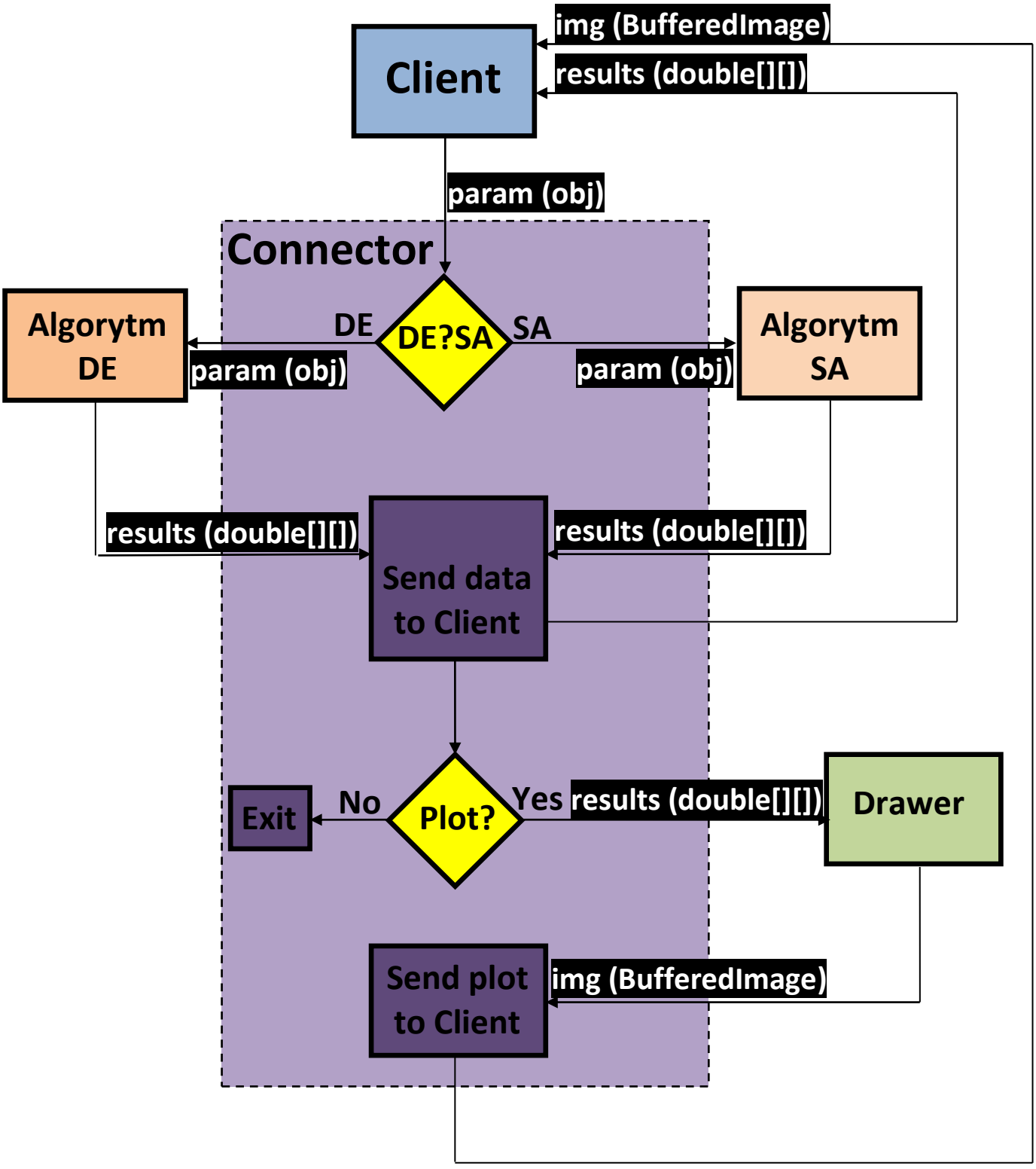
1. Implementacja algorytmów genetycznych z wizualizacją jako usługi sieciowe.

- **Algorytmy genetyczne:**
  - Genetyczny różnicowy (DE)
  - Symulowanego wyżarzania (SA)
- **Funkcje optymalizacji:**
  - RosenBrock
  - Beale
  - Both
  - Matyas
- **Zewnętrzna biblioteka (JMathPlot) do rysowania wykresów**
- **Projekt Maven**
- **Przesyłanie danych między modułami za pomocą socketów**

2. Opis ogólny

Client przekazuje parametry do Connectora. Connector na ich podstawie przesyła parametry dalej do algorytmu DE lub SA i otrzymuje wynik w postaci dwuwymiarowej tablicy liczb double. Odsyła tę tablicę do Clienta. Na podstawie parametrów kończy program lub odsyła wyniki do Drawera. Drawer rysuje wykres, odsyła obraz do Connectora, a Connector do Clienta.

3. Schemat projektu



4. Opis szczegółowy

• Common

W pakiecie są zawarte dwa typu enumeryczne: AlgorytmType i OptimalizationFunction. Do wspólnych klas należy klasa Parameters zawierająca parametry przekazywane między Clientem, Connectorem i Algorytmami: wielkość populacji, ilość pokoleń, dwa parametry dla algorytmów, parametry określające typ algorytmu(DE lub SA), funkcję optymalizacyjną i zmienną określającą czy ma być wysyłany wykres.

• Client

Client wysyła parametry i odbiera wyniki. Inicjalizuje parametry, przesyła do Connectora i odbiera wyniki jako double[][]. Wyświetla je i liczy średnią. Jeśli Client chciał wykres, odbiera go od Connectora jako BufferedImage, zapisuje na dysku i pokazuje obrazek jako JFrame.

• Connector

Connector tylko przesyła dane. Pobiera parametry od Clienta i wysyła je do algorytmu DE czy SA. Odbiera wynik jako double[][] i odsyła do Clienta. Jeśli w parametrach flaga wykresu == true, przesyła wynik do Drawera, odbiera BufferedImage i odsyła do Clienta. Jeśli ta flaga == false, kończy program.

• Algorytm DE

Generuje wyniki double[][] na podstawie parametrów. Odbiera od Connectora parametry i tworzy początkowe losowe pokolenie. Następnie odpowiednią ilość razy je mutuje, krzyżuje i dokonuje selekcji. Ostatnie pokolenie konwertuje na double[][] i wysyła do Connectora.

• Algorytm SA

Generuje wyniki double[][] na podstawie parametrów. Odbiera od Connectora parametry i tworzy początkowe losowe pokolenie. Następnie odpowiednią ilość razy losuje nowe losowe pokolenie, dokonuje selekcji i zmienia temperaturę. Ostatnie pokolenie konwertuje na double[][] i wysyła do Connectora.

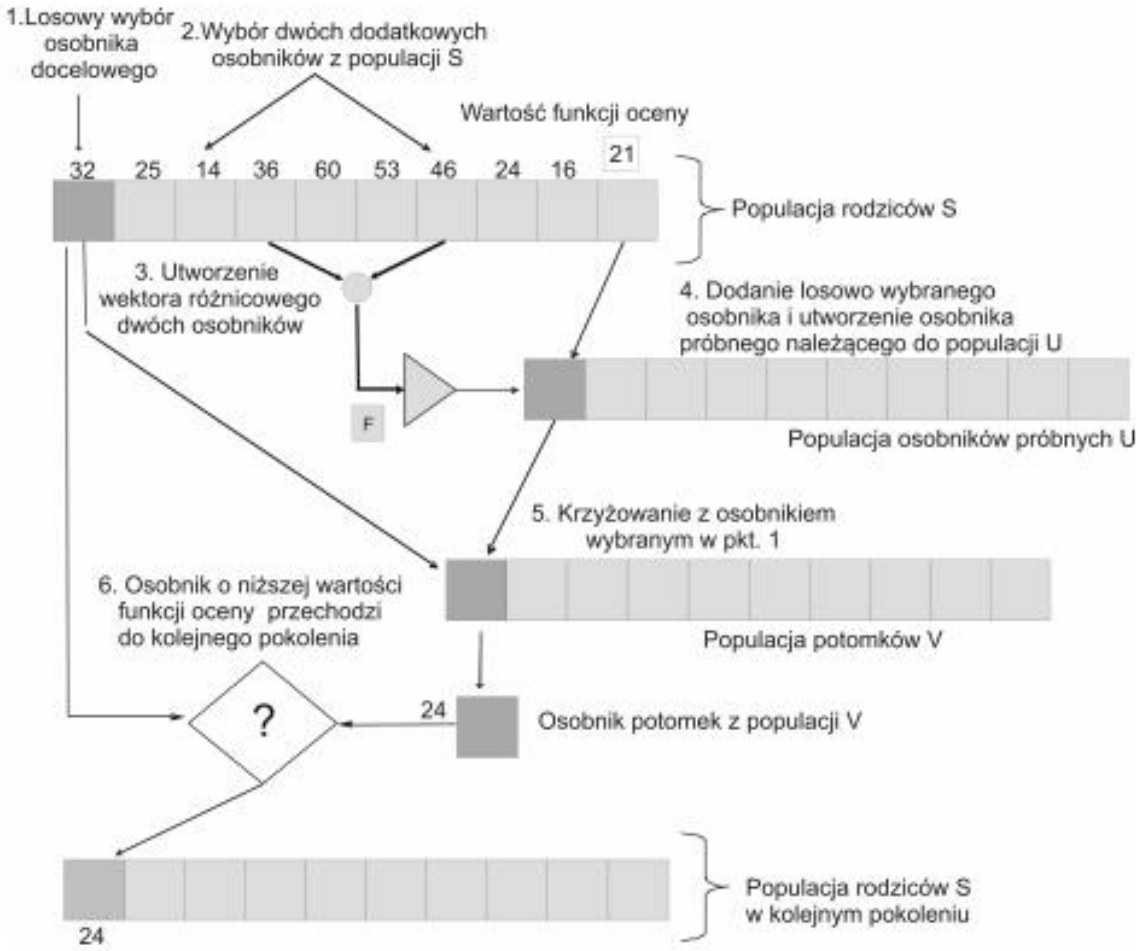
• Drawer

Rysuje wykres na podstawie wyników. Odbiera od Connectora wyniki jako double[][]. Na podstawie zewnętrznej biblioteki JMathPlot tworzy wykres i wyświetla go jako JFrame, pobiera obraz jako zrzut ekranu i wysyła go do Connectora. Wysłanie JFrame z biblioteki swing nie jest możliwe. JavaDoc:

"Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeansTM has been added to the java.beans package. Please see XMLEncoder."

5. Algorytm DE:

Schemat ewolucji różnicowej (źródło str. 14)



Rysunek 2.1: Schemat ewolucji różnicowej

Algorytm Ewolucji różnicowej najpierw pobiera od Connectora parametry. Tworzy nowy obiekt ResultGetter, który tworzy nową populację wynikową (w klasie Generation) i konwertuje wyniki na double[][]. Wyniki są przesyłane do Connectora.

Osobnik to obiekt klasy Point. Ma dwa pola double w odpowiednim DecimalFormacie. Posiada metody obliczające przystosowanie osobnika według odpowiedniej funkcji, która jest wybierana na podstawie parametrów.

Tworzenie nowej generacji:

Generacja ma trzy listy osobników (arrayListy klasy Point) oraz przekazane parametry F i CR. Najpierw losowana jest losowa populacja początkowa w zakresach zależnych od typu funkcji optymalizacyjnej. Potem odpowiednią ilość razy wykonują się kolejno:

- Mutacja
  - Każdy kolejny osobnik z listy #1 wraz z dwoma innymi, różnymi osobnikami z tej listy jest mutowany według wzoru:

    $$Ui = Pr1 + F * (Pr2 – Pr3), \text{ gdzie:}$$

    Ui – nowy osobnik dodawany do listy #2
    Pr1 – kolejny osobnik z listy #1
    Pr2 i Pr3 – losowe wybrane osobniki różne od siebie i od Pr1
    F – współczynnik przekazany w parametrach
- Krzyżowanie
  - Dla każdej pary osobników z listy #2 jest losowana liczba z przedziału (0,1). Jeśli jest większa niż parametr CR, ta para jest dodawana do listy #3. Jeśli jest mniejsza, następuje krzyżowanie: losuj locus, wykonaj operacje (gdzie pierwszyX to zmienna X pierwszego punktu)
  - pierwszyX.substring(0, locus) + drugiX.Substring(locus)
  - drugiX.substring(0, locus) + drugiX.Substring(locus).
  - pierwszyY.substring(0, locus) + drugiY.Substring(locus).
  - drugiY.substring(0, locus) + drugiY.Substring(locus).
    Wygeneruj nową parę punktów:
  - Point1 = pierwszyX, pierwszyY, Point2 = drugiX, drugiY
  - Dodaj otrzymane punkty do listy #3
- Selekcja
  - Porównywany jest każdy osobnik z listy #1 z odpowiadającym mu osobnikiem z listy #3. Obliczana jest wartość odpowiedniej funkcji. Osobnik mający wartość bliższą zero, czyli lepiej przystosowany, zostaje dodany do początkowej listy generacji #1
- Wyczyszczenie listy #2 oraz #3, powtórzenie cyklu.

6. Algorytm SA

Algorytm Ewolucji różnicowej najpierw pobiera od Connectora parametry. Tworzy nowy obiekt ResultGetter, który tworzy nową populację wynikową (w klasie Generation) i konwertuje wyniki na double[][]. Wyniki są przesyłane do Connectora.

Osobnik to obiekt klasy Point. Ma dwa pola double w odpowiednim DecimalFormacie. Posiada metody obliczające przystosowanie osobnika według odpowiedniej funkcji, która jest wybierana na podstawie parametrów.

Tworzenie nowej generacji:

Generacja ma dwie listy osobników (arrayListy klasy Point) oraz przekazane parametry: bieżącą temperaturę i mnożnik nowej temperatury. Najpierw losowana jest losowa populacja początkowa w zakresach zależnych od typu funkcji optymalizacyjnej. Potem odpowiednią ilość razy wykonują się kolejno:

- Losowanie nowej generacji do listy #2 tak jak populacji początkowej
- Wybór lepszego osobnika:
    - Porównuje się każdego osobnika z listy #1 z odpowiadającym mu osobnikiem z listy #2 na podstawie wyliczonego wyniku wybranej funkcji optymalizacyjnej
    - Jeśli osobnik listy #2 jest lepszy, przechodzi on do nowej populacji
    - Jeśli jest gorszy, losowana jest liczba z zakresu (0,1)
    - Oblicza się X:

      $X = \exp(-delta/t)$, gdzie:

      delta – różnica funkcji przystosowania osobnika z listy #2 i osobnika z listy #1

      jeśli wylosowana liczba jest mniejsza od X, osobnik z listy #2 przechodzi do nowej populacji. W przeciwnym wypadku osobnik z listy #1
- Ustawianie nowej temperatury: obecna temperatura pomnożony przez mnożnik nowej temperatury.