Zastosowanie algorytmów ewolucyjnych

Optymalizacja zysków piekarni

Autorzy: Karolina Kulpa, Julia Gajtkowska
Wydział: Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej
Kierunek: Automatyka i Robotyka

Spis Treści

Model zagadnienia

Opis analizowanego problemu

Nasz projekt ma na celu przeprowadzenie optymalizacji zysków piekarni. Mając odpowiednie dane tzn. koszty i ilość potrzebnych półproduktów oraz końcową cenę wypieków zakładamy maksymalizację zysków.

Pierwszą rzeczą, którą należało zrobić było określenie typu zadania. Główną częścią zadania jest poszukiwanie wektora przedstawiającego ilość produktów do wytworzenia, która przyniesie największe zyski. Nałożyłyśmy ograniczenia związane z możliwościami piekarni. Personel nie może wyprodukować więcej niż konkretną ilość produktów określoną przez pracodawcę. Dodatkowo posiadamy wektor liczb, które określają minimum produktów każdego rodzaju. Naszym celem jest jednoczesna minimalizacja wydatków związanych ze zużyciem składników i maksymalizacja zysków, które obliczamy sumując ceny wyrobów.

Do rozwiązania naszego problemu używamy algorytmu ewolucyjnego. Jest on bardziej skomplikowany, więc nie da się go aproksymować żadnym z najczęściej poruszanych problemów optymalizacyjnych.

Przyjęte uproszczenia

- koszt energii elektrycznej związany z produkcją danego produktu jest w przypadku każdego produktu taki sam , oraz pomijamy go podczas optymalizacji zysków
- z każdego wyprodukowanego produktu osiągniemy zysk (wszystkie produkty zostają sprzedane)
- czas poświęcony na produkcję danego produktu jest w przypadku każdego produktu taki sam, pomijamy różnice w cenie wynikające w różnego czasu pracy personelu nad danym produktem.

Model Matematyczny

Nasz projekt przedstawia optymalizację zysków piekarni, która ma możliwość wyprodukowania określone wyroby i ich określoną z góry cenę.

Funkcja celu

$$f(g) = \sum_{i=1}^{6} p_i * n_i - \sum_{i=1}^{10} w_j * v_j \quad (1)$$

gdzie:

N_i- ilość i-tego produktu do stworzenia i ∈ {1,2,3,4,5,6}
x_j- aktualna zakupiona ilość j-tego półproduktu j ∈ {1,2,3,4,5,6,7,8,9,10}
p_i- cena i-tego produktu
v_j- wartość cenowa j-tego półproduktu

w_j - ilość zużytych j-tych półproduktów
n_i - liczba wyprodukowanych i-tych produktów
g - zysk końcowy (wartość produkcji minus wartość zużytych półproduktów)

Ograniczenia:

Naszymi ograniczeniami są możliwości piekarni. Pracownicy mogą jednorazowo wyprodukować maksymalnie 250 produktów. Zakład ma również określoną ilość półproduktów oraz minimalną ilość każdego wypieku. Za przekroczenie któregokolwiek limitu naliczana jest kara. W przypadku półproduktów, w wysokości aktualnej ceny składnika pomnożonej przez przekroczoną ilość. Analogicznie w przypadku minimum produktów.

Funkcje kary:

$$f_{k1}(g) = f(g) - \sum_{j=0}^{10}(w_j - x_j) \quad (2)$$

jeśli wartość któregoś działania $(w_j - x_j) < 0$ to odejmujemy 0

$$f_{k2}(g) = f(g) - \sum_{i=0}^{6}(N_i - n_i) \quad (3)$$

jeśli wartość któregoś działania $(N_i - n_i) < 0$ to odejmujemy 0

Algorytm

Metoda selekcji

Zadaniem selekcji jest wybór osobników, które zostaną wybrane do nowej generacji. Podczas selekcji istotne jest to aby wśród osobników wybranych do nowego pokolenia znalazły się te , które zostały najlepiej ocenione. Selekcja powinna również zapewnić różnorodność nowego pokolenia, dzięki czemu algorytm ma szansę stworzyć lepsze lepsze rozwiązania poprzez mutacje i krzyżowanie zarówno lepiej jak i gorzej ocenianych osobników. W algorytmie zastosowaliśmy i testowaliśmy dwie różne metody selekcji: metodę ruletki oraz turnieju. Obie te metody mają wspólne zalety:

- nie wymagają żadnych informacji statystycznych opisujących populacje
- pozwalają na utrzymanie stałej wielkości nowej generacji , co jest ważne w przypadku dołączenia nowych osobników podczas operacji krzyżowania i mutacji

Typy selekcji

1. Metoda turnieju

Z obecnej populacji jest wybierana dana ilość osobników (rozmiar turnieju). Następnie spośród wybranych osobników najlepszy przechodzi do nowej generacji. turniej jest

powtarzany tyle razy, ile ma wynosić liczebność nowej generacji. Rozmiar turnieju jest stały dla każdej instancji testowej, a jego dobór odbywa się w sposób losowy. Wraz ze wzrostem rozmiar turnieju wzrasta napór selekcyjny i do nowej generacji trafiają lepiej oceniane osobniki.

2. Metoda ruletki

Dla każdego osobnika jest przydzielany ‘’wycinek koła’ (u nas ‘kołem’ jest przedział <0,1>), którego rozmiar jest zależny od oceny osobnika (im lepiej oceniany jest osobnik tym większy wycinek koła dostaje) i jest obliczany za pomocą poniższego wzoru:

$$p(i) = \frac{F(i)}{\sum_{i=1}^{n} F(i)}$$

gdzie:

p(i) - prawdopodobieństwo wybrania i-tego osobnika (wielkość wycinka koła tego osobnika)
F(i) - funkcja przystosowania i-tego osobnika
n - wielkość populacji

Następnie jest losowana liczba z przedziału koła i do następnego pokolenia przechodzi ten osobnik w którego wycinek koła liczba trafiła. taki rodzaj selekcji sprawia że większą szansę na przejście do następnego pokolenia mają osobniki lepiej oceniane.

Operatory krzyżowania

Krzyżowanie można rozumieć jako analogiczne do rozmnażania osobników. Jest ono operatorem, który nie modyfikuje wartości poszczególnych genów, a dokonuje ich “wymiany”. W naszym przypadku losujemy 2 rodziców, następnie generujemy dla genu “taty” maskę, czyli wektor składający się z zer i jedynek (krzyżowanie równomierne). Następnie tworzymy maskę “mamy” przy użyciu funkcji ‘*numpy.logical_not*’. W ten sposób uzyskujemy odwrotne sobie maski, które pozwalają nam na krzyżowanie zaprezentowane poniżej. Dodatkowo krzyżujemy danych rodziców dwa razy, ale z odwrotnymi maskami.

Maski: [0 0 1 1 0 1] [1 1 0 0 1 0]
Parent gen: [73 60 49 41 57 15] x [70 60 47 41 52 16] = [70 60 49 41 52 15]

Maski: [1 1 0 0 1 0] [0 0 1 1 0 1]
Parent gen: [73 60 49 41 57 15] x [70 60 47 41 52 16] = [73 60 47 41 57 16]

W celu znalezienia najlepszego rozwiązania postanowiłyśmy zastosować jeszcze drugą metodę krzyżowania, która polega na wymianie dokładnie połowy genów dwóch osobników (krzyżowanie jednopunktowe).

Maski: [0 0 0 1 1 1] [1 1 1 0 0 0]

Parent gen: [73 60 49 41 57 15] x [70 60 47 41 52 16] = [70 60 47 41 57 15]

Maski: [1 1 1 0 0 0] [0 0 0 1 1 1]

Parent gen: [73 60 49 41 57 15] x [70 60 47 41 52 16] = [73 60 49 41 52 16]

Krzyżowanie jest stosowane z pewnym prawdopodobieństwem na losowych osobnikach populacji. Elementem, który o tym decyduje jest parametr *pc*. Ma on swoją wartość domyślną równą *0,8*. Po wylosowaniu rodziców wybieramy parametr *e* z przedziału od 0 do 1. Jeśli *e* <= *pc* to krzyżowanie się odbywa, w odwrotnej sytuacji pomijamy tą czynność i losujemy nowych rodziców.

