2: Wykres funkcji 3*deceptive*

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 (ich odległość wynosi $\sqrt{2}$). Tak postawione zadanie ma oczywiście dwa rónoważne rozwiązania optymalne.
Funkcja MD ($MaxDiversity$), funkcja celu której maximum poszukujemy przyjmuje zatem postać:

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

gdzie $A = \{\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_m\}$ – m-elementowy podzbiór zbioru X, a $d(\cdot, \cdot)$ 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).

W zadanie $MaxDiversity$ rozwiązania poszukuje się w postaci wektora o k składowych z których m ma wartość 1 (1 na pozycji i oznacza, że i-ty punkt zbioru X należy do A)

6 Wyniki testów

Celem testów numerycznych była ocena metod *PBIL* oraz *cGA*, ich porównanie oraz ewentualny dobór 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 pojedynczym teście wykonywano 100 eksperymentów, przy czym eksperyment rozumie się jako procedurę iteracyjna prowadzona 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, różnej liczebności populacji oraz różnych wartości współczynnika uczenia się.
W ogólnym przypadku maksimum globalne funkcji $trap_n$ to

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

. W tabelach poniżej, jako rezultat, zamieszczono średnią (ze 100 eksperymentów) liczbę iteracji koniecznych do uzyskania rozwiązania optymalnego.
W nielicznych przypadkach, liczba 100 kroków iteracyjnych nie wystarczała do wyznaczenia maksimum globalnego. Wówczas w tabeli zamieszczono dodatkowo (w nawiasie) informacje o średnim (~~trzeba napisać jak liczono ten błąd?~~) błędzie rozwiązania w 100 doświadczeniach. Jeśli w trakcie 100 kroków znalezione zostało optimum, błąd przyjmuje wartość zero. Natomiast w przypadku zakończenia algorytmu po 100 krokach bez uzyskania szukanej wartości, błąd był przyjmowany jako odległość od najlepszego znanego wyniku. Po przeprowadzeniu 100 doświadczeń, jako błąd uznawana była uśredniona wartość błędów pojedynczych doświadczeń.
W każdej tabeli dodatkowo wyróżniono przypadki, w których algorytm znalazł dokładne rozwiązanie w (średnio) najmniejszej liczbie iteracji.

Tabela 1: Wyniki testów na $trap_5$

	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(0.)	1.59(0.)	1.88	2.02	2.6
50	**1.16**	1.2	1.18	1.21	1.32	1.21

Trap 6

Tabela 2: Wyniki testów na $trap_6$

	Ilość iteracji (błąd)					
	CGA	*PBIL*				
Liczebność populacji		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
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

Trap 7

Funkcja zadana wzorem (**??**) dla $n = 7$ ma maksimum globalne

$$f^{max}_{trap_7}(\underbrace{1, 1, \ldots, 1}_{7}) = 7$$

Tabela 3: Wyniki testów na $trap_7$

	Ilość iteracji (błąd)					
	CGA	*PBIL*				
Liczebność populacji		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
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

Trap 10

Funkcja zadana wzorem (**??**) dla $n = 10$ ma maksimum globalne

$$f^{max}_{trap_{10}}(\underbrace{1, 1, \ldots, 1}_{10}) = 10$$

Tabela 4: Wyniki testów na $trap_{10}$

	Ilość iteracji (błąd)					
	CGA	*PBIL*				
Liczebność populacji		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
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 dla funkcji $trap_n$ metoda cGA sprawdza się lepiej od $PBIL$ w przypadku mniejszej licznych populacji, ale gdy do obliczeń można wykorzystać większe populacje, błąd metody $PBIL$ spada do zera i szukane maksimum znajdowane jest w mniejszej liczbie korkó niż w przypadku cGA.

Jeśli chodzi o zależność ilość iteracji w stosunku do współczynnika uczenia się, to trudno sformułować uogólnione wnioski. Na przykład dla populacji 50 osobników lepszy rezultat uzyskiwany jest dla $\lambda = 0.1$, a dla populacji dwukrotnie większej $\lambda = 0.2$ daje lepsze rezultaty.

Dodatkowo warto zauważyć, że wraz ze wzrostem rzędu n, rośnie złożoność funkcji $trap_n$, co oczywiście wpływa na czas pracy obu metod. Jeśli więc istnieje potrzeba optymalizacji przy wukorzystaniu mniejszych populacji, algorytm cGA wydaje się być lepszym wyborem. Wykorzystując PBIL, najbezpieczniejszą wartością λ jest 0.01, (dla tej wartości ryzyko nieuzyskania dokładnego wyniku było najmniejsze).

6.2 3-deceptive

Funkcja *3-deceptive* osiąga maksimum globalne o wartości 1 jeśli co najmniej 3 ze składowych wektora są jedynkami.
W pracy rozważono 3 warianty funkcji, określonej ogólnym wzorem (**??**):
Pierwszy, kiedy rozważane były wektory długości 3, co implikowało istnienie dokładnie jednego ekstremum globalnego. W pozostałych przypadkach funkcja *3-deceptive* przyjmuje optimum w wielu punktach przestrzeni.
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.

