**Jędrzej Kuczyński**

**Zaawansowane Programowanie**

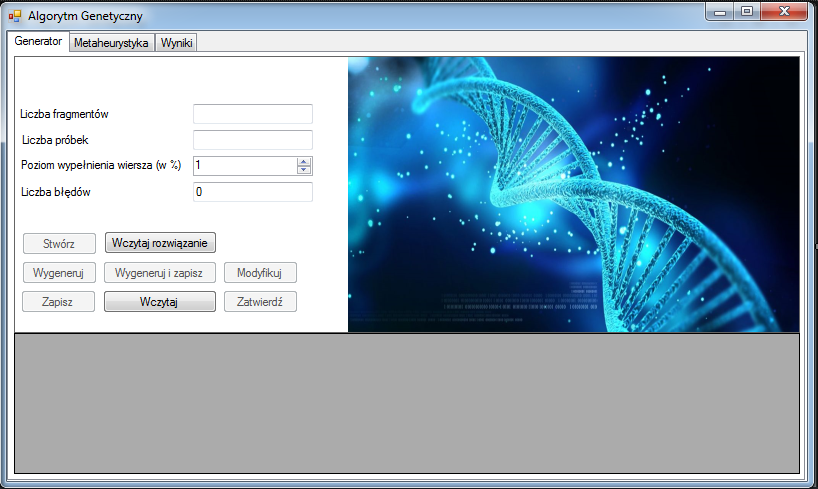
**Sprawozdanie - problem 4**

**1. Program**

**1.2. Opis interfejsu**

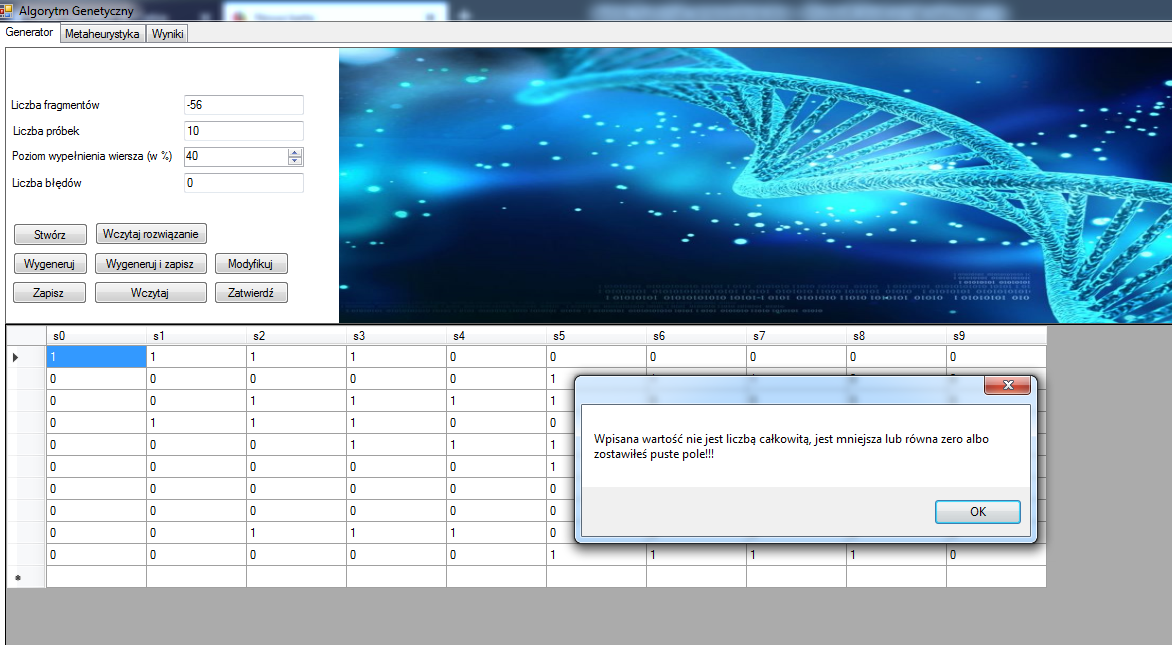
Interfejs programu składa się z jednego okna zawierającego trzy zakładki: *generator, metaheurystyka oraz wyniki*.

### *Generator*

**

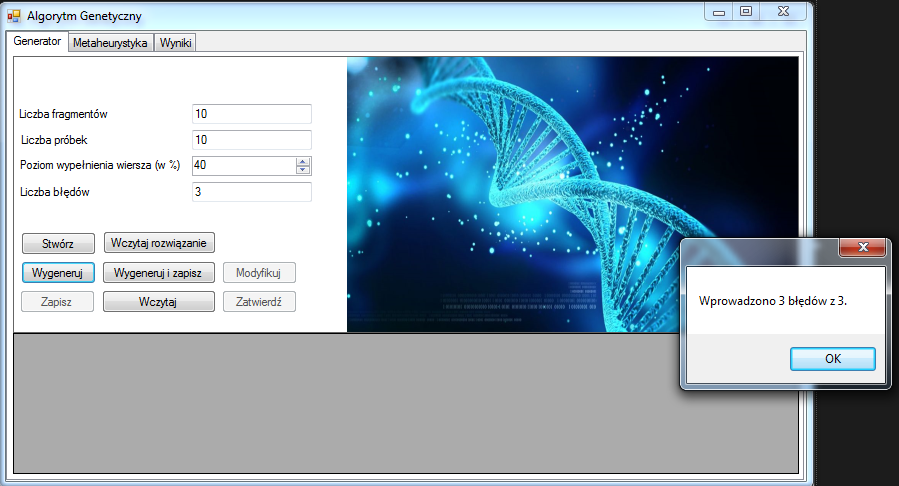
Rysunek 1. Zakładka generatora.

Kiedy program jest uruchamiany można jedynie wczytać instancję lub gotowe rozwiązanie odpowiednio z pliku XML lub CSV. Reszta przycisków staje się dostępna po poprawnym wypełnieniu wszystkich wymaganych pól. Przycisk *Stwórz* pozwala wypełnić macierz samodzielnie po wprowadzeniu liczby fragmentów i próbek. W przypadku wpisania złej wartości w którekolwiek z pól program wyświetli stosowny komunikat:



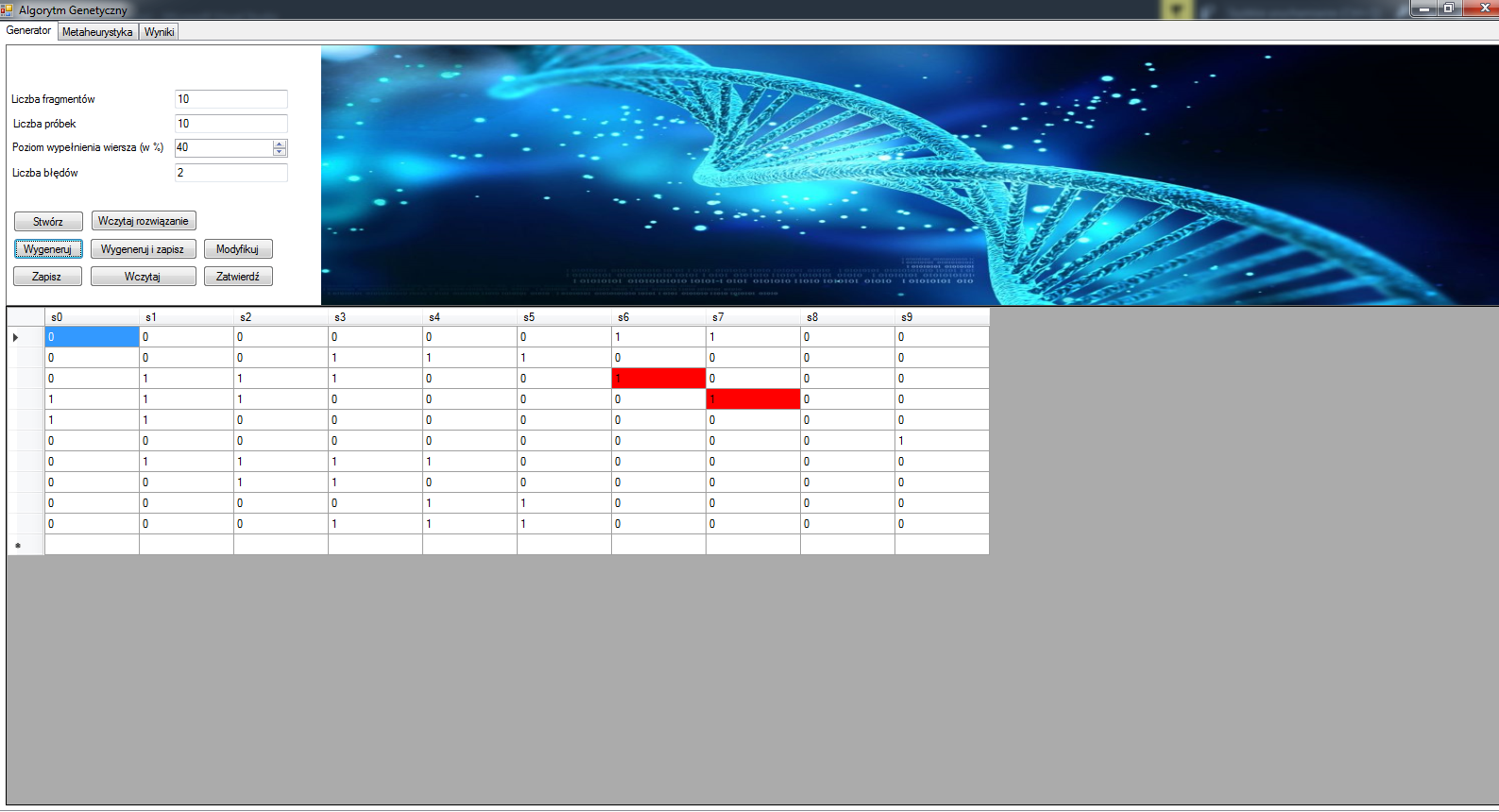
Rysunek 2. Reakcja generatora na źle wprowadzane wartości.

Po wciśnięciu przycisku *Wygeneruj* pojawia się komunikat o liczbie wprowadzonych błędów (ze względu na dany poziom wypełnienia wiersza oraz losowość generatora nie zawsze można wprowadzić tyle błędów, ile użytkownik sobie zażyczył):



Rysunek 3. Wprowadzanie błędów.

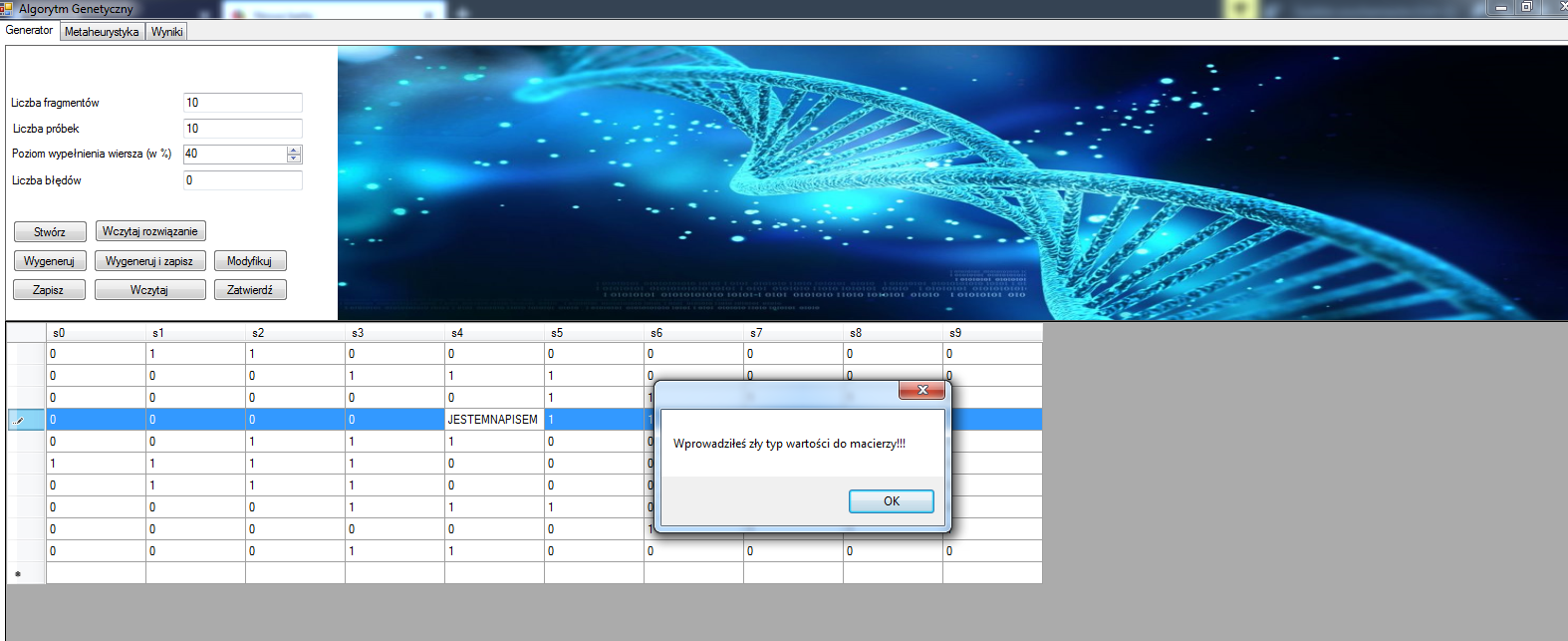
A następnie wyświetlana jest wygenerowana instancja. Na czerwono zaznaczone są komórki, w które generator wprowadził błędy:



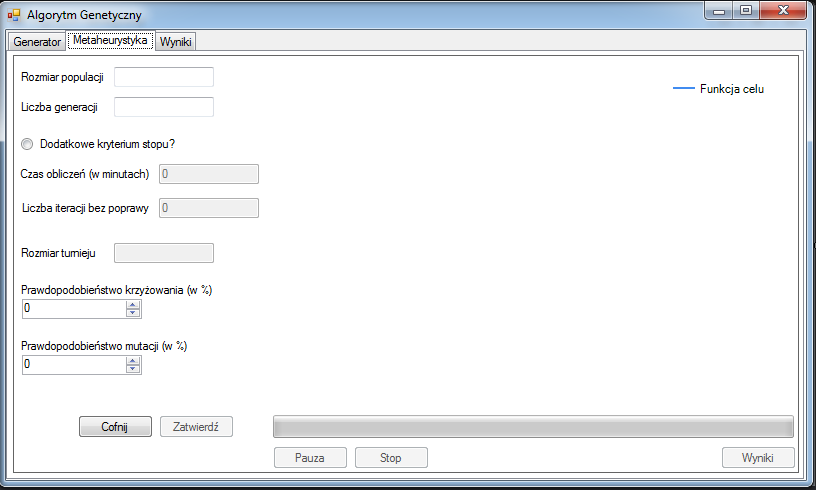
Rysunek 4. Prezentacja instancji.

Po wciśnięciu przycisku *Modyfikuj* użytkownik może ręcznie wprowadzać zmiany w tabelce.

Przycisk *Zatwierdź* zapisuje daną instancję i przenosi użytkownika do zakładki z parametrami algorytmu. W przypadku złych wartości w macierzy wyświetla stosowny komunikat:

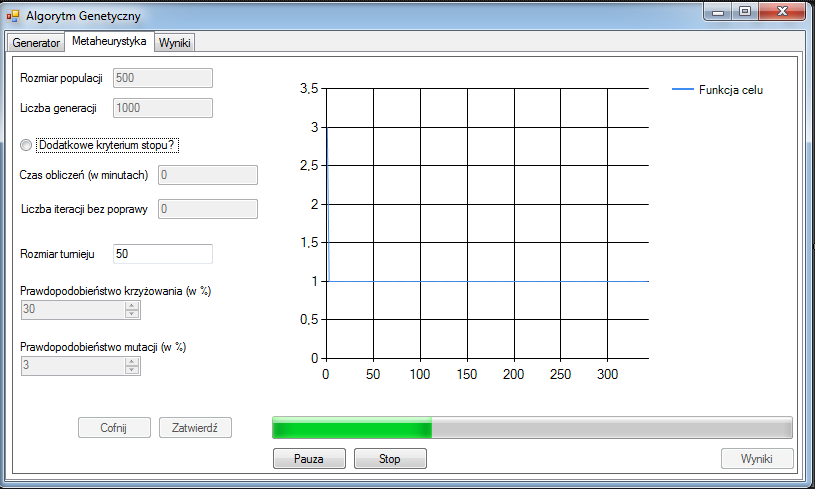


Rysunek 5. Kolejna reakcja na źle wprowadzane wartości.

*Metaheurystyka* 

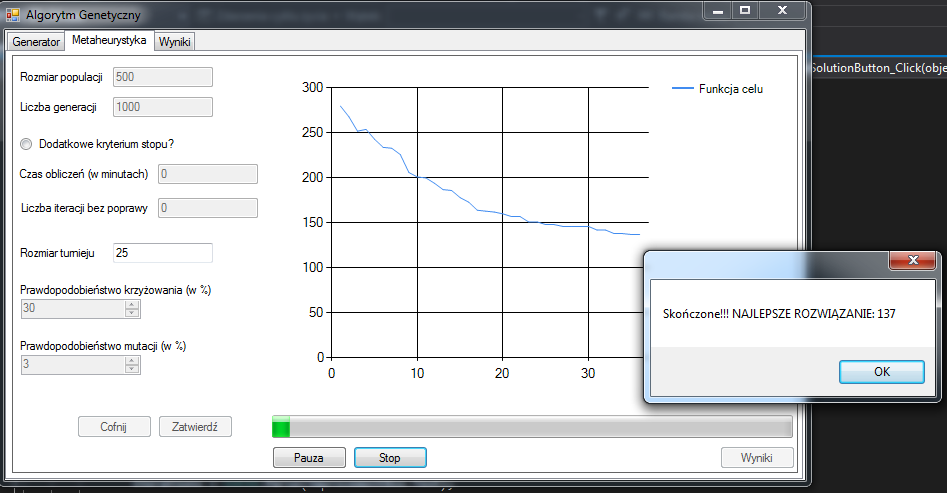
Rysunek 6. Zakładka algorytmu.

W tej zakładce sytuacja prezentuje się podobnie jak w poprzedniej. Dodatkowy przycisk *Cofnij* pozwala użytkownikowi wrócić do zakładki generatora, aby zmienić instancję. Ponadto użytkownik może wybrać dodatkowe kryterium stopu: ograniczyć czas obliczeń lub po ilu iteracjach bez poprawy funkcji celu algorytm ma skończyć działanie i zwrócić aktualny wynik. Użytkownik może wybrać oba z nich lub tylko jedno - jeżeli dane kryterium go nie interesuje musi wpisać 0 lub zostawić puste pole. Pole *Rozmiar turnieju* staje się dostępne po wprowadzeniu rozmiaru populacji - turniej nie może być liczniejszy niż sama populacja. Po wprowadzeniu wszystkich wymaganych parametrów i wciśnięciu przycisku *Zatwierdź* algorytm rozpoczyna działanie i na bieżąco informuje użytkownika o postępach obliczeń aktualizując wykres:



Rysunek 7. Działanie algorytmu.

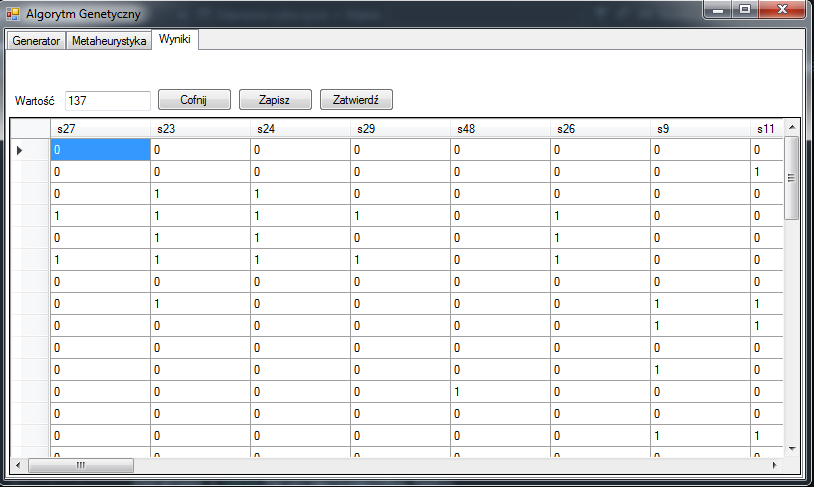
Użytkownik może również skorzystać z funkcjonalności przycisków *Pauza* lub *Stop*. Po skończeniu obliczeń lub wciśnięciu przycisku *Stop* algorytm kończy działanie a program wyświetla odpowiedni komunikat:



Rysunek 8. Zakończenie obliczeń.

Wciśnięcie przycisku *Wyniki* przenosi użytkownika do zakładki prezentującej rozwiązanie.

***Wyniki***



Rysunek 9. Prezentacja wyników.

Ta zakładka prezentuje tabelkę z otrzymanym rozwiązaniem oraz udostępnia funkcje zapisu rozwiązania do pliku CSV oraz cofnięcia się do poprzedniej zakładki. Przycisk *Zatwierdź* sprowadza użytkownika do stanu początkowego - czyści wszelkie dane oraz przenosi do pierwszej zakładki *Generator*.

**1.3. Opis kodu i algorytmu**

***Generator - tworzenie instancji i wprowadzanie błędów***

Tworzona jest macierz o wymiarach podanych przez użytkownika: *liczba próbek* odpowiada liczbie kolumn w macierzy, a *liczba fragmentów* liczbie wierszy. Wprowadzony przez użytkownika procentowy *poziom wypełnienia wiersza* stanowi maksimum, tj. w przypadku wartości 50% dla 10 kolumn możliwą liczbą jedynek w wierszu jest 1, 2, 3, 4 lub 5 (w instancji nigdy nie występuje pusty wiersz). Następnie wprowadzane są błędy:

- dopóki generator nie wprowadził zażądanej liczby błędów, z listy dostępnych, niewykorzystanych komórek w macierzy losowana jest jedna z nich (i usuwana z listy dostępnych komórek)

- następnie generator sprawdza, w którym miejscu macierzy znajduje się wylosowana komórka: lewa skrajna, środek lub prawa skrajna

- następnie w zależności od układu zer i jedynek występujących w danym miejscu odpowiednio wprowadza "znaczący" błąd lub nie.

Dostępne układy, w których można wprowadzić "znaczący" błąd (na czerwono jest zaznaczone miejsce, w które wprowadzany jest błąd):

Lewy skrajny: **0**01, **0**00

Środek: 1**1**1, 0**0**0

Prawy skrajny: 10**0**, 00**0**

- ze względu na dany poziom wypełnienia macierzy i dany układ w niej występujący (wynikający z losowości generatora), generator nie zawsze może wprowadzić tyle błędów, ile użytkownik sobie zażyczył. Dlatego losowanie kolejnych komórek odbywa się dopóki w macierzy występują "niewykorzystane" komórki.

***klasa Individual***

Klasa *Individual* jest reprezentacją osobnika. Posiada jego genom, wartość przystosowania (która jest utożsamiana z funkcją celu) oraz tabelę z rozwiązaniem, które genom osobnika reprezentuje. Ponadto metoda *calculate\_fitness* oblicza wartość przystosowania:

***Obliczanie przystosowania***

Dla każdego wiersza w tabeli:

- metoda zamienia układ zer i jedynek na ciąg literowy i usuwa skrajne zera:

001110011101000 --> "1110011101"

- następnie "mapuje" ciąg literowy na listę, która zawiera liczby reprezentujące występowanie danej cyfry (liczby reprezentujące jedynki i zera występują na niej naprzemiennie):

"1110011101" --> [3, 2, 3, 1, 1]

- obliczana jest liczba zmian, które należy wykonać, aby sprowadzić wiersz do "consecutive ones". Iterując po "jedynkach" (co drugą wartość) metoda sprawdza, czy liczba zer (znajdujących się na prawo od aktualnej jedynki) jest mniejsza od liczby samych jedynek. Jeżeli tak, to ta liczba jest dodawana do wartości przystosowania. W przeciwnym przypadku dodawana jest liczba jedynek (pomarańczowy - aktualne "jedynki", czerwony - aktualne "zera")

[**3**, **2**, 3, 1, 1] --> fitness += 2

[3, 2, **3**, **1**, 1] --> fitness += 1

Kiedy pętla dojdzie do ostatniej wartości w liście (liczbie jedynek) możliwe są trzy scenariusze:

- liczba ostatnich jedynek nie jest mniejsza od liczby zer, znajdującej się na lewo: algorytm nie podejmuje działań (nadal opłaca się bardziej zamienić zera na jedynki):

[3, 2, 3, **1**, **1**] --> fitness = 3

- w przeciwnym przypadku:

- jeżeli liczba zer jest większa lub równa liczbie poprzednich jedynek oraz liczba ostatnich jedynek jest mniejsza od liczby poprzednich jedynek, bardziej opłacalnym jest zamienić skrajne jedynki na zera (niebieski - liczba poprzednich jedynek):

[..., **2**, **2**, **1**] --> fitness -= 2, fitness += 1

- jeżeli liczba zer nie jest większa lub równa liczbie poprzednich jedynek od wartości przystosowania odejmowana jest różnica pomiędzy liczbą zer a liczbą ostatnich jedynek (również bardziej opłacalnym jest zamienić skrajne jedynki na zera):

[..., **7**, **5**, **3**] --> diff = 2, fitness -= 2

***Algorytm genetyczny (klasa GA)***

***inicjalizacja populacji***

Inicjalizacja populacji następuje w konstruktorze klasy *GA*. Dla każdego osobnika losowany jest genom z kolumn macierzy wejściowej - lista kolumn zostaje przetasowana algorytmem Fisher'a-Yates'a.

***selekcja - turniej***

Ponieważ wartość przystosowania zależy od układu kolumn i danego układu zer i jedynek z niego wynikających w każdym momencie "dobre" rozwiązanie może stać się "złym". Dlatego w algorytmie wybrana została metoda selekcji, która daje dużą szansę na wybór osobnikom słabszym.

***krzyżowanie - z zachowaniem porządku***

Krzyżowanie z zachowaniem porządku zachowuje właściwości permutacji, dlatego zostało ono wybrane.

***mutacje***

W algorytmie występują trzy typy mutacji:

- przestawienie danej kolumny w inne miejsce

- zamiana dwóch kolumn miejscami

- odwrócenie danego podciągu kolumn

**2. Testy i wyniki**

**2.1. Testy**

Algorytm był testowany z następującymi kombinacjami parametrów:

- na symetrycznych macierzach o rozmiarach: 10x10, 15x15 i 20x20 (testy rozmiaru 20x20 musiały zostać przerwane)

- wypełnieniu wiersza: 10%, 30%, 50% (macierze 10x10 bez wypełnienia 10% ze względu na poprawność każdej instancji)

- liczbie błędów równej 0 oraz 20%, 40%, 60%, 80% i 100% z 20% dostępnych komórek (np. dla macierzy 10x10 liczba błędów: 0, 4, 8, 12, 16, 20)

- rozmiarach populacji:

10x10: 50, 100

15x15: 50, 100

20x20: 50, 100, 200

- liczbie generacji:

10x10: 50, 100, 200

15x15: 50, 100, 200, 300

20x20: 50, 100, 200, 300, 400

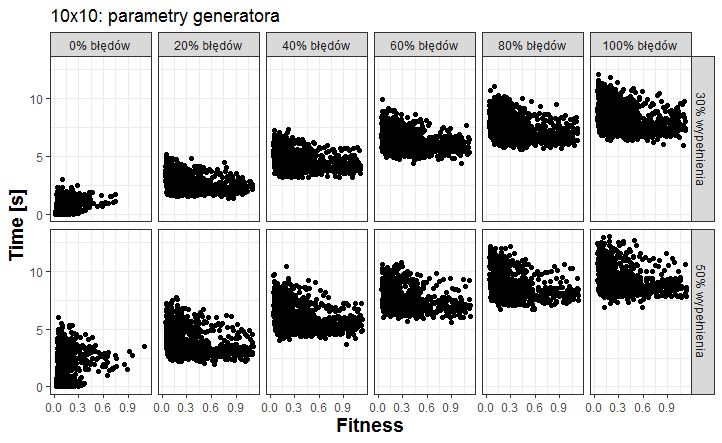
- rozmiarze turnieju równym 10%, 30% oraz 50% rozmiaru populacji

- prawdopodobieństwu crossing-over: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9

- prawdopodobieństwu mutacji: 0, 0.02, 0.04, 0.06, 0.08, 0.1

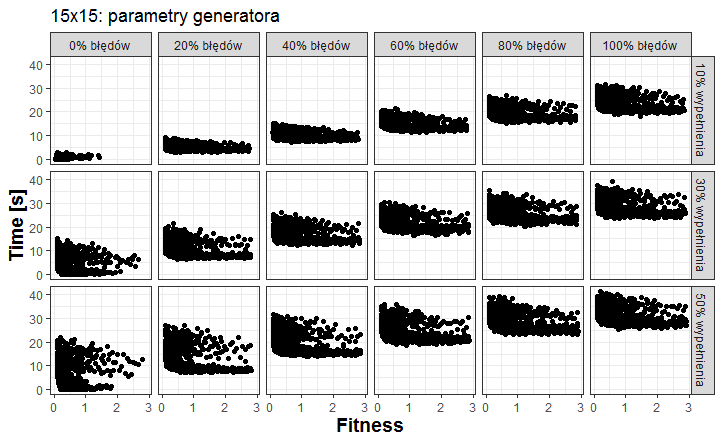
**2.2. Wyniki**

***Parametry generatora***



Wykres 1. Wykres prezentujący zależności pomiędzy parametrami generatora a czasem działania algorytmu i wartością dopasowania dla macierzy rozmiaru 10x10 (liczba błędów: 0, 4, 8, 12, 16, 20).

Z powyższego wykresu widać wyraźnie, że największy wpływ zarówno na czas obliczeń, jak i wartość dopasowania rozwiązania ma liczba błędów. Natomiast poziom wypełnienia wiersza niespecjalnie wpływa na czas obliczeń i wartość przystosowania.



Wykres 2. Wykres prezentujący zależności pomiędzy parametrami generatora a czasem działania algorytmu i wartością dopasowania dla macierzy rozmiaru 15x15 (liczba błędów: 0, 9, 18, 27, 36, 45).

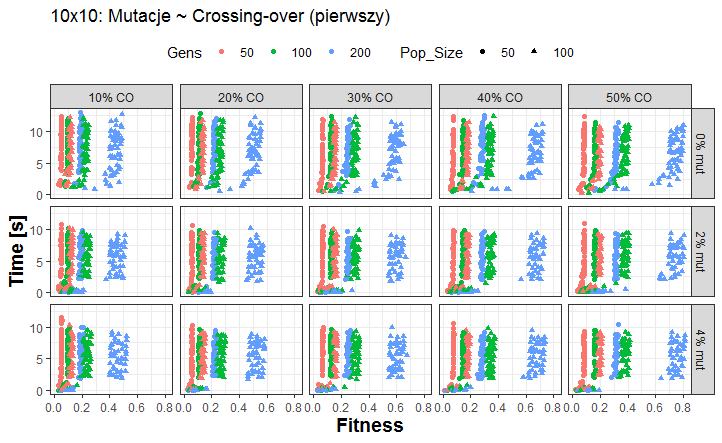
W przypadku macierzy rozmiaru 15x15 również największy wpływ na czas działania algorytmu, jak i wartość przystosowania rozwiązania ma liczba błędów. Jednakże w przypadku większej instancji poziom wypełnienia wiersza ma większy wpływ na czas obliczeń - im niższy poziom wypełnienia wiersza, tym szybszy czas obliczeń. Może to wynikać z mniejszej liczby dostępnych kombinacji, im mniej jedynek w wierszu.

***Parametry heurystyki***

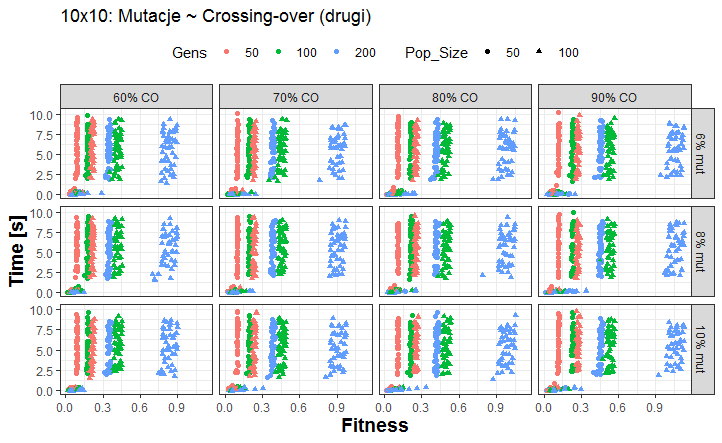
W celu ustalenia, które parametry najbardziej wpływają na wynik i czas działania algorytmu wykonana została analiza regresji metodą *Random Forest*.

Analiza wykazała, że największy wpływ na wartość przystosowania rozwiązania ma liczba błędów (92%), poziom wypełnienia wiersza (2%) oraz mutacje (2%):

Rozkład "ważności" parametrów ze względu na przystosowanie: [Fragmenty: 0.006, Próbki: 0.006, Wypełnienie: 0.023, Błędy: 0.927, Populacja: 0.003, Generacje: 0.004, Turniej: 0.002, Crossing-over: 0.005, Mutacja: 0.024]



Wykres 3. Wykres prezentujący zależność czasu obliczeń algorytmu i wartości przystosowania rozwiązania od prawdopodobieństwa krzyżowania i prawdopodobieństwa mutacji (kolorami i kształtami zaznaczono liczbę generacji i wielkość populacji). Część 1.



Wykres 4. Wykres prezentujący zależność czasu obliczeń algorytmu i wartości przystosowania rozwiązania od prawdopodobieństwa krzyżowania i prawdopodobieństwa mutacji (kolorami i kształtami zaznaczono liczbę generacji i wielkość populacji). Część 2.

Natomiast w przypadku wpływu parametrów na czas obliczeń sytuacja prezentuje się następująco:

Rozkład "ważności" parametrów ze względu na czas obliczeń: [Fragmenty: 0.054, Próbki: 0.061, Wypełnienie: 0, Błędy: 0.071, Populacja: 0.367, Generacje: 0.339, Turniej: 0.004, Crossing-over: 0.076, Mutacja: 0.028]

Największy wpływ na czas obliczeń mają wielkość populacji (36%) oraz liczba generacji (33%). Inaczej niż w przypadku wartości dopasowania pozostałe parametry również mają wpływ na czas obliczeń.