Systemy Wspomagania Decyzji

Temat

Sprawozdanie - Metody wielokryterialnej optymalizacji dyskretnej

Spis treści:

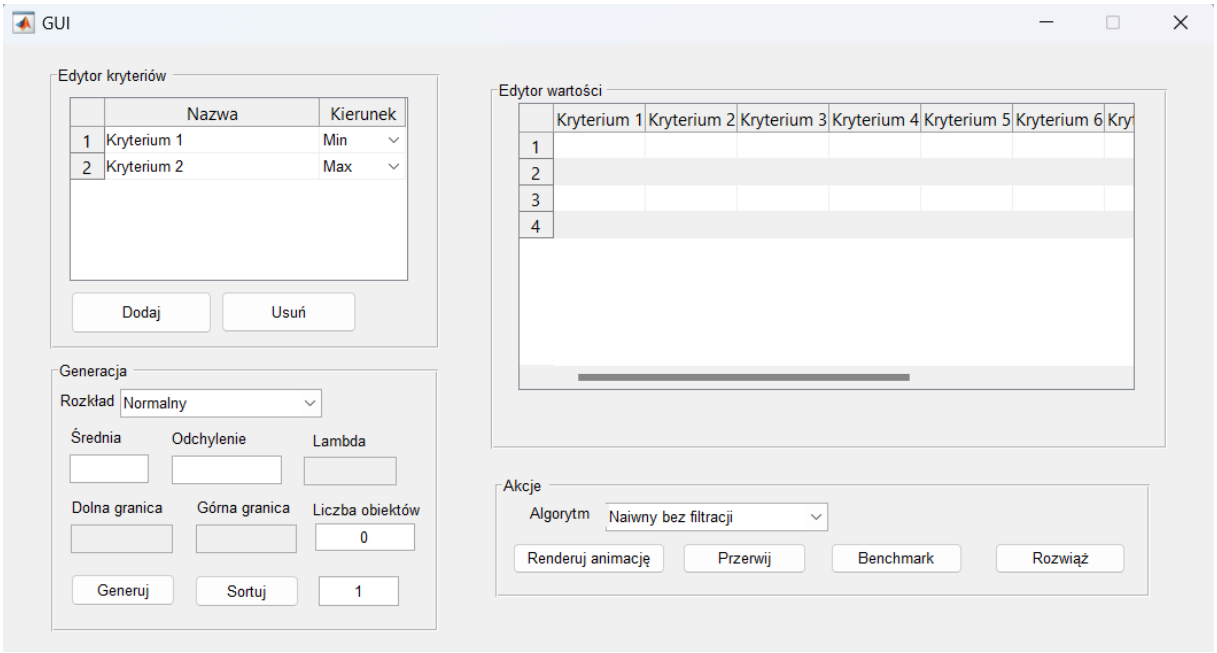
Wstęp:

Celem niniejszego projektu, który został stworzony z myślą o zaawansowanej analizie wielokryterialnej, było zaimplementowanie i przetestowanie różnych algorytmów, które miały na celu wszechstronne rozpoznanie oraz zrozumienie struktury punktów niezdominowanych w przestrzeni wielokryterialnej, a całość tych działań odbywała się z wykorzystaniem środowiska programistycznego MATLAB, które stanowi doskonałe narzędzie do prowadzenia skomplikowanych analiz matematycznych i graficznych. Projekt zawierał również zaawansowany graficzny interfejs użytkownika (GUI), który miał na celu nie tylko ułatwienie obsługi samego programu, ale także zapewnienie, że każdy, kto posługuje się narzędziem, będzie mógł w intuicyjny sposób przemieszczać się między różnymi jego funkcjonalnościami, co znacząco przyczyniło się do efektywności procesu analizy i interakcji z danymi. W ramach projektu uwzględniono implementację trzech algorytmów, spośród których pierwszy był algorytmem naiwnym bez filtracji, mającym na celu zaprezentowanie prostej, lecz nieoptymalnej metody określania punktów niezdominowanych, drugi algorytm był także naiwnym, lecz zastosowano w nim dodatkową filtrację, która miała za zadanie zredukować liczbę analizowanych punktów w celu optymalizacji procesu wyszukiwania, a trzeci algorytm, będący przygotowaniem danych dla algorytmu Luccio-Prepata, miał na celu zademonstrowanie bardziej zaawansowanego podejścia do problemu wielokryterialnego, które obejmowało przekształcanie oraz wstępne przetwarzanie danych w sposób umożliwiający ich efektywne przetworzenie przez algorytm docelowy. Kluczowym zadaniem, na którym skupiały się wszystkie działania w projekcie, było wyznaczenie punktów niezdominowanych w przestrzeni wielokryterialnej, stanowi to jedno z najistotniejszych zagadnień w analizie tego rodzaju, ponieważ identyfikacja takich

punktów pozwala na lepsze zrozumienie kompromisów między różnymi kryteriami, które mogą być stosowane do podejmowania decyzji, co jest szczególnie istotne w kontekście problemów decyzyjnych, gdzie każdy z punktów reprezentuje pewne rozwiązanie, które jest korzystne w różnych aspektach, a odnalezienie punktów niezdominowanych oznacza wskazanie tych, które nie są w żaden sposób gorsze od innych w każdym z rozważanych kryteriów, co z kolei ma ogromne znaczenie dla efektywnej optymalizacji decyzji oraz wyznaczenia rozwiązań preferowanych z punktu widzenia decydenta.

Opis programu:

Program składa się z graficznego interfejsu użytkownika (GUI) oraz trzech algorytmów implementujących różne podejścia do analizy wielokryterialnej, przy czym GUI umożliwia interakcję z użytkownikiem, co jest kluczowe dla poprawnego działania programu, ponieważ zapewnia ono łatwy dostęp do wszystkich funkcji, takich jak wczytywanie danych, uruchamianie algorytmów, a także prezentację wyników w formie tabel i wykresów, które stanowią czytelny sposób wizualizacji wyników, pozwalający na szybkie zrozumienie i porównanie różnych rozwiązań.

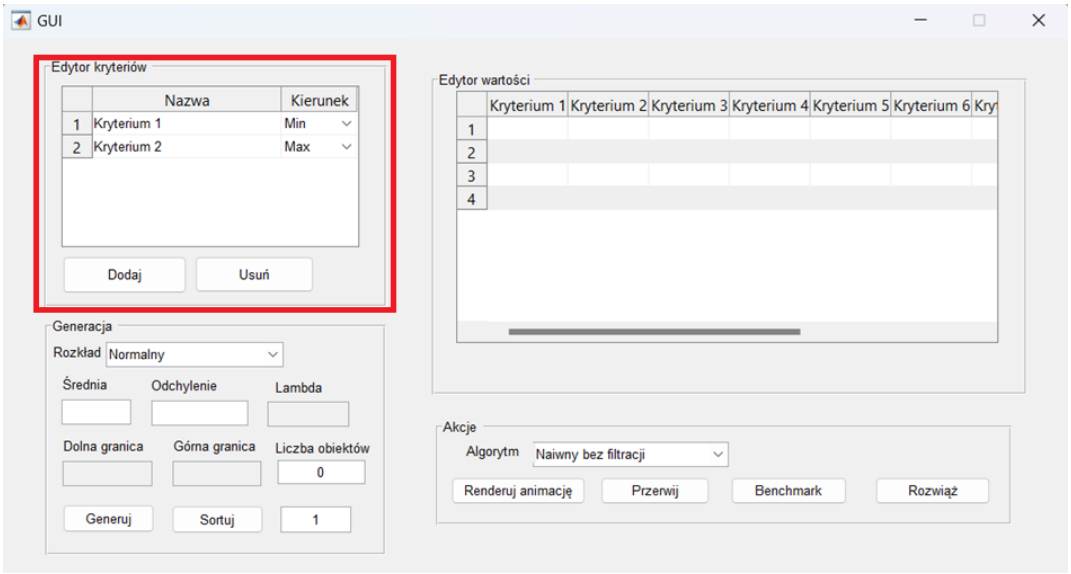


Rys.1 Panel graficzny użytkownika

Interfejs użytkownika programu jest podzielony na cztery główne sekcje, które organizują funkcjonalności w sposób przejrzysty i intuicyjny, co ma kluczowe znaczenie w przypadku skomplikowanych analiz wielokryterialnych, ponieważ pozwala użytkownikowi na swobodne poruszanie się między różnymi elementami programu.

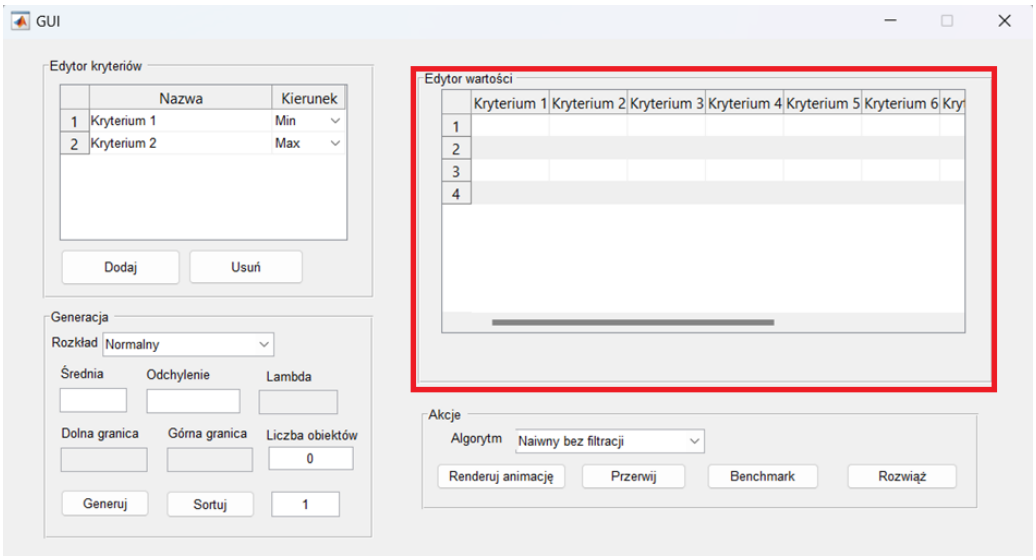
Pierwszą z tych sekcji jest **Edytor kryteriów**, który umożliwia definiowanie kryteriów analizy, a użytkownik może nie tylko określić nazwę każdego kryterium, ale również zdecydować,

czy jego kierunek będzie polegał na minimalizacji czy maksymalizacji wartości, co jest niezwykle istotne, gdyż różne kryteria mogą mieć różny wpływ na ostateczny wynik analizy, a ich odpowiednie zdefiniowanie pozwala na dokładniejsze i bardziej trafne wyznaczenie punktów niezdominowanych. Sekcja ta pozwala również na dynamiczne modyfikowanie listy kryteriów dzięki przyciskom umożliwiającym dodawanie i usuwanie wpisów, co zapewnia dużą elastyczność i umożliwia dostosowanie analizy do zmieniających się wymagań użytkownika lub specyficznych warunków analizowanego problemu.



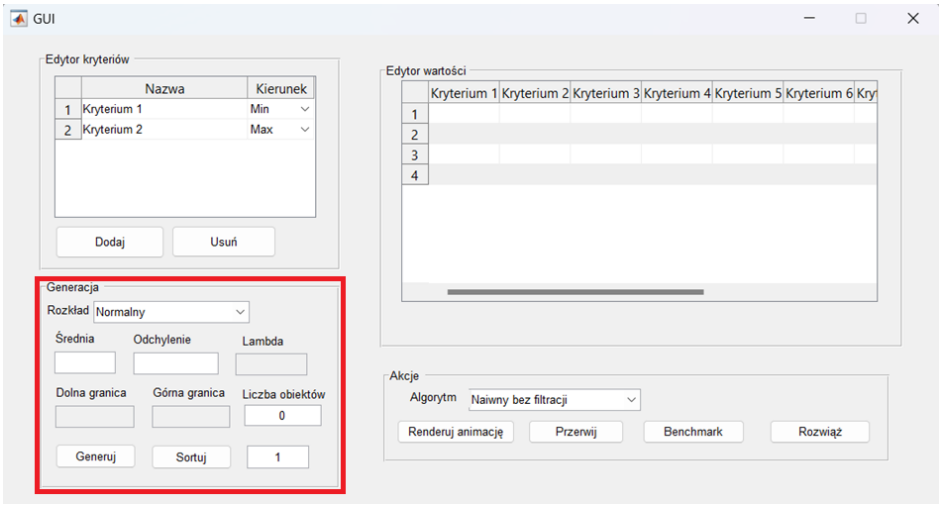
Rys.2 Sekcja Kryteria

Kolejną sekcją jest **Edytor wartości**, w którym użytkownik wprowadza dane do analizy, a tabela zawiera wiersze reprezentujące poszczególne punkty oraz kolumny odpowiadające wartościom kryteriów, co pozwala na precyzyjne odwzorowanie struktury danych, które będą analizowane, oraz zapewnia możliwość przechowywania danych w przejrzystej formie. Dzięki temu użytkownik może ręcznie wprowadzać wartości, co jest przydatne w przypadku, gdy dane nie są dostępne w sposób automatyczny lub wymagają specyficznych modyfikacji, ale może także korzystać z danych generowanych w innej części programu, co zapewnia elastyczność i pozwala na łatwiejsze przygotowanie danych wejściowych do analizy, co z kolei przekłada się na dokładność uzyskanych wyników.



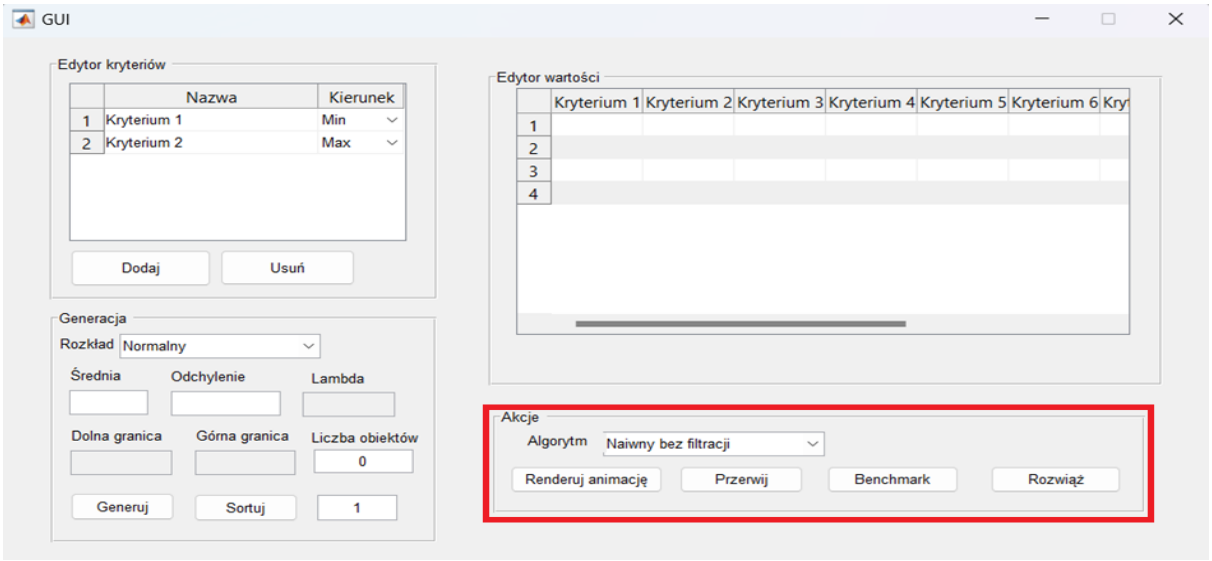
Rys.3 Sekcja Wartości

Sekcja Generacja oferuje narzędzia do automatycznego tworzenia danych wejściowych, co jest szczególnie istotne w sytuacjach, gdy konieczne jest przetestowanie algorytmów na dużych zbiorach danych lub gdy brak jest danych rzeczywistych. Użytkownik ma możliwość wyboru typu rozkładu, takiego jak rozkład normalny, co pozwala na symulację danych o określonych właściwościach statystycznych, a także może określić parametry, takie jak średnia, odchylenie standardowe, zakres wartości czy liczba generowanych punktów, co daje pełną kontrolę nad charakterystyką generowanych danych i pozwala na tworzenie zestawów danych, które odzwierciedlają rzeczywiste warunki analizowanego problemu. Dodatkowo można skorzystać z funkcji sortowania, która porządkuje dane zgodnie z wybranym kryterium, co może być przydatne podczas przygotowywania danych do analizy, gdyż uporządkowane dane mogą być łatwiejsze do interpretacji, a także mogą przyspieszyć działanie algorytmów, które korzystają z określonej kolejności danych.



Rys.4 Sekcja Generacja

Ostatnia sekcja, **sekcja Akcje**, pozwala na wybór i uruchamianie algorytmów analitycznych, co stanowi kluczowy element całego procesu analizy, ponieważ bezpośrednio przekłada się na możliwość wykorzystania zaimplementowanych algorytmów do rozwiązania problemów decyzyjnych, z jakimi mierzy się użytkownik. Użytkownik może wybrać jedną z dostępnych metod analizy, takich jak algorytm „Naiwny bez filtracji", który, choć prosty, może stanowić istotny punkt odniesienia w porównaniach wyników z bardziej zaawansowanymi metodami, a także uruchomić wizualizację procesu analitycznego, który jest przedstawiany w formie animacji, co umożliwia lepsze zrozumienie, w jaki sposób dane są przetwarzane oraz jak poszczególne algorytmy prowadzą do wyznaczenia punktów niezdominowanych, a wizualizacja ta pozwala także na bieżące śledzenie działania programu, co jest niezwykle przydatne w przypadku bardziej skomplikowanych analiz, ponieważ daje możliwość wychwycenia ewentualnych błędów lub niespójności. Sekcja ta umożliwia również zatrzymanie działania programu w dowolnym momencie, co jest szczególnie ważne, gdy użytkownik zauważy, że analiza zmierza w niepożądanym kierunku lub gdy konieczna jest korekta danych wejściowych, a także oferuje funkcję uruchamiania testów porównawczych (benchmarków), które pozwalają na ocenę efektywności różnych algorytmów, co z kolei jest przydatne przy podejmowaniu decyzji o wyborze najlepszego z dostępnych podejść analitycznych. Dodatkowo sekcja Akcje umożliwia rozwiązanie zadania poprzez wyznaczenie punktów niezdominowanych, co jest kluczowym celem całego projektu, a dzięki intuicyjnie zaprojektowanym przyciskom oraz przejrzystemu układowi funkcji sekcja ta zapewnia pełną kontrolę nad procesem analizy, co jest nieocenione dla użytkownika, który pragnie mieć pewność, że każda czynność podejmowana w ramach analizy jest przemyślana i dobrze uzasadniona, a także że może on dowolnie modyfikować i kontrolować przebieg procesu, co znacząco wpływa na jakość uzyskanych wyników i pozwala na elastyczne reagowanie na zmieniające się potrzeby analityczne.



Rys.5 Sekcja Akcje

Opis Algorytmów:

W ramach projektu uwzględniono implementację trzech algorytmów, spośród których pierwszy był algorytmem naiwnym bez filtracji, mającym na celu zaprezentowanie prostej, lecz nieoptymalnej metody określania punktów niezdominowanych, co miało służyć jako punkt wyjścia do porównań i wyznaczania bardziej efektywnych metod, drugi algorytm był także naiwnym, lecz zastosowano w nim dodatkową filtrację, która miała za zadanie zredukować liczbę analizowanych punktów w celu optymalizacji procesu wyszukiwania, ponieważ zmniejszenie liczby punktów wejściowych przekłada się bezpośrednio na skrócenie czasu obliczeń, a trzeci algorytm, będący przygotowaniem danych dla algorytmu Luccio-Prepata, miał na celu zademonstrowanie bardziej zaawansowanego podejścia do problemu wielokryterialnego, które obejmowało przekształcanie oraz wstępne przetwarzanie danych w sposób umożliwiający ich efektywne przetworzenie przez algorytm docelowy, co pozwala na lepsze wykorzystanie zaawansowanych technik optymalizacyjnych i zwiększa prawdopodobieństwo uzyskania precyzyjnych oraz wiarygodnych wyników, które mogą stanowić solidną podstawę do dalszych analiz decyzyjnych.

Algorytm Naiwny Bez Filtracji, jak sama nazwa wskazuje, to podstawowe podejście do wyznaczania zbioru punktów niedominowanych, w którym każdy punkt w wielowymiarowym zbiorze danych jest porównywany z każdym innym punktem w celu sprawdzenia, czy jest zdominowany (tzn. czy istnieje inny punkt, który jest lepszy w co najmniej jednym wymiarze i nie gorszy w pozostałych). W przypadku stwierdzenia dominacji punkt zostaje wyeliminowany z dalszej analizy, a te, które nie są zdominowane przez żadne inne, trafiają do zbioru Pareto. Główna zaleta tego podejścia to prostota – algorytm jest łatwy do zaimplementowania i działa poprawnie bez potrzeby stosowania dodatkowych mechanizmów. Jednak jego największą wadą jest niska wydajność, wynikająca z potrzeby przeprowadzenia porównań w liczbie rzędu O(n^2) dla zbioru liczącego n punktów. Algorytm ten, ze względu na swoją prostotę, ma stosunkowo niską efektywność, ponieważ wymaga przeprowadzenia dużej liczby porównań, co prowadzi do znacznego wzrostu złożoności obliczeniowej w przypadku dużych zbiorów danych, a każde kolejne dodanie nowego punktu do analizy powoduje zwiększenie liczby porównań, które muszą zostać wykonane. Niemniej jednak, algorytm ten pełni istotną funkcję edukacyjną, umożliwiając użytkownikom zrozumienie, na czym polega pojęcie dominacji w analizie wielokryterialnej, ponieważ poprzez ręczne śledzenie działania algorytmu można łatwo dostrzec, w jaki sposób dokonywane są porównania między punktami oraz jakie kryteria decydują o tym, że jeden punkt jest zdominowany przez inny. Warto zaznaczyć, że choć algorytm Naiwny Bez Filtracji nie jest zalecany do stosowania w przypadku dużych zbiorów danych, ze względu na swoją

niską efektywność, to jednak jest cennym narzędziem w kontekście edukacyjnym oraz jako punkt odniesienia, który pozwala na ocenę korzyści wynikających z zastosowania bardziej zaawansowanych metod filtracji i optymalizacji.

```
function [P, comparisons, coordComparisons] = naive_no_filter(X)
    % Algorytm bez filtracji zgodny z pseudokodem
    % Input: X - macierz punktów
    % Output: P - macierz punktów niezdominowanych
    %         comparisons - liczba porównań punktów
    %         coordComparisons - liczba porównań współrzędnych

    n = size(X, 1); % Liczba punktów w zbiorze X
    P = []; % Inicjalizacja zbioru punktów niezdominowanych
    comparisons = 0; % Liczba porównań punktów
    coordComparisons = 0; % Liczba porównań współrzędnych

    for i = 1:n
        Y = X(i, :); % Rozważany punkt
        fl = 0; % Flaga określająca, czy punkt Y został zastąpiony

        for j = i+1:n
            comparisons = comparisons + 1; % Liczymy porównania punktów
            coordComparisons = coordComparisons + 2 * size(X, 2); % Porównania współrzędnych

            % Sprawdzamy dominację
            if all(Y <= X(j, :)) && any(Y < X(j, :)) % Y ≤ X(j)
                % Punkt X(j) jest zdominowany, usuwamy go
                X(j, :) = inf; % Usunięcie X(j) przez oznaczenie jako punkt "nieskończony"
            elseif all(X(j, :) <= Y) && any(X(j, :) < Y) % X(j) ≤ Y
                % Punkt Y jest zdominowany, zastępujemy go przez X(j)
                Y = X(j, :);
                X(j, :) = inf; % Usunięcie X(j)
                fl = 1;
            end
        end

        % Dodanie punktu Y do listy niezdominowanych
        P = [P; Y];

        % Usunięcie Y z listy X, jeśli fl=0 (czyli Y = X(i))
        if fl == 0
            X(i, :) = inf;
        end
    end
end
```

Rys.6 Algorytm Naiwny Bez Filtracji

Algorytm Naiwny z Filtracją stanowi rozwinięcie koncepcji algorytmu naiwnego, wprowadzając dodatkową filtrację mającą na celu ograniczenie liczby punktów, które muszą zostać porównane w procesie wyznaczania punktów niezdominowanych, co z kolei prowadzi do zwiększenia efektywności obliczeniowej i redukcji złożoności czasowej całego procesu analizy. W odróżnieniu od algorytmu bez filtracji, wersja z filtracją korzysta z wstępnego przetwarzania danych, które polega na eliminacji punktów, które na podstawie prostych kryteriów można uznać za zdominowane bez konieczności ich porównywania z każdym innym punktem w zbiorze, co znacząco redukuje liczbę koniecznych operacji obliczeniowych. Tego rodzaju filtracja może obejmować różne techniki, takie jak porządkowanie punktów według wartości jednego z kryteriów, co pozwala na szybkie wyeliminowanie tych punktów, które są oczywiście gorsze pod względem wszystkich

kryteriów, bez potrzeby bardziej szczegółowej analizy. Dzięki zastosowaniu filtracji, algorytm ten charakteryzuje się lepszą skalowalnością niż algorytm naiwny bez filtracji, co sprawia, że jest bardziej odpowiedni do analizy większych zbiorów danych, gdzie liczba porównań może stać się czynnikiem ograniczającym. Warto jednak podkreślić, że mimo wprowadzenia filtracji, algorytm nadal nie jest optymalnym rozwiązaniem w kontekście najbardziej zaawansowanych metod analizy wielokryterialnej, gdyż jego złożoność wciąż rośnie wykładniczo w miarę zwiększania liczby punktów, ale stanowi istotny krok naprzód w porównaniu z podejściem całkowicie naiwnym i pozwala na lepsze zrozumienie korzyści płynących z zastosowania wstępnej selekcji danych.

```
function [P, comparisons, coordComparisons] = naive_with_filter(X)
    % Algorytm z filtracją punktów zdominowanych zgodny z pseudokodem
    % Input: X - macierz punktów
    % Output: P - macierz punktów niezdominowanych
    %         comparisons - liczba porównań punktów
    %         coordComparisons - liczba porównań współrzędnych

    P = []; % Inicjalizacja zbioru punktów niezdominowanych
    comparisons = 0; % Liczba porównań punktów
    coordComparisons = 0; % Liczba porównań współrzędnych

    while size(X, 1) > 1 % Dopóki w X jest więcej niż jeden punkt
        Y = X(1, :); % Wybierz pierwszy punkt
        X(1, :) = []; % Usuń go z listy X

        i = 1;
        while i <= size(X, 1)
            comparisons = comparisons + 1; % Liczba porównań punktów
            coordComparisons = coordComparisons + 2 * size(X, 2); % Porównania współrzędnych

            % Porównaj punkt Y z X(i,:)
            if all(Y <= X(i, :)) && any(Y < X(i, :)) % Y ≤ X(i)
                X(i, :) = []; % Usuń punkt X(i), bo jest zdominowany przez Y
            elseif all(X(i, :) <= Y) && any(X(i, :) < Y) % X(i) ≤ Y
                Y = X(i, :); % Zastąp Y punktem X(i)
                X(i, :) = []; % Usuń punkt X(i)
            else
                i = i + 1; % Przejdź do następnego punktu
            end
        end

        % Dodaj Y do zbioru niezdominowanego
        P = [P; Y];

        % Usuń z X wszystkie punkty zdominowane przez Y
        dominatedIndices = all(bsxfun(@le, Y, X), 2) & any(bsxfun(@lt, Y, X), 2);
        X(dominatedIndices, :) = [];
    end

    % Jeśli w X pozostał tylko jeden punkt, dodaj go do P
    if size(X, 1) == 1
        P = [P; X];
    end
end
```

Rys.7 Algorytm Naiwny z filtracją

Jest on wariantem prostego algorytmu stosowanego do znajdowania punktów niedominowanych (zbioru Pareto) w wielokryterialnym zbiorze danych, który wykorzystuje

mechanizm filtracji, eliminujący zdominowane punkty na wczesnym etapie działania. Dzięki temu redukuje liczbę porównań, co przyspiesza obliczenia w porównaniu do klasycznego naiwnego podejścia. Algorytm działa w kilku krokach: najpierw identyfikuje i usuwa punkty zdominowane (np. punkt A, który jest gorszy od punktu B we wszystkich wymiarach, jest eliminowany), a następnie analizuje pozostałe punkty w celu utworzenia końcowego zbioru. Jego główną zaletą jest skrócenie czasu obliczeń, szczególnie w przypadku zbiorów zawierających wiele punktów zdominowanych, a także prostota implementacji, która sprawia, że jest łatwy do zrozumienia i wdrożenia. Jednak algorytm ma również swoje wady – jego efektywność znacząco zależy od struktury danych, a w sytuacji, gdy zbiór danych zawiera niewiele zdominowanych punktów, zysk wynikający z filtracji może być minimalny lub wręcz negatywny, biorąc pod uwagę dodatkowy koszt jej przeprowadzania.

Algorytm Luccio-Prepata to zaawansowane podejście do problemu wyznaczania punktów niezdominowanych w przestrzeni wielokryterialnej, które różni się od poprzednich algorytmów zarówno pod względem złożoności, jak i efektywności obliczeniowej, a jego głównym celem jest maksymalne zredukowanie liczby koniecznych porównań poprzez odpowiednie przetworzenie i organizację danych wejściowych. Algorytm ten opiera się na założeniu, że odpowiednie przygotowanie danych przed właściwą analizą pozwala na znaczące przyspieszenie procesu wyznaczania punktów niezdominowanych, a w praktyce oznacza to, że dane są najpierw wstępnie przekształcane w taki sposób, aby możliwe było ich uporządkowanie według określonych kryteriów, co pozwala na szybkie eliminowanie punktów zdominowanych bez konieczności porównywania każdego punktu z każdym innym. Algorytm Luccio-Prepata wykorzystuje zaawansowane struktury danych oraz techniki sortowania, które umożliwiają efektywne przetwarzanie dużych zbiorów danych, a jego złożoność obliczeniowa jest znacząco niższa niż w przypadku algorytmów naiwnych, co sprawia, że jest on odpowiedni do zastosowania w analizie bardzo dużych zbiorów danych, gdzie inne podejścia okazują się niewystarczające. Ponadto, algorytm ten został zaprojektowany w taki sposób, aby możliwe było równoległe przetwarzanie danych, co dodatkowo zwiększa jego efektywność, zwłaszcza w przypadku wykorzystania nowoczesnych systemów komputerowych wyposażonych w wiele jednostek obliczeniowych. W kontekście projektu, algorytm Luccio-Prepata pełnił rolę najbardziej zaawansowanego narzędzia analitycznego, które nie tylko pozwalało na wyznaczenie punktów niezdominowanych w sposób bardziej efektywny niż algorytmy naiwne, ale także stanowiło demonstrację korzyści wynikających z zastosowania zaawansowanych metod przetwarzania danych i optymalizacji, które są kluczowe w kontekście analizy wielokryterialnej w rzeczywistych, złożonych problemach decyzyjnych.

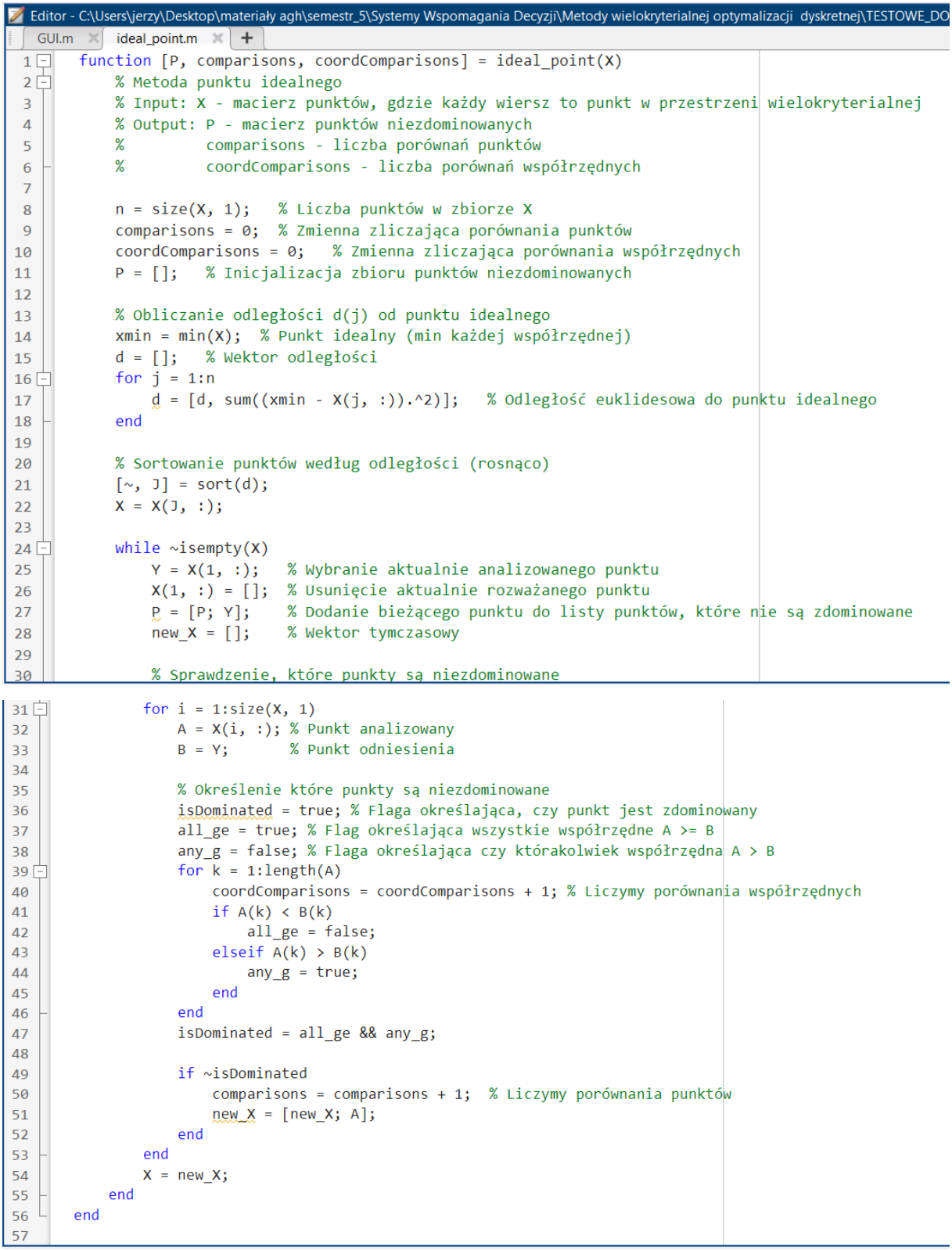
```
% Algorytm Luccio Prepata w pierwszym kroku dokonuje sortowania po
% pierwszym kryterium. Funkcja sprawdza czy punkty sa posortowane, jesli
% nie dokonuje sortowania
function [X countN countP] = prepareDataForLuccio(input, dir)
    countN = 0;
    countP = 0;
    needSort = 0;
    X = input;
    for i=1:size(input,2)-1
        if X(1,i)*dir(1) < X(1,i+1)*dir(1)
            needSort = 1;
            break;
        end
        countN = countN +1;
    end

    if(needSort==1)
        [X countN_ countP_] = sortInput(input, dir(1));
        countN = countN + countN_;
        countP = countP + countP_;
    end
end

function [X countN countP] = sortInput(input, dir)
    X = input;
    countN = 0;
    countP = 0;

    pointsCount =  size(input,2);
    pivot_ind = uint8(floor(pointsCount/2));
    Left = [];
    Right = [];
    j = 1;
    k = 1;
                if(pointsCount<2)
                   X = input;
                 else
                   pivot = input(:,pivot_ind);
                   countP = countP + 1;
                   for i=1:pointsCount
                     if(i~=pivot_ind)
                       if(input(1,i) * dir > pivot(1)*dir)
                         Left(:,j) = input(:,i);
                         j = j+1;
                       else
                         Right(:,k) = input(:,i);
                         k = k+1;
                       end
                       countN = countN +1;
                     end
                   end

                   [Left cl cpl] = sortInput(Left, dir);
                   [Right cr cpr] = sortInput(Right, dir);
                   countN = countN + cl + cr;
                   countP = countP + cpl + cpr;
                   X = [Left pivot Right];
                end
            end
```

Powyższy kod znajdował się w pierwszej części pliku i służył sortowaniu danych, poniżej kod przedstawiający właściwy algorytm.

```
function F = KungLuccioPreparata( X, dir )

    %[data count] = KLP(X,dir);
    a = size(X);
    [input counter countP] = prepareDataForLuccio(X, dir);
    assignin('base', 'toPlot', [1 1 1 1]);
    f = figure('Position',[400 150 800 450]);
    set(f, 'Resize', 'off');
    cur_title = ['Algorytm Kung-Luccio-Preparata, liczba obiektów= ', num2str(size(X,2)) ];

    pause on
    dominated = [];
    nonDominated = [];
    compared = [];
    % rysuj poczatkowe punkty
    plotInputs(f,X,compared,nonDominated,dominated, cur_title,0);

    plotStaticText(f);
    F(1) = getframe(f); % pierwsza klatka animacji

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [column,row] = size(X);
    if( column == 2)
        [nodominated, F1, dominated,nonDominated,compared, counter ] = MinimalSet2D(input, row, dir,F,dominated, nonDominated, compared, counter, f);
    else
        [nodominated, F1, dominated,nonDominated,compared, counter ]= MinimalSet(input, 1, row, dir,F,dominated, nonDominated, compared, counter, f, X);
    end

    % rysowanie ostatniej klatki z ostatecznym rozwiazaniem
    F = F1;
    plotInputs(f,X,[],nodominated,dominated, cur_title,1);
    F1 = getframe(f);
    F = [F F1];
end

            plotInputs(f,X,compared,nonDominated,dominated, cur_title,0);
            F = [F getframe(f)];
        else
            dominated = [dominated X(:,i)];
            plotInputs(f,X,compared,nonDominated,dominated, cur_title,0);
            F = [F getframe(f)];
        end
    end

    %data = [data, min];
    F_return = F;
    dominated_ = dominated;
    nonDominated_ = nonDominated;
    data = nonDominated;
    compared_ = compared;
    counter_ = counter;
end

function [data, F_ret, dominated_,nonDominated_,compared_, counter_, frameCounter_] = UnionM(r,s, dir,F,dominated, nonDominated, compared, counter, f, X)
    [column,row] = size(s);

    %isdominated = true;
    %%%%%%%%%%%%%%
    cur_title = ['Algorytm Kung-Luccio-Preparata, liczba obiektów= ', num2str(size(X,2)) ];
    %%%%%%%%%%%%%%
    for is = 1 : row
       isdominated = false;
       jr = 1;
       [~,rRow] = size(r);
       while(jr <= rRow && isdominated == false)

           % Utworz klatke z zaznaczonymi punktami porownywanymi
            compared(:,1) = s(:,is);
            compared(:,2) = r(:,jr);
            plotInputs(f,X,compared,nonDominated,dominated, cur_title,0);
            F = [F getframe(f)];
            % odczekaj troche, Matlab wykonuje najpierw obliczenia a pozniej akcje graficzne
            % bez pauzy zamrozenie animacji
            pause(0.01);
            % sprawdzamy czy uzytkownik nie przerwal renderowania
            if evalin('base', 'interruptFlag') == 1
                return;
```

```
    end

    counter = counter + 1;
    if true == Dominates(2, r(:,jr),s(:,is), dir)
        isdominated = true;

        dominated = [dominated  r(:,jr)];
         plotInputs(f,X,compared,nonDominated,dominated, cur_title,0);
         F = [F getframe(f)];

    else if true == Dominates(1,s(:,is), r(:,jr), dir)
            dominated = [dominated  s(:,is)];
             plotInputs(f,X,compared,nonDominated,dominated, cur_title,0);
             F = [F getframe(f)];

            %isdominated = true;
            flag = true;
            for tmp = 1:rRow
                 if(r(:,tmp)==s(:,is))
                     r(:,tmp) = [];
                     [~,rRow] = size(r);
                     tmp = tmp - 1;
                     flag = false;
                     %isdominated = false;
                 end
            end
            if flag == true
                jr = jr + 1;
            end

            %tmp = find(ismember(r(:,:),s(:,is)),1,'first');
        else
            jr = jr + 1;
        end
    end
    plotCount(f, counter);
end
if(isdominated == false)
    r = [r, s(:,is)];
    nonDominated = r;%[nonDominated s(:,is)];
    plotInputs(f,X,compared,nonDominated,dominated, cur_title,0);
    F = [F getframe(f)];
                % break;
            else
                dominated = [dominated s(:,is)];
            end
        end
        data = r;
        F_ret = F;
        dominated_ = dominated;
        nonDominated_ = nonDominated;
        compared_ = compared;
        counter_ = counter;
    end

    function isDominates = Dominates(first, x, y, dir)
        len = length(x);
        counter = 1;
        for i=first:len
            if(x(i)*dir(i) < y(i)*dir(i))
                isDominates = false;
                counter = counter + 1;
                return;
            else if(x(i)==y(i))
                    counter = counter + 1;
                end
            end
        end
        if counter == len
            isDominates = false;
            return;
        else
            isDominates = true;
            return;
        end

    end
```

Czwarty algorytm, **Algorytm Punktu Idealnego**, jest kolejnym podejściem do wyznaczania punktów niezdominowanych w analizie wielokryterialnej, które koncentruje się na wykorzystaniu koncepcji punktu referencyjnego, zwanego punktem idealnym. Algorytm ten zakłada, że w przestrzeni wielokryterialnej istnieje punkt, który reprezentuje idealne wartości dla wszystkich kryteriów – jest to punkt, który optymalizuje wszystkie rozważane kryteria jednocześnie, choć w rzeczywistości taki punkt rzadko jest osiągalny. Algorytm Punktu Idealnego działa poprzez mierzenie odległości wszystkich punktów od tego idealnego punktu referencyjnego, co pozwala na ocenę, które z rozwiązań są najbardziej zbliżone do ideału, a które wyraźnie odstają i mogą być uznane za mniej korzystne. Następnie, w iteracyjnej procedurze selekcji, algorytm identyfikuje i eliminuje punkty, które są zdominowane przez inne, pozostawiając jedynie te, które wykazują najlepszy kompromis pomiędzy analizowanymi kryteriami. Procedura iteracyjna polega na powtarzaniu procesu porównywania odległości do punktu idealnego, aż do momentu, gdy żaden punkt nie zostanie uznany za zdominowany, co prowadzi do wyznaczenia zbioru punktów niezdominowanych. Dzięki takiemu podejściu możliwe jest nie tylko wyznaczenie optymalnych rozwiązań, ale także uzyskanie lepszego wglądu w strukturę kompromisów, jakie istnieją pomiędzy poszczególnymi kryteriami. Algorytm ten ma szerokie zastosowanie, zwłaszcza w sytuacjach, gdy istotne jest uwzględnienie odległości do punktu odniesienia i ocena, jak daleko poszczególne rozwiązania są od ideału, co pozwala na bardziej świadome podejmowanie decyzji. Mimo że algorytm Punktu Idealnego może wymagać znacznej ilości obliczeń, zwłaszcza w przypadku dużych zbiorów danych, to jego zaletą jest zdolność do efektywnego identyfikowania punktów, które stanowią najlepsze możliwe kompromisy, co jest szczególnie cenne w kontekście skomplikowanych problemów decyzyjnych, gdzie każda decyzja wiąże się z koniecznością wyboru pomiędzy różnymi, często sprzecznymi kryteriami.



Rys.9 Algorytm Punktu Idealnego

Algorytm punktu idealnego to metoda wyznaczania zbioru punktów niezdominowanych, która wykorzystuje koncepcję idealnego punktu referencyjnego. Idealny punkt jest definiowany jako wektor składający się z minimalnych wartości każdej współrzędnej w wielowymiarowym zbiorze danych. Następnie obliczane są odległości każdego punktu od tego idealnego punktu (najczęściej i w naszej implementacji w sensie euklidesowym), co pozwala na ich posortowanie według tego kryterium. Proces iteracyjnie usuwa punkty

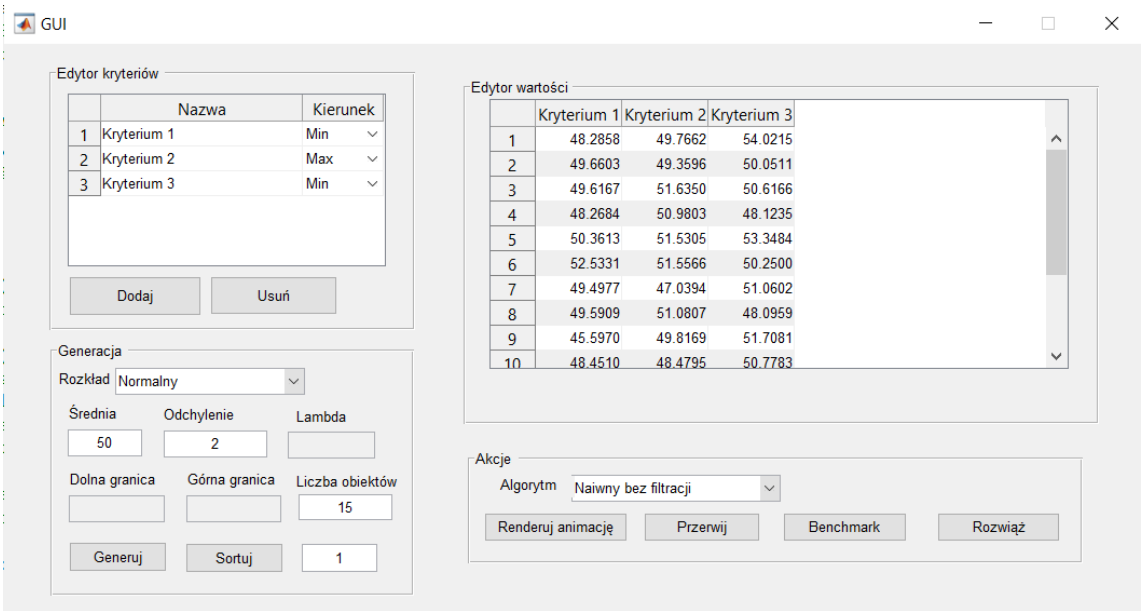
dominowane, zaczynając od najbliższego idealnego punktu, aż do uzyskania zbioru punktów niezdominowanych.

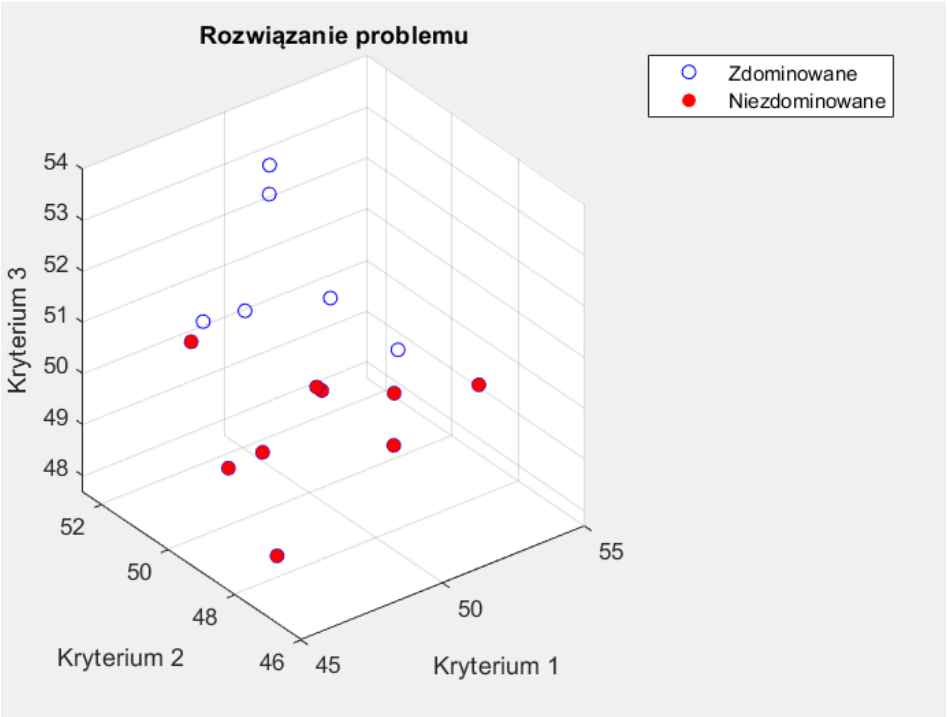
Wyróżnia się on tym, że wprowadza dodatkową strukturę w analizie poprzez wstępne sortowanie punktów, co pozwala na ograniczenie liczby porównań i stopniowe eliminowanie punktów dominowanych. Jego zaletą jest intuicyjność oraz możliwość szybszej identyfikacji kandydatów do zbioru niezdominowanego w porównaniu do algorytmów naiwnych. Główną wadą pozostaje jednak konieczność przeprowadzenia wielokrotnych porównań współrzędnych, co wciąż może być czasochłonne dla dużych zbiorów danych. Algorytm punktu idealnego jest często wykorzystywany w zadaniach o średniej złożoności, oferując lepszą efektywność niż podejścia bez filtracji, ale wymagając mniej zaawansowanej implementacji niż metody oparte na bardziej złożonych strukturach danych.

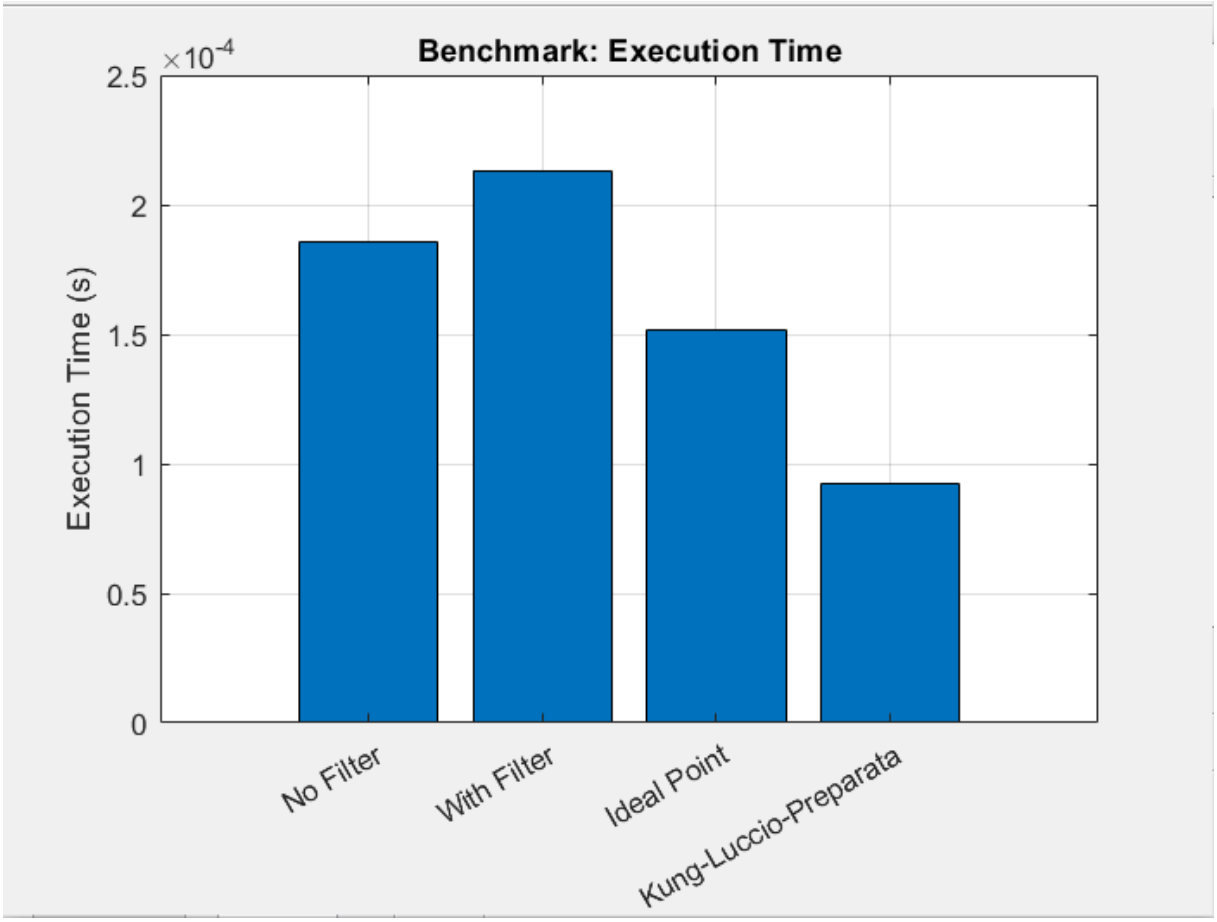
Badanie Metod wielokryterialnej optymalizacji dyskretnej:

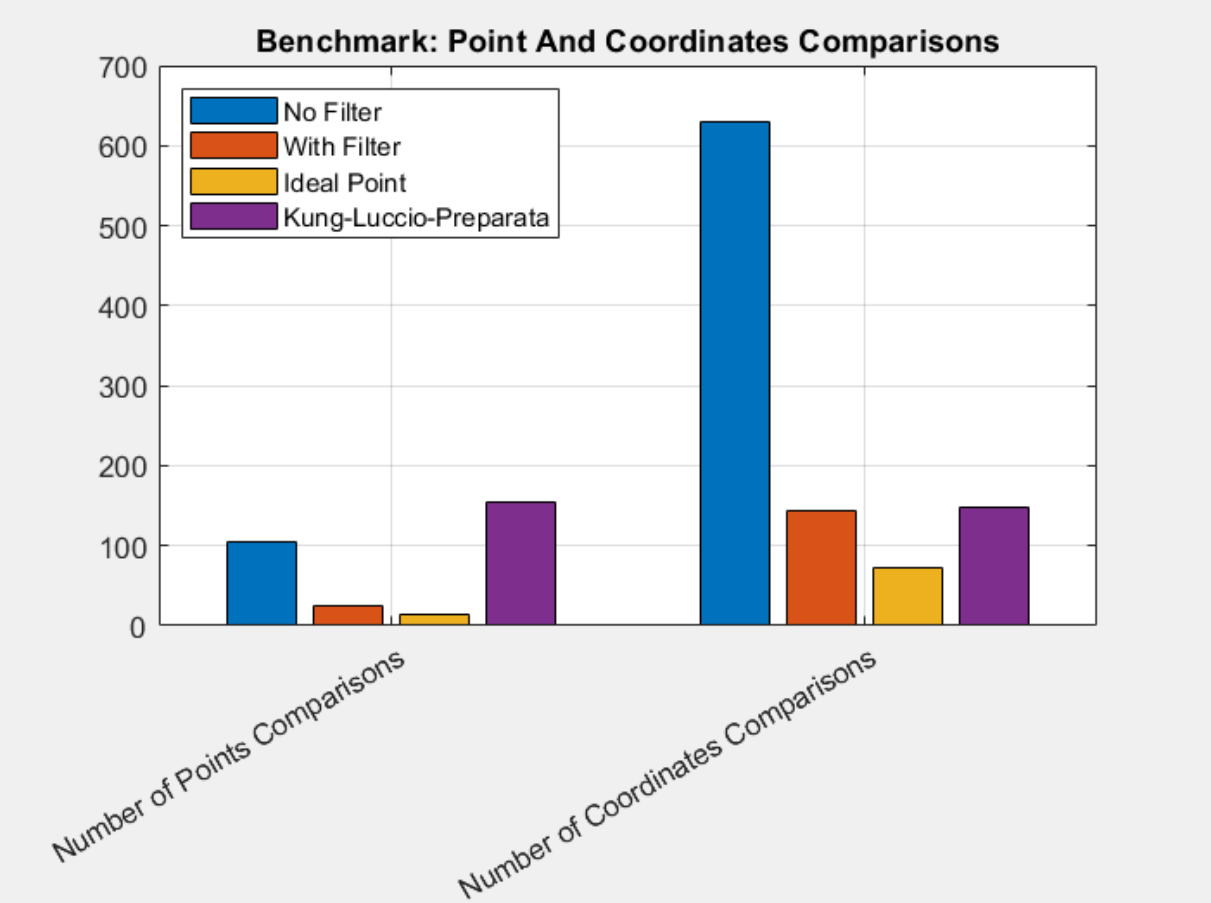
Po zaimplementowaniu algorytmów oraz interfejsu użytkownika przystąpiono do głównej części zadania, którą było badanie metod wielokryterialnej optymalizacji dyskretnej. Przeprowadzone zostały doświadczenia dla różnych parametrów i ustawień układu.

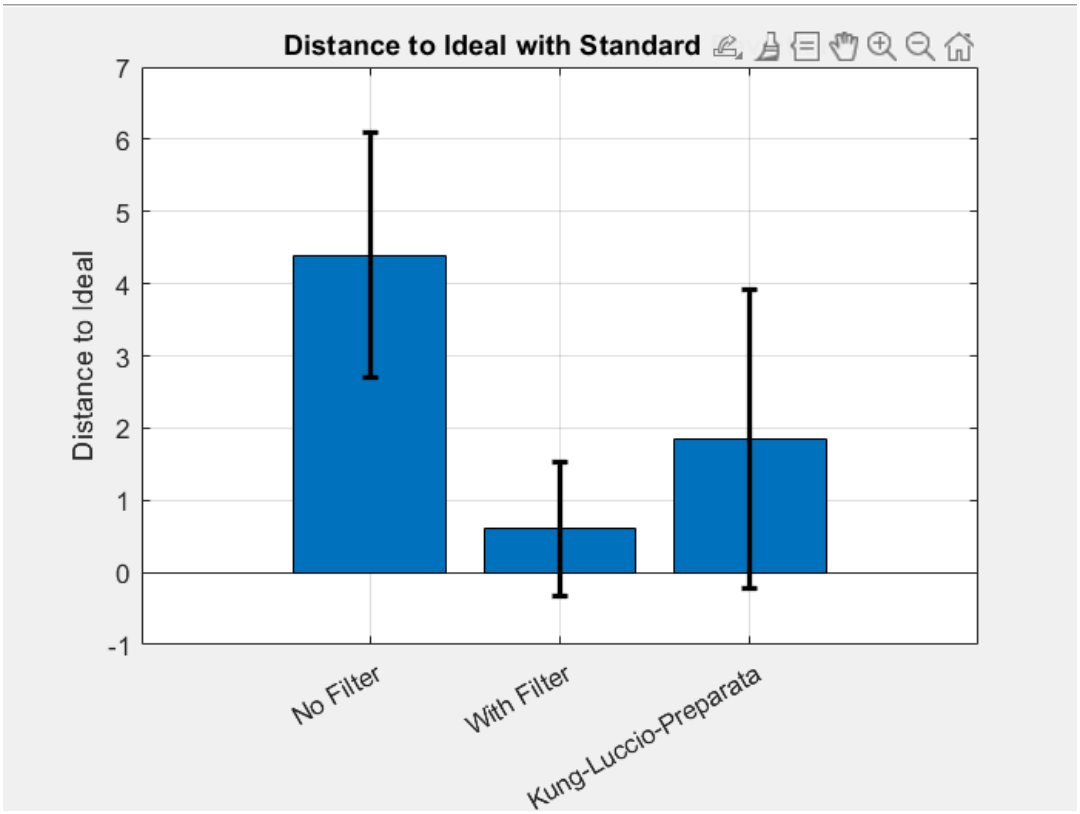
- Rozkład Normalny, Algorytm naiwny bez filtracji



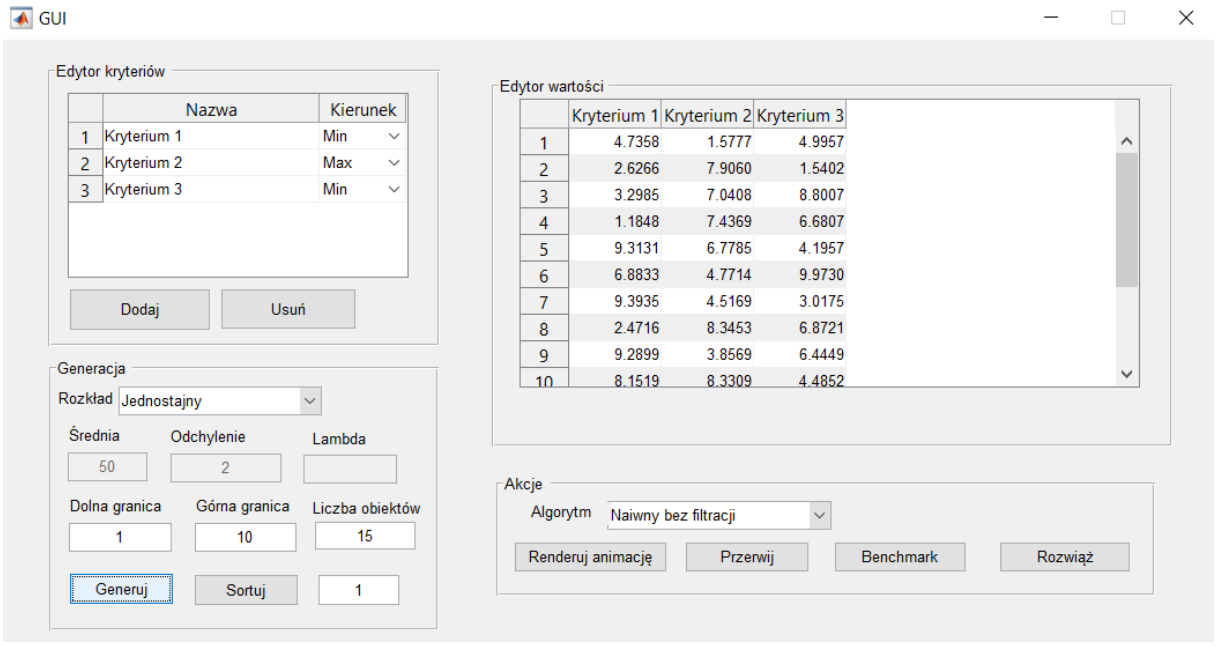

Rozwiązanie problemu
Zdominowane
Niezdominowane
Kryterium 3
54
53
52
51
50
49
48
52
50
48
Kryterium 2
46
45
50
55
Kryterium 1