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

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

A–zbiór wektorów z co najmniej trzema jedynkami.

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

Wariant 1 - przestrzeń wektorów długości 3

Tabela 5: Wyniki testów na 3 – *deceptive* dla 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 W przypadku najmniej licznej populacji oraz współczynnika uczenia się na poziomie 0.5 algorytm PBIL nie zawsze w 100 iteracjach znajdował wynik dokładny. Przyczyną jest tutaj z pewnością zbyt szybkie (duża wartość λ) zdominowanie populacji przez jednego osobnika. Każda inna konfiguracja parametrów daje bardzo zbliżone rezultaty.

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

Wariant 2 - przestrzeń wektorów długości 5

Tabela 6: Wyniki testów na 3 – *deceptive* dla 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

Podsumowanie: Metoda daje porównywalnie dobre rezultaty w przypadku obu stosowanych algorytmów. W tym przypadku dobór współczynnika uczenia się nie ma znacznego wpływu na rezultat. W wielu eksperymentach na dokłoptymalny wynik trafiono w iteracji zerowej. Czy w zadaniu obserwowano tylko wartośc funkcji, czy też zwracano uwagę na to dla jekiego wektora została ona znaleziona? nie wiem czy nie zrezygnować z tego testu :/

Wektory 10-cio elementowe

Tabela 7: Wyniki testów na $3-$ *deceptive* dla 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

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 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 Max diversity

W przypadku funkcji $MaxDiversity$, zadanie polega na znalezieniu m-elementowego podzbioru A danego zbioru X (o mocy k). Rozwiązanie reprezentowane jest przez k-elementowy wektor binarny, przy czym jedynka na i-tej pozycji oznacza, że i-ty element zbioru X należy do podzbioru A. Ze względu na odmienną naturę problemu, błąd rozwiązania będzie przedstawiany w skali procentowej. Wielkość tą należy rozumieć jako odsetek poprawnie wyznaczonych rozwiązań w 100 doświadczeniach. Podobnie jak w poprzednich testach, kryterium zatrzymania algorytmu jest znalezienie rozwiązania optymalnego, bądź wykonanie maksymalnej dopuszczalnej liczby iteracji (wówczas za rozwiązanie przyjmuje się najlepszy uzyskany wynik).

6.3.1 X - wierzchołki kwadratu w przestrzeni 2-wymiarowej

W zadaniu szukamy n-elementowego podzbioru zbioru wierzchołków

$$X = \{(0,0),(0,1),(1,0),(1,1)\},$$

takiego, aby suma odległość pomiędzy punktami była największa.

1. Szukamy pary punktów ($m = 2$) ze zbioru X, których odległość jest maksymalna.
   W tym wypadku rozwiązanie optymalne to

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

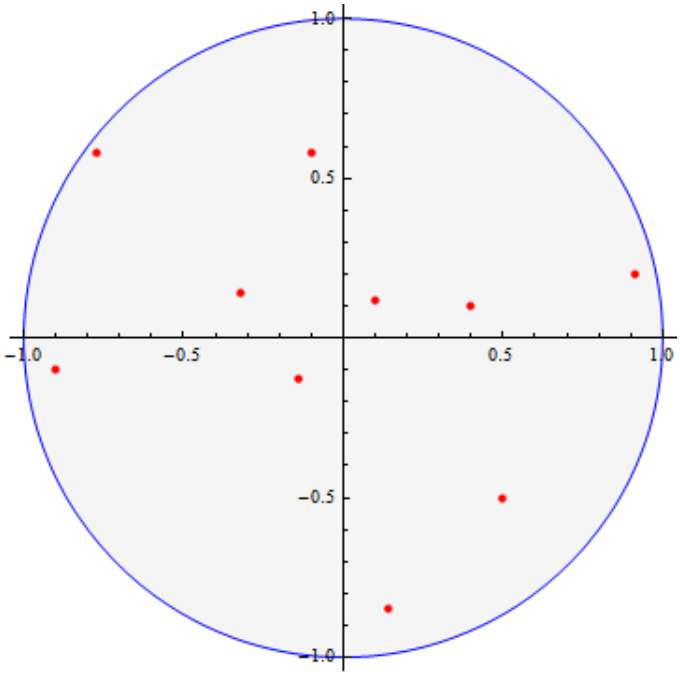
   gdzie $A = \{(0,0),\ (1,1)\}$ lub $A = \{(1,0),(0,1)\}$

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%*

2. W zbiorze X szukamy 3-elementowego podzbioru A.
   W tym przypadku, każdy dowolny 3-elementowy podzbiór badanego problemu daje rozwiązanie optymalne $MD^{max}(A, X) = 2 + \sqrt{2}$
   . Oba algorytmy zadziałały poprawnie i zwróciły dokładne wartości ekstremum już na etapie losowania populacji startowej. W związku z tym, nieistotny był dobór parametrów algorytmów, tj. liczebności populacji i współczynnika uczenia się.

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

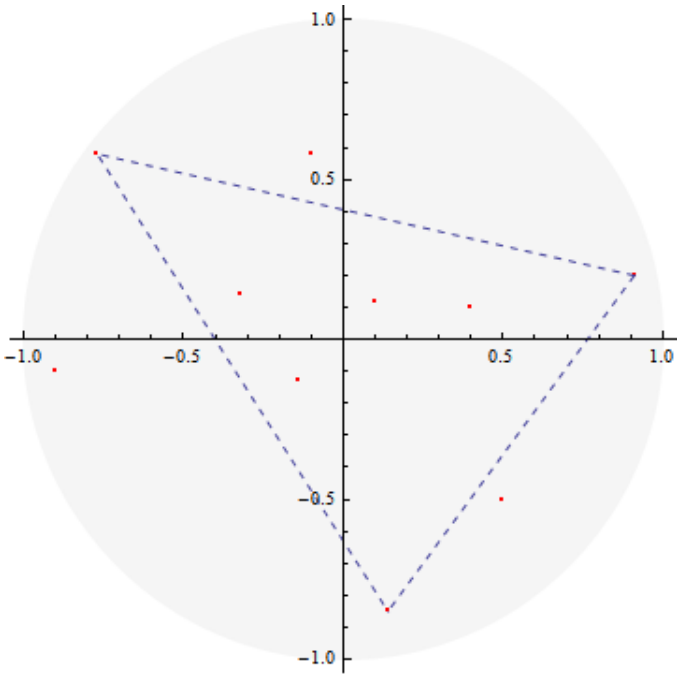
W zadaniu przyjęto, że
$X = \{(0.5, -0.5), (0.4, 0.1), (-0.9, -0.1), (0.1, 0.12), (-0.32, 0.14), (-0.1, 0.58),$
$(0.911, 0.2), (-0.77, 0.58), (0.14, -0.85), (-0.14, -0.13)\}$ oraz że w zbiorze X poszukuje się 3-elementowego podzbioru A.
W tabeli poniżej zaprezentowano wyniki uzyskane w eksperymentach 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%*

Wnioski: jeszcze to trzeba przemyśleć

Jeśli rozważamy występowanie błędów, algorytm *cGA* zwraca poprawne wyniki dla mniejszej liczebności populacji. Porównując czas pracy algorytmów, można zauważyć, że *cGA* zwraca rezultat w krótszym czasie niż *PBIL*. Zarówno wysoki, jak i bardzo niski wskaźnik uczenia się algorytmu *PBIL* znacznie zmniejsza efektywność pracy algorytmu. Dodatkowo wskaźnik rzędu 0.5 powoduje wzrost częstotliwości wystąpienia błędu w obliczeniach. Najbardziej optymalną wartością współczynnika uczenia się okazała się wartość 0.1, dla której rezultaty *cGA* i *PBIL* są zbliżone



Rysunek 4: Maksymalna odległość

6.3.3 Wierzchołki sześcianu

Szukamy pary punktów ze zbioru wierzchołków sześcianu X, których wzajemna odległość jest maksymalna.

$$X = \{(0,0,0),(0,0,1),(0,1,0),(1,0,0),(1,1,0),(1,0,1),(0,\ 1,1),(1,1,1)\}$$

Szukanym ekstremum jest zbiór A

$$MD^{max}(A,X) = \sqrt{3},$$

gdzie $A = \{(0,0,0),(1,1,1)\}$ lub $A = \{(1,0,0),(0,1,1)\}$ lub $A = \{(0,0,1),(1,0,0)\}$ lub $A = \{(0,1,0),(1,0,1)\}$

Tabela 10: Wyniki testów na $MaxDiversity$ dla wierzchołków sześcianu ($m = 2$)

Liczebność populacji	*Ilość iteracji*											
	cGA		*PBIL*									
			$\lambda = 0.5$		$\lambda = 0.2$		$\lambda = 0.1$		$\lambda = 0.01$		$\lambda = 0.0001$	
3	4.67	*100%*	19.67	*83%*	3.78	*100 %*	**3.21**	*100%*	3.36	*100%*	3.69	*100%*
5	3.04	*100%*	11.52	*91%*	**2.6**	*100%*	2.95	*100%*	2.71	*100%*	2.90	*100%*
20	1.95	*100%*	1.92	*100 %*	1.97	*100%*	**1.86**	*100%*	1.87	*100%*	1.93	*100%*
50	1.91	*100%*	1.83	*100%*	1.81	*100%*	**1.80**	*100%*	1.84	*100%*	1.89	*100%*
100	1.89	*100%*	1.87	*100%*	1.86	*100%*	1.87	*100%*	**1.81**	*100%*	1.84	*100%*