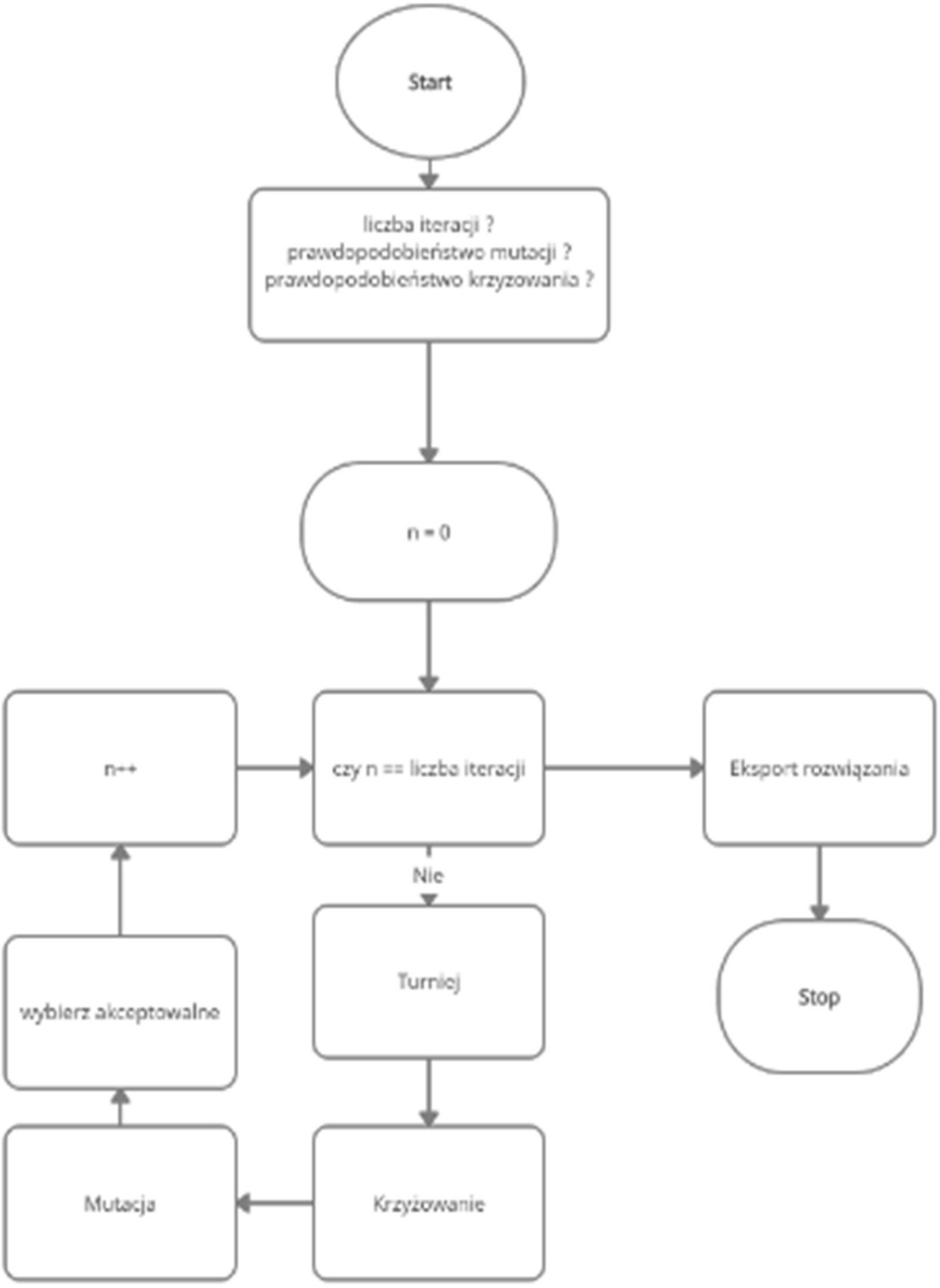
Dana funkcja jest wywoływane w każdej iteracji naszego programu, a na końcu uzyskujemy ilość dzieci równą ilości rodziców.

Operator mutacji

Dzięki mutacji możemy wprowadzić różnorodność rozwiązań, jednak jest możliwość zepsucia dobrego rozwiązania. Mutacja następuje z pewnym prawdopodobieństwem. W zależności od parametru *pm (0.2)*, dany osobnik jest poddany tej czynności lub nie (analogicznie do krzyżowania). Jeśli *e* > *pm* to dodajemy rodzica do zbioru dzieci Zmienna *n_mutations* decyduje o tym ile elementów wektora zostanie losowo wybranych z przedziału od 10 do 115. Wartość 10 to minimalna ilość produktów, gdy przyjmowałyśmy 0 to ostatni produkt za każdym razem miał właśnie tą wartość. Natomiast maksimum jest równe 115. Wybiera się również losowo elementy wektora, które mają ulec zmianie. Funkcja mutacji jest wykonywana w każdej iteracji programu. Przykład:

n_mtations = *2* [73 60 49 41 57 15] → [73 60 49 27 69 15]

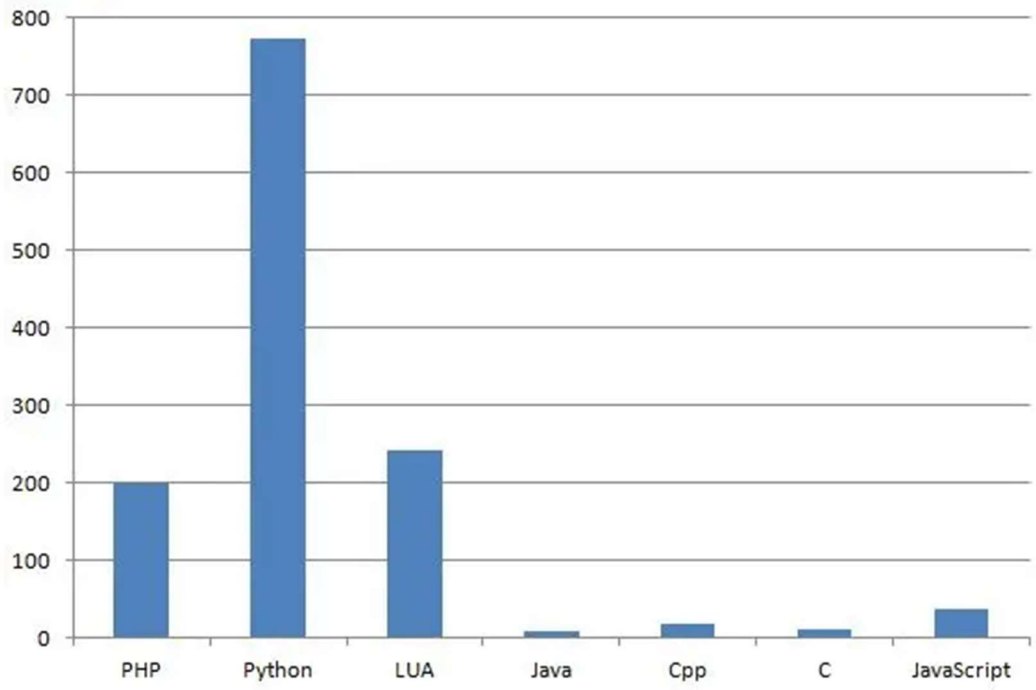
Uproszczony schemat blokowy algorytmu



Aplikacja

Język programowania

Wybranym przez nas językiem programowania jest Python - język programowania wysokiego poziomu. używana przez nas wersja to 3.8. Język ten został przez nas wybrany przez spory pakiet bibliotek standardowych oraz funkcjonalnych. Jednak cenę, którą płacimy za wygodę i dostępność wielu komponentów jest relatywna szybkość w stosunku do innych język



Rys 1. Porównanie szybkości kilku języków programowania

Biblioteki

Numpy - została stworzona, aby umożliwić szybkie i sprawne operacje na macierzach. Każdy element jest tego samego typu – zazwyczaj są to liczby. Dostarcza również implementacje pseudolosowego doboru liczb. Ma bardzo szerokie zastosowania, w związku z czym istnieje mnóstwo materiałów na jej temat a także łatwo dostępna szczegółowa dokumentacja.

Matplotlib - biblioteka do tworzenia wykresów dla języka programowania Python i jego rozszerzenia numerycznego NumPy. Zawiera ona API "pylab" zaprojektowane tak aby było

jak najbardziej podobne do MATLABa, przez co jest łatwy do nauczenia przez jego użytkowników

Time - biblioteka wykorzystywana do testów czasowych

PySimpleGUI - jest pakietem Pythona, który umożliwia tworzenie GUI. Okno GUI określa się za pomocą „układu", który zawiera widżety (w PySimpleGUI nazywane są one „Elementami"). Twój układ jest używany do tworzenia okna przy użyciu jednej z 4 obsługiwanych struktur do wyświetlania i interakcji z oknem. Obsługiwane platformy obejmują tkinter, Qt, WxPython lub Remi.

Dane które należy podać w oknie początkowym:

- parametr krzyżowania
- parametr mutacji
- rozmiar populacji
- liczba iteracji

W wyniku otrzymujemy najlepszy wynik oraz rozwiązanie w postaci wektora. Wyświetla się również wykres, który pokazuje w jaki sposób algorytm doprowadził do takiego wyniku.

Testy

Instancje testowe

W celu wykonywania testów przygotowaliśmy dwie instancje testowe:

Pierwsza instancja testowa - instancja losowo wygenerowanych wektorów , które spełniają główny warunek a wartość funkcji celu w tej instancji waha się pomiędzy 0 a około 120.

norm = [[31, 15, 51, 29, 84, 27] , [30, 35, 114, 22, 14, 16] , [52, 4, 11, 81, 25, 10] , [4, 42, 32, 39, 74, 48] , [31, 49, 68, 18, 47, 27] , [3, 2, 9, 105, 57, 50] , [114, 23, 0, 20, 20, 2] , [23, 6, 64, 44, 86, 4] , [77, 16, 91, 13, 1, 36] , [15, 55, 2, 73, 51, 41] , [25, 3, 20, 8, 6, 8] , [68, 33, 18, 3, 18, 79] , [3, 5, 23, 75, 36, 42] , [16, 29, 6, 51, 54, 14] , [12, 5, 19, 97, 52, 29] , [23, 20, 55, 78, 52, 19] , [28, 75, 16, 14, 2, 71] , [30, 8, 17, 97, 65, 20] , [32, 5, 41, 41, 8, 29] , [17, 39, 9, 55, 22, 53] , [61, 13, 35, 3, 96, 19] , [52, 13, 53, 32, 78, 13] , [13, 55, 49, 42, 16, 24] , [50, 17, 38, 9, 22, 43] , [16, 27, 29, 65, 8, 66] , [73, 9, 16, 3, 24, 20] , [13, 10, 37, 32, 1, 34] , [7, 13, 68, 4, 76, 30] , [14, 88, 13, 20, 34, 75] , [31, 17, 11, 74, 65, 40] , [4, 25, 51, 7, 27, 6] , [1, 43, 30, 41, 37, 75] , [22, 28, 74, 19, 69, 36] , [12, 66, 0, 45, 39, 37] , [13, 37, 113, 30, 41, 1] , [18, 53, 15, 4, 98, 45] , [61, 16, 3, 8, 110, 7] , [25, 8, 54, 51, 6, 43] , [16, 43, 19, 56, 7, 53] , [10, 40, 22, 58, 15, 99] , [5, 40, 82, 33, 30, 48] , [15, 29, 5, 9, 82, 56] , [4, 55, 14, 4, 10, 65] , [2, 45, 10, 85, 6, 53] , [40, 10, 58, 73, 53, 14] , [12, 6, 1, 30, 27, 14] , [57, 23, 7, 37, 32, 63] , [6, 37, 7, 86, 73, 17] , [23, 103, 29, 2, 16, 71] , [74, 47, 11, 19, 49, 25] , [10, 112, 46, 1, 17, 44] , [10, 8, 37, 43, 69, 15] , [34, 38, 20, 57, 16, 39] , [77, 17, 14, 49, 9, 58] , [3, 0, 107, 33, 1, 45] , [1, 104, 63, 36, 11, 8] , [16, 54, 17, 40, 23, 69] , [3, 36, 39, 37, 51, 39] , [30, 39, 2, 36, 21, 57] , [14, 10, 103, 13, 3, 11] , [27, 45, 87, 38, 7, 14] , [32, 19, 90, 42, 3, 8] , [22, 44, 34, 11, 46, 0] , [52, 52, 25, 28, 5, 9] , [14, 4, 30, 79, 102, 18] , [28, 3, 34, 56, 55, 42] , [8, 15, 71, 35, 75, 43] , [85, 30, 7, 38, 5, 56] , [6, 15, 87, 12, 19, 27] , [9, 41, 52, 13, 56, 48] , [50, 15, 5, 112, 42, 24] , [89, 42, 17, 8, 10, 4] , [71, 37, 13, 20, 5, 22] , [6, 19, 29, 30, 33, 60] , [68, 28, 56, 25, 13, 23] , [12, 71, 25, 6, 48, 24] , [41, 8, 29, 40, 69, 8] , [44, 1, 57, 16, 87, 23] , [31, 52, 48, 11, 57, 50] , [16, 1, 21, 21, 111, 32] , [68, 0, 68, 14, 20, 18] , [0, 96, 68, 8, 22, 36] , [3, 21, 53, 25, 81, 23] , [5, 57, 4, 0, 62, 1] , [27, 44, 66, 47, 16, 30] , [46, 10, 12, 23, 65, 47] , [74, 0, 56, 36, 17, 1] , [37, 78, 35, 14, 8, 20] , [36, 33, 21, 50, 31, 19] , [16, 102, 13, 21, 7, 37] , [35, 10, 85, 71, 10, 27] , [60, 30, 23, 26, 81, 4] , [42, 49, 21, 17, 77, 38] , [97, 7, 42, 8, 30, 1] , [9, 109, 47, 34, 47, 3] , [8, 41, 3, 75, 11, 78] , [62, 33, 57, 17, 34, 34] , [30, 105, 27, 22, 0, 46] , [65, 24, 23, 29, 102, 4] , [36, 22, 19, 26, 45, 64]]

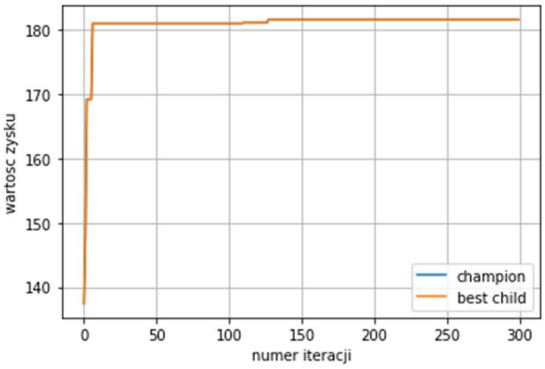
Druga instancja testowa - jest to instancja złośliwa, wartości funkcji celu dla tej instancji są ujemne, takie rozwiązania powodują straty ze względu na nałożone na nie kary.

zlo = [[22, 8, 11, 24, 100, 69] , [31, 14, 99, 16, 3, 66] , [39, 9, 8, 4, 53, 41] , [1, 44, 23, 5, 15, 95] , [11, 25, 44, 3, 88, 79] , [39, 1, 1, 19, 30, 96] , [48, 7, 47, 0, 19, 68] , [88, 12, 6, 30, 16, 93] , [29, 12, 31, 35, 57, 85] , [19, 24, 12, 25, 89, 66] , [20, 5, 10, 2, 40, 62] , [16, 85, 9, 9, 12, 98] , [0, 19, 20, 4, 109, 56] , [23, 4, 5, 3, 11, 29] , [43, 2, 5, 69, 7, 102] , [57, 26, 27, 0, 0, 91] , [5, 11, 33, 5, 111, 42] , [4, 11, 27, 0, 103, 98] , [27, 26, 35, 7, 35, 110] , [13, 43, 29, 6, 3, 98] , [30, 19, 6, 4, 74, 46] , [15, 5, 22, 25, 17, 90] , [18, 24, 52, 19, 39, 86] , [28, 8, 7, 1, 49, 58] , [9, 9, 2, 10, 24, 58] , [0, 12, 16, 16, 81, 40] , [54, 33, 10, 5, 30, 85] , [23, 6, 113, 7, 40, 45] , [13, 6, 70, 25, 40, 96] , [40, 14, 49, 16, 15, 113] , [51, 34, 23, 24, 4, 104] , [5, 66, 15, 4, 38, 93] , [23, 10, 26, 3, 11, 114] , [24, 10, 44, 46, 34, 91] , [13, 6, 3, 20, 5, 85] , [12, 2, 38, 47, 67, 72] , [25, 9, 61, 2, 18, 78] , [6, 0, 32, 3, 84, 64] , [46, 7, 7, 23, 39, 91] , [1, 15, 17, 20, 71, 89] , [12, 26, 37, 10, 38, 62] , [9, 11, 9, 39, 64, 110] , [46, 38, 6, 18, 31, 107] , [13, 4, 45, 45, 6, 90] , [0, 26, 5, 10, 68, 58] , [7, 6, 10, 48, 91, 83] , [45, 44, 9, 16, 24, 102] , [31, 10, 57, 9, 10, 109] , [63, 8, 15, 12, 62, 70] , [3, 8, 60, 6, 23, 64] , [3, 3, 41, 3, 14, 43] , [8, 37, 40, 45, 15, 101] , [28, 10, 7, 5, 93, 78] , [50, 9, 49, 3, 27, 90] , [9, 16, 0, 13, 102, 32] , [25, 4, 16, 12, 62, 92] , [2, 0, 90, 18, 1, 75] , [15, 6, 85, 5, 4, 110] , [30, 0, 62, 1, 20, 54] , [5, 33, 22, 14, 81, 70] , [17, 6, 22, 30, 20, 90] , [28, 0, 57, 4, 25, 104] , [29, 29, 26, 51, 6, 106] , [22, 52, 15, 15, 36, 102] , [3, 2, 10, 54, 49, 109] , [32, 0, 12, 68, 0, 106] , [3, 9, 4, 36, 115, 44] , [11, 20, 20, 25, 4, 113] , [16, 12, 15, 3, 113, 62] , [26, 13, 11, 30, 25, 60] , [4, 9, 4, 36, 101, 47] , [2, 5, 61, 21, 10, 90] , [25, 35, 49, 5, 32, 93] , [33, 37, 24, 10, 31, 113] , [6, 43, 48, 18, 4, 90] , [31, 7, 13, 10, 112, 54] , [7, 3, 29, 26, 49, 111] , [10, 48, 18, 17, 39, 99] , [31, 4, 9, 31, 39, 55] , [24, 2, 24, 5, 69, 60] , [25, 49, 43, 12, 4, 111] , [45, 10, 19, 13, 54, 82] , [12, 23, 3, 32, 85, 59] , [12, 7, 114, 25, 8, 53] , [32, 47, 35, 2, 4, 89] , [47, 24, 3, 16, 5, 97] , [13, 16, 74, 20, 6, 108] , [23, 13, 21, 10, 7, 52] , [6, 18, 11, 9, 94, 72] , [9, 9, 5, 55, 68, 86] , [31, 0, 11, 18, 53, 114] , [43, 3, 56, 40, 6, 93] , [3, 35, 40, 29, 5, 79] , [35, 13, 82, 1, 0, 64] , [44, 18, 45, 6, 24, 81] , [10, 0, 8, 1, 108, 63] , [10, 8, 8, 7, 25, 107] , [43, 42, 22, 2, 17, 103] , [17, 30, 35, 5, 2, 59] , [0, 16, 31, 45, 64, 89]]

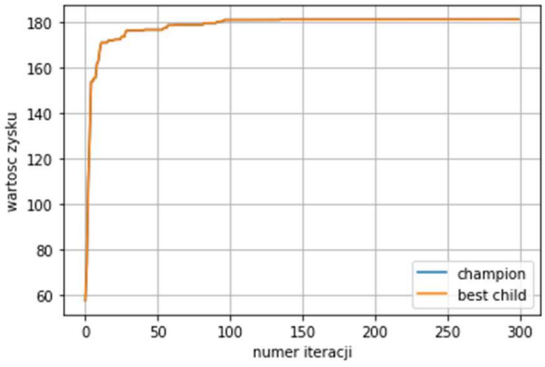
Test wpływu typu rodziców początkowych - instancja złośliwa i łagodna

Tabela 1.

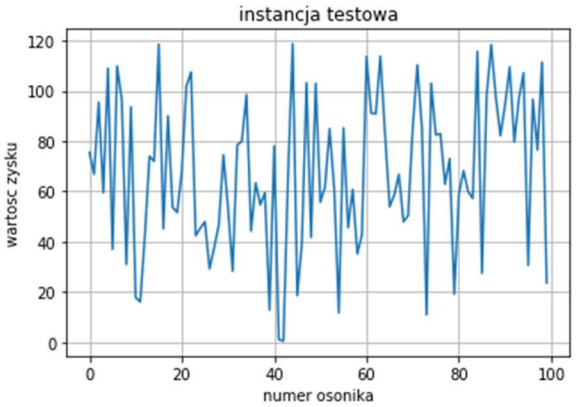
lp.	populacja	F_{start}	F_{best}	F_{worst}	$\overline{F}$	σ	δ	I_{best}
1.	łagodna	119.20	181.58	118.80	173.93	18.95	1.2	127
2.	złośliwa	9.96	180.90	120.18	175.25	14.07	1.59	138



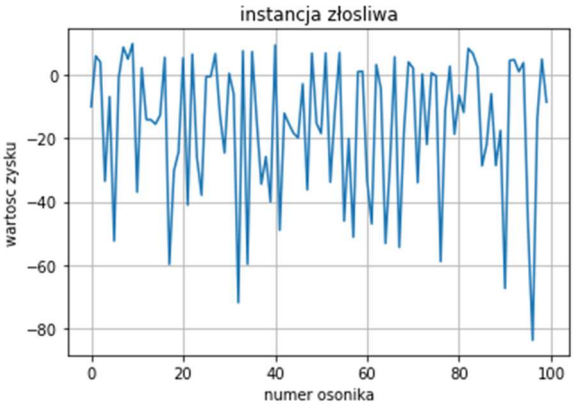
Wykres 1. Populacja łagodna.



Wykres 2. Populacja złośliwa.



Wykres 3. Wartości funkcji celu dla populacji łagodnej.



Wykres 4.Wartości funkcji celu dla populacji złośliwej.

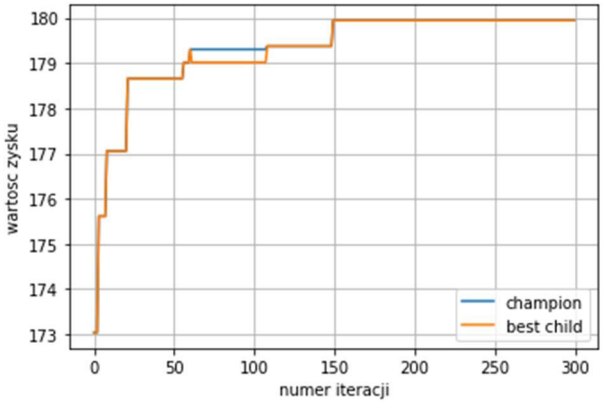
Wykres 5. Wartości błędów względnych kolejnych testów.

Wnioski: Nasz algorytm osiąga bardzo dobrą wartość nie tylko dla zbioru testowego, ale również dla złośliwego. Numery iteracji, w którym funkcje docierają do tej wartości niewiele się od siebie różnią.

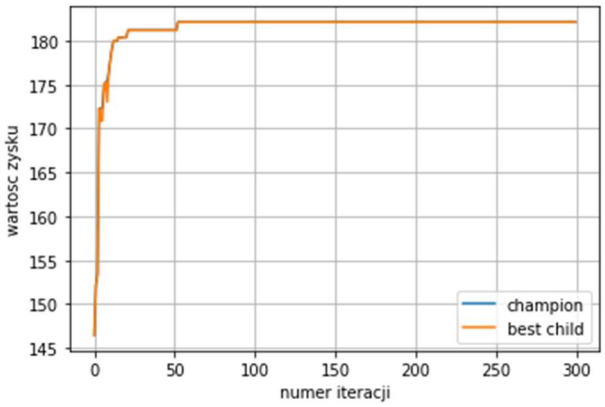
Test wpływu wielkości populacji na wynik

Tabela 2.

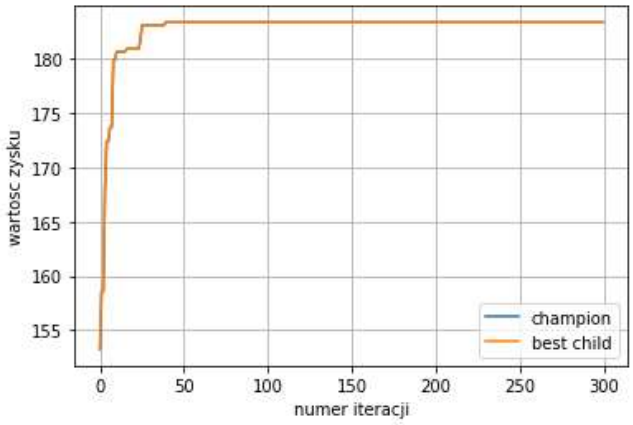
lp.	populacja	F_{start}	F_{best}	F_{worst}	$\overline{F}$	σ	δ	I_{best}
1.	50	-63.08	179.94	123.23	173.26	17.02	2.11	149
2.	100	92.34	182.12	133.85	178.38	11.40	0.93	52
3.	200	146.79	183.39	121.55	176.49	15.61	0.24	39



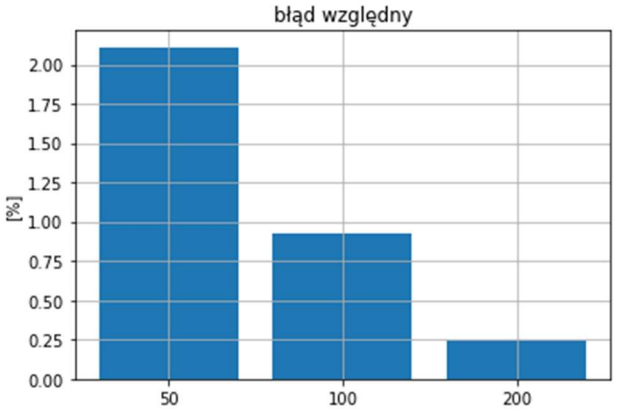
Wykres 6 Wielkość populacji równa 50



Wykres 7. Wielkość populacji równa 100



Wykres 8. Wielkość populacji równa 200



Wykres 9. Wartości błędów względnych kolejnych testów.

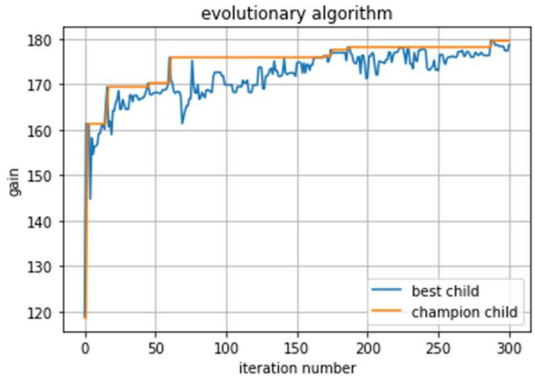
Wnioski: Można stwierdzić, że algorytm zachowuje się lepiej dla większej populacji. Funkcja celu osiąga wyższą wartość, bo ma szerszy wybór i możliwość krzyżowania. Wzrasta nabór selekcyjny i tylko najlepsze osobniki są wybierane.

Test wpływu parametru krzyżowania równomiernego pc

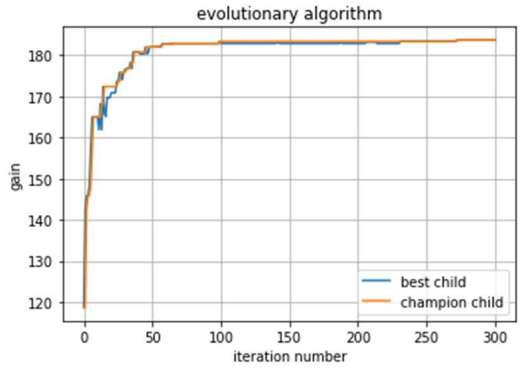
Tabela 3.

lp.	param. krzyżowania pc	F_{start}	F_{best}	F_{worst}	$\overline{F}$	σ	δ	I_{best}
1.	0.3	118,71	179.6	88.22	141.04	28.73	2.30	288
2.	0.6	118,71	183.64	123.77	176.64	15.41	0.10	273

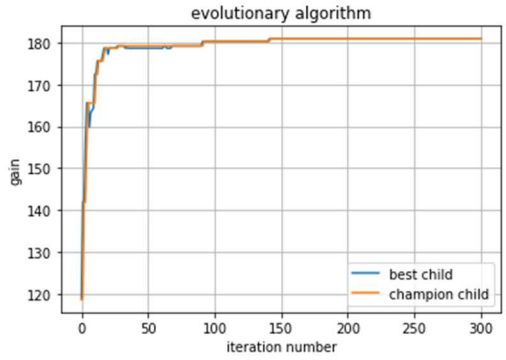
3.	0.8	118.71	179.96	140.23	177.28	8.13	2.10	26
4	0.95	118.71	180.82	123.54	178.77	8.91	1.63	142



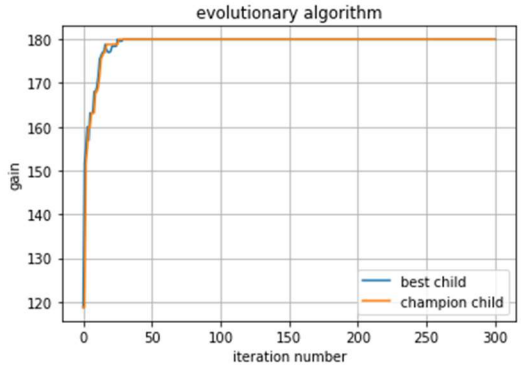
Wykres 10. Wykres dla wartości parametru pc = 0.3



Wykres 11. Wykres dla wartości parametru pc = 0.6



Wykres 12. Wykres dla wartości parametru pc = 0.8



Wykres 13. Wykres dla wartości parametru pc = 0.95

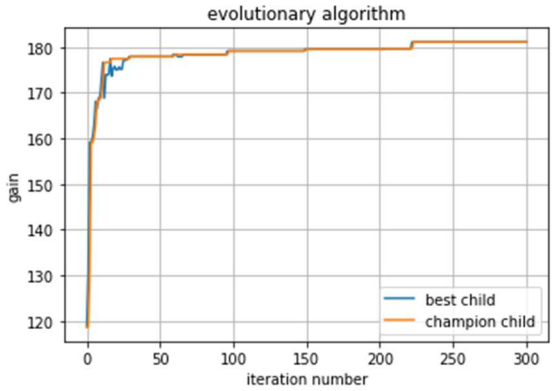
Wykres 14. Wartości błędów względnych kolejnych testów.

Wnioski: Na podstawie wykresów oraz danych zamieszczonych w tabeli można stwierdzić, że wzrost prawdopodobieństwa krzyżowania powoduje mniejsze wahania najlepszej wartości funkcji celu oraz to że wartości funkcji celu mieszczą się w mniejszym przedziale.

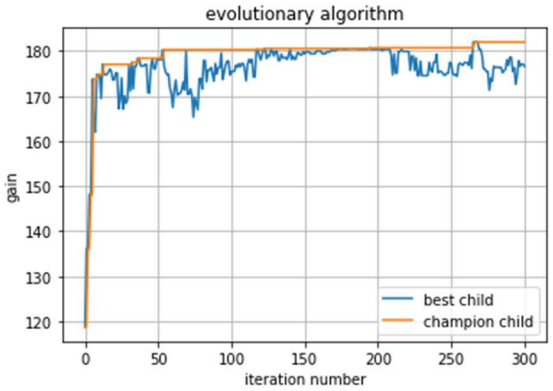
Test wpływu parametru mutacji pm

Tabela 4.

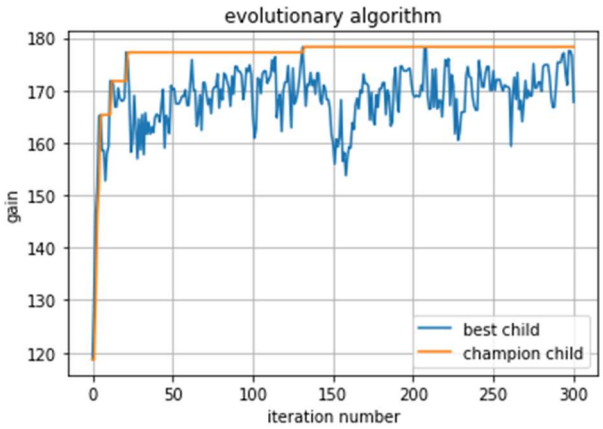
lp.	param. mutacji pm	F_{start}	F_{best}	F_{worst}	$\overline{F}$	σ	δ	I_{best}
1.	0.2	118.71	181.11	144.76	178.42	7.79	1.47	223
2.	0.5	118.71	181.98	64.18	134.71	30.67	1.00	266
3.	0.7	118.71	178.27	2.54	124.98	32.88	3.02	132
4	0.9	118.71	180.44	5.10	104.14	36.70	1.84	157



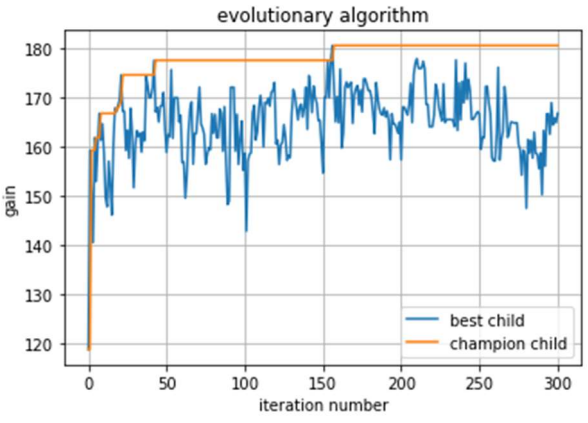
Wykres 15. Wykres dla wartości parametru pm = 0.2



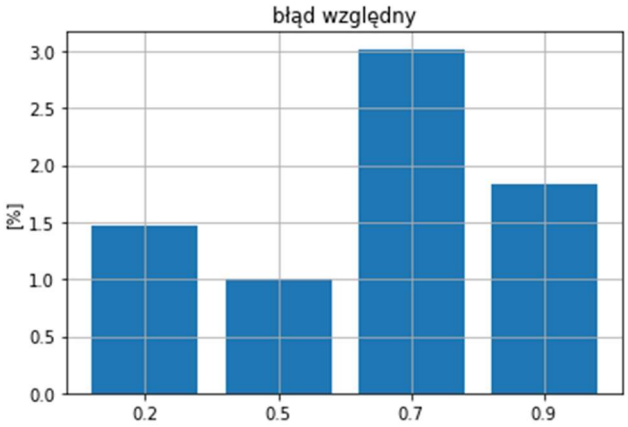
Wykres 16. Wykres dla wartości parametru pm= 0.5



Wykres 17. Wykres dla wartości parametru pm = 0.7



Wykres 18. Wykres dla wartości parametru pm = 0.9



Wykres 19. Wartości błędów względnych kolejnych testów

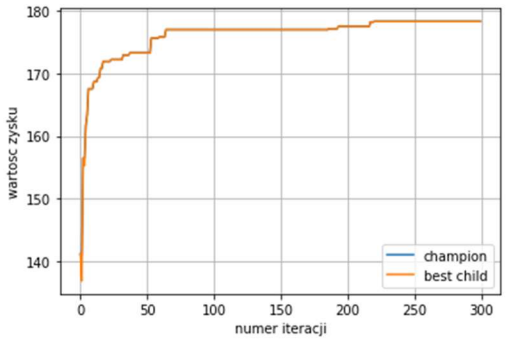
Wnioski: Dzięki mutacji mamy możliwość stworzenia osobników o zupełnie nowych wartościach, jednak zbyt wielkie prawdopodobieństwo mutacji wpływa niekorzystnie na wyniki symulacji, wprowadza zbyt dużą losowość, a najlepsza wartość funkcji nie poprawia się.

Test wpływu rodzaju krzyżowania - krzyżowanie jednopunktowe i równomierne

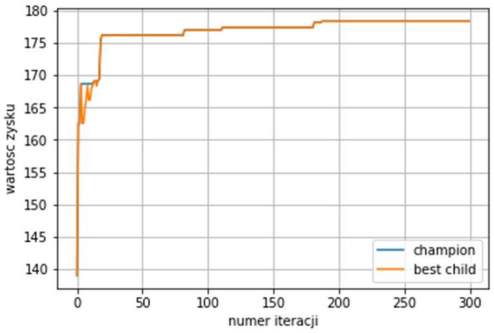
Niżej przeprowadzono test dla zbioru testowego niezłośliwego.

Tabela 5.

lp.	krzyżowanie	F_{start}	F_{best}	F_{worst}	$\overline{F}$	σ	δ	I_{best}
1.	pół na pół	118.71	178.37	119.90	170.04	17.14	2.97	187
2.	losowe	118.71	178.34	145.72	176.62	6.88	2.98	220



Wykres 20. Krzyżowanie równomierne.

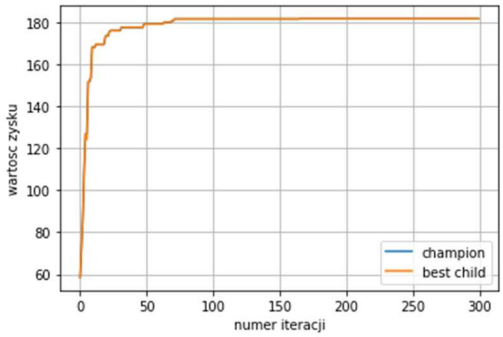


Wykres 21. Krzyżowanie jednopunktowe.

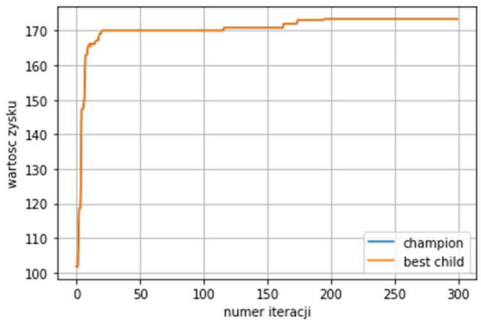
Następnie przeprowadziłyśmy te same testy dla zbioru testowego złośliwego.

Tabela 6.

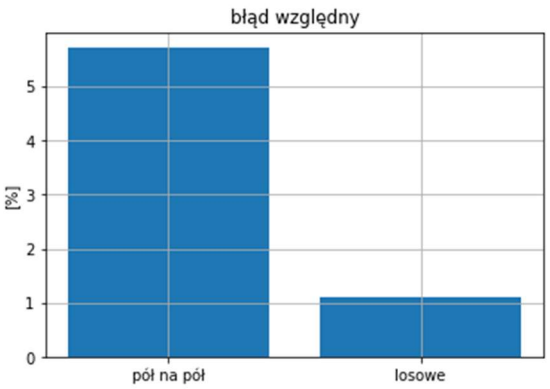
lp.	krzyżowanie	F_{start}	F_{best}	F_{worst}	$\overline{F}$	σ	δ	I_{best}
1.	pół na pół	9.96	173.32	128.58	169.89	9.14	5.71	195
2.	losowe	9.96	181.77	149.05	179.02	6.75	1.12	175



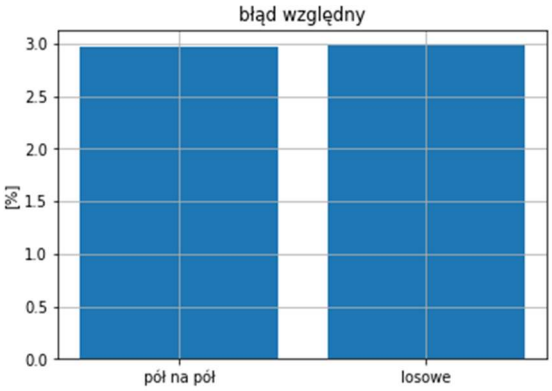
Wykres 22. Krzyżowanie równomierne.



Wykres 23. Krzyżowanie jednopunktowe.



Wykres 24. Wartości błędów względnych kolejnych testów dla instancji testowej.



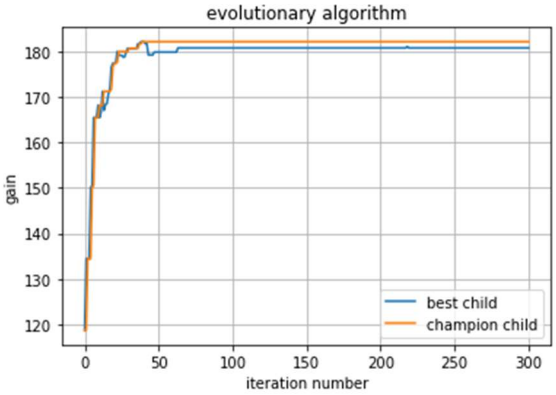
Wykres 25. Wartości błędów względnych kolejnych testów dla instancji testowej złośliwej.

Wnioski: Oba typy krzyżowania zachowują się podobnie dla zbioru łagodnego, rodzaj losowy działa trochę wolniej. Natomiast dla zbioru złośliwego funkcja krzyżowania po połowie nie osiąga tak dobrego wyniku jak druga.

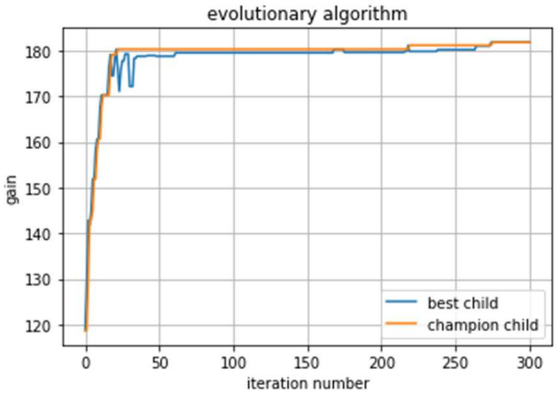
Test wpływu ilości zmutowanych genów podczas mutacji

Tabela 7.

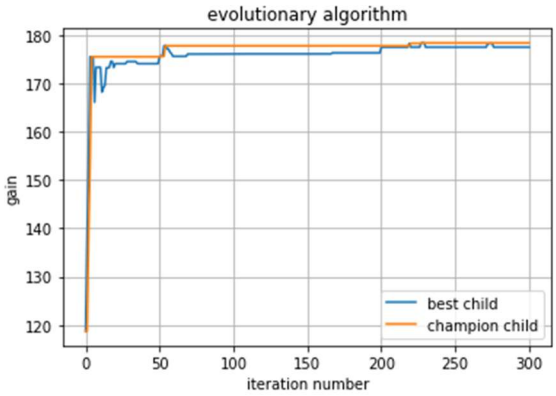
lp.	ilość mutowanych genów	F_{start}	champion	F_{best}	F_{worst}	$\overline{F}$	σ	δ	I_{best}
1.	1	118.71	182.14	183.83	124.08	175.55	13.75	0	39
2.	2	118.71	181.80	180.78	87.80	174.46	17.46	1.65	275
3.	3	118.71	178.39	178.39	68.46	170.49	22.97	2.95	228
4	4	118.71	179.72	179.72	-8.00	175.49	26.79	2.23	287



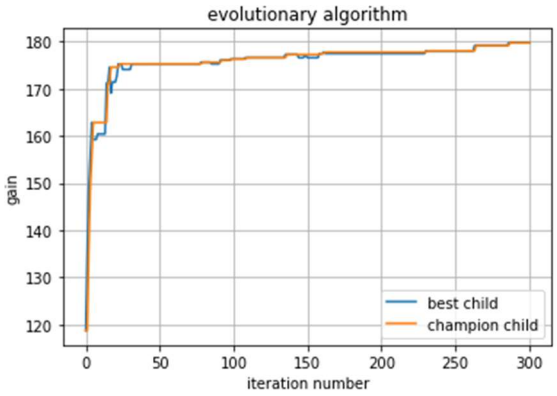
Wykres 26. Wykres dla ilości mutowanych genów = 1.



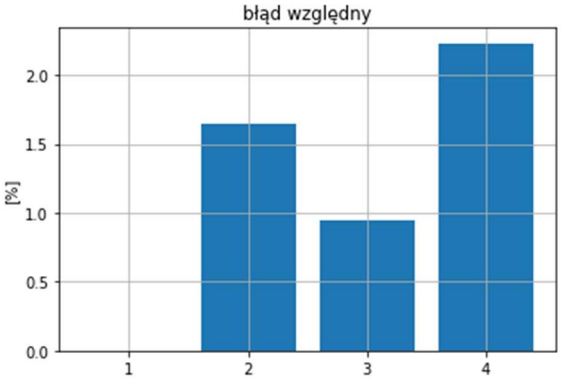
Wykres 27. Wykres dla ilości mutowanych genów = 2.



Wykres 28. Wykres dla ilości mutowanych genów = 3



Wykres 29. Wykres dla ilości mutowanych genów = 4



Wykres 30. Wartości błędów względnych kolejnych testów.

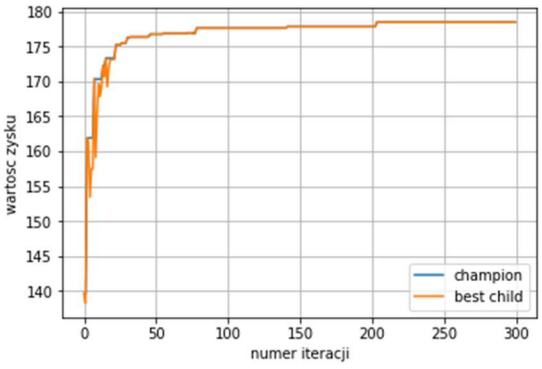
Wnioski: podobnie jak w przypadku prawdopodobieństwa mutacji, większa ilość zmutowanych osobników pozwala na dokładniejsze przeszukanie, lecz wprowadza większą losowość.

Test wpływu ilości osobników turnieju

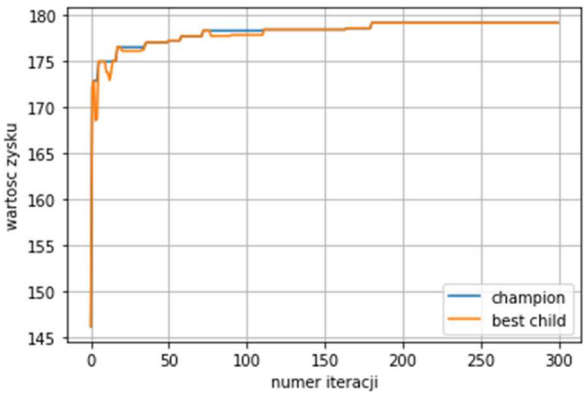
Testy na zbiorze testowym niezłośliwym.

Tabela 8.

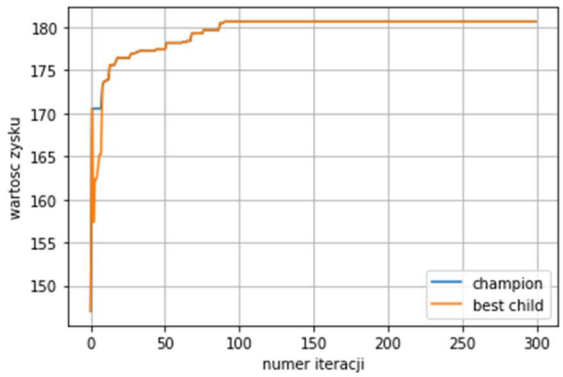
lp.	ilość osobników	F_{start}	F_{best}	F_{worst}	$\overline{F}$	σ	δ	I_{best}
1.	2	118.71	178.46	112.97	174.98	11.73	2.92	203
2.	3	118.71	179.08	96.29	167.67	27.16	2.58	180
3.	6	118.71	180.61	133.32	170.00	15.55	1.75	90
4.	8	118.71	182.06	115.92	173.84	18.05	0.96	164



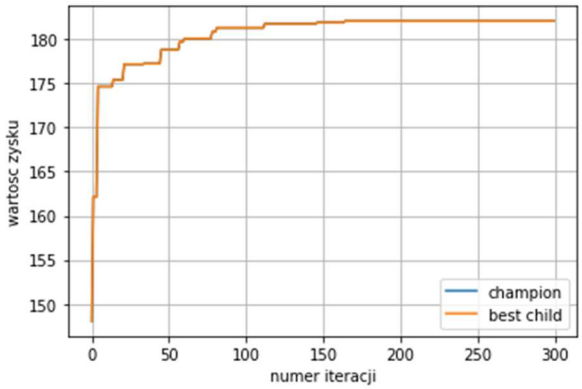
Wykres 31. Wykres dla ilości osobników w turnieju równej 2.



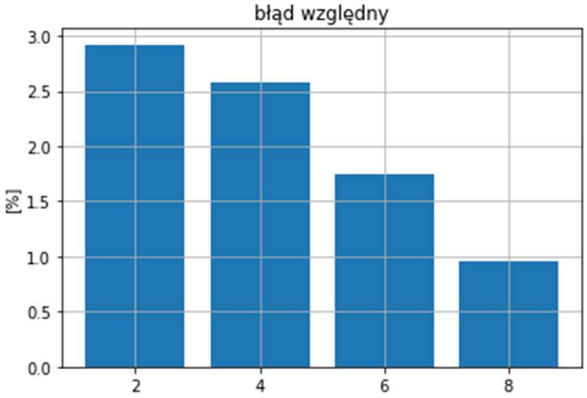
Wykres 32. Wykres dla ilości osobników w turnieju równej 3.



Wykres 33. Wykres dla ilości osobników w turnieju równej 6.



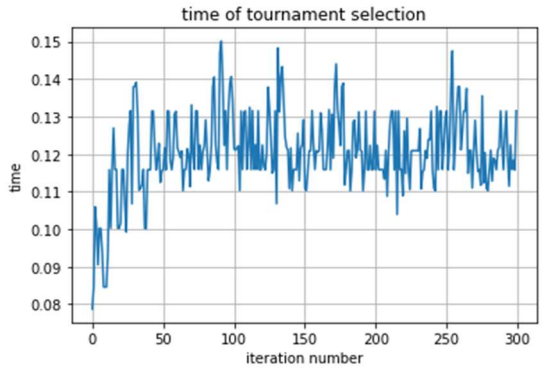
Wykres 34. Wykres dla ilości osobników w turnieju równej 8.



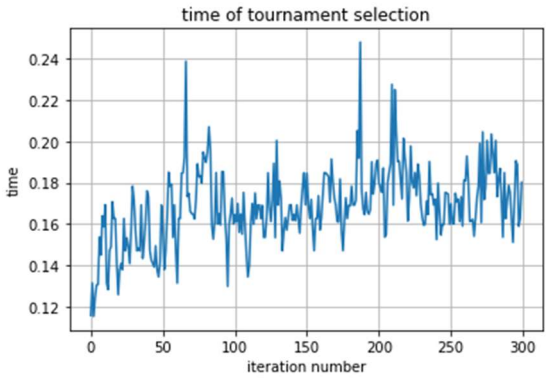
Wykres 35. Wartości błędów względnych kolejnych testów.

Wnioski: Dla mniejszej ilości osobników porównywanych wartości funkcji celu są niższe od bardzo dobrej i osiągają go po większej ilości iteracji. Dla turniejów z większą ilością uczestników wynik jest lepszy i nieco słabszy.

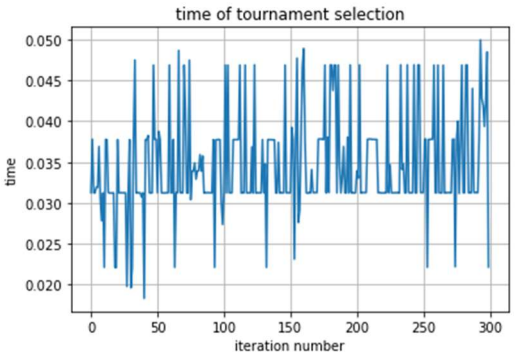
Test zależności złożoności czasowej turnieju od jego rozmiaru



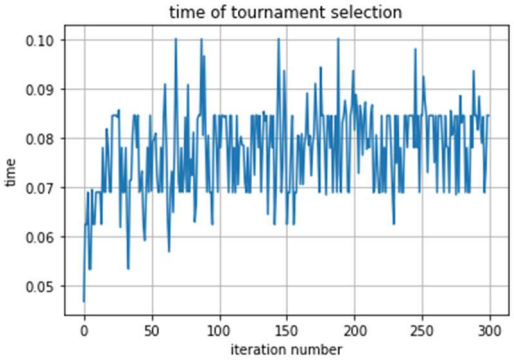
Wykres 36. Wykres rozmiaru turnieju = 2



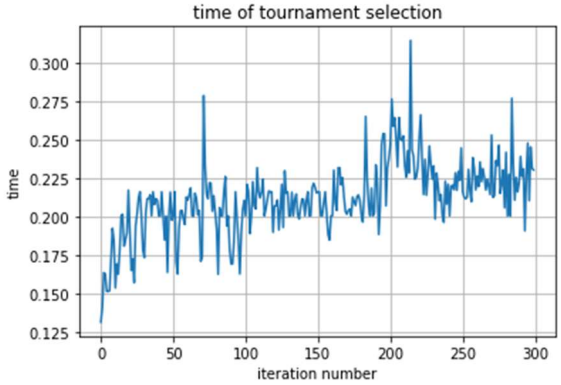
Wykres 37. Wykres rozmiaru turnieju = 4



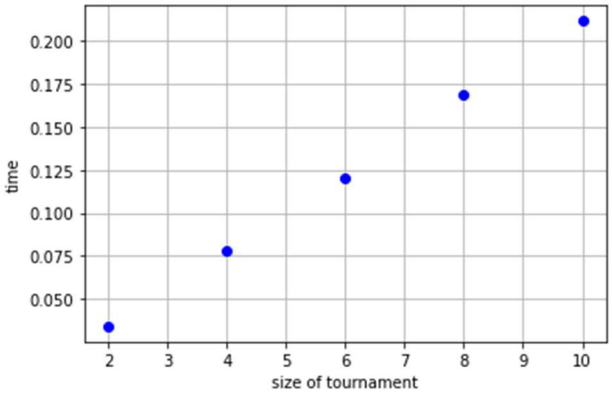
Wykres 38. Wykres rozmiaru turnieju = 6



Wykres 39. Wykres rozmiaru turnieju = 8



Wykres 40. Wykres rozmiaru turnieju = 8



Wykres 41. Wykres przedstawiający liniową zależność czasu od wielkości turnieju.

Wnioski

- Algorytm dobrze sobie radzi z wyliczeniem dobrych zysków piekarni w krótkim czasie.
- Przeprowadzone testy udowodniły, że tak naprawdę najbardziej wykorzystywany jest operator krzyżowania losowego. Z tego powodu można by pomyśleć nad dodatkowymi operatorami. Może któryś z nich przyniósłby lepsze rozwiązania.
- Turniej okazał się lepszą metodą selekcji dla naszego przypadku - umożliwiał dostanie się do następnego pokolenia słabszych osobników, które mogły ewoluować w przyszłych iteracjach na lepsze rozwiązania.
- Instancja początkowa ma bardzo mały wpływ na wynik naszego algorytmu. Po wykonaniu testów na zbiorach, które przedstawiłyśmy wyżej, stwierdzamy, że zarówno dla instancji testowej jak i złośliwej wyniki funkcji celu są przybliżone i zawsze wahają się między wartościami 175,25 a 183,83. W większości przypadków odchylenie nie przekracza 3%.
- W przyszłości można pomyśleć nad rozwinięciem projektu - poprzez wyliczanie zysków z kolejnych miesięcy , magazynowanie składników oraz obmyślenie strategii piekarni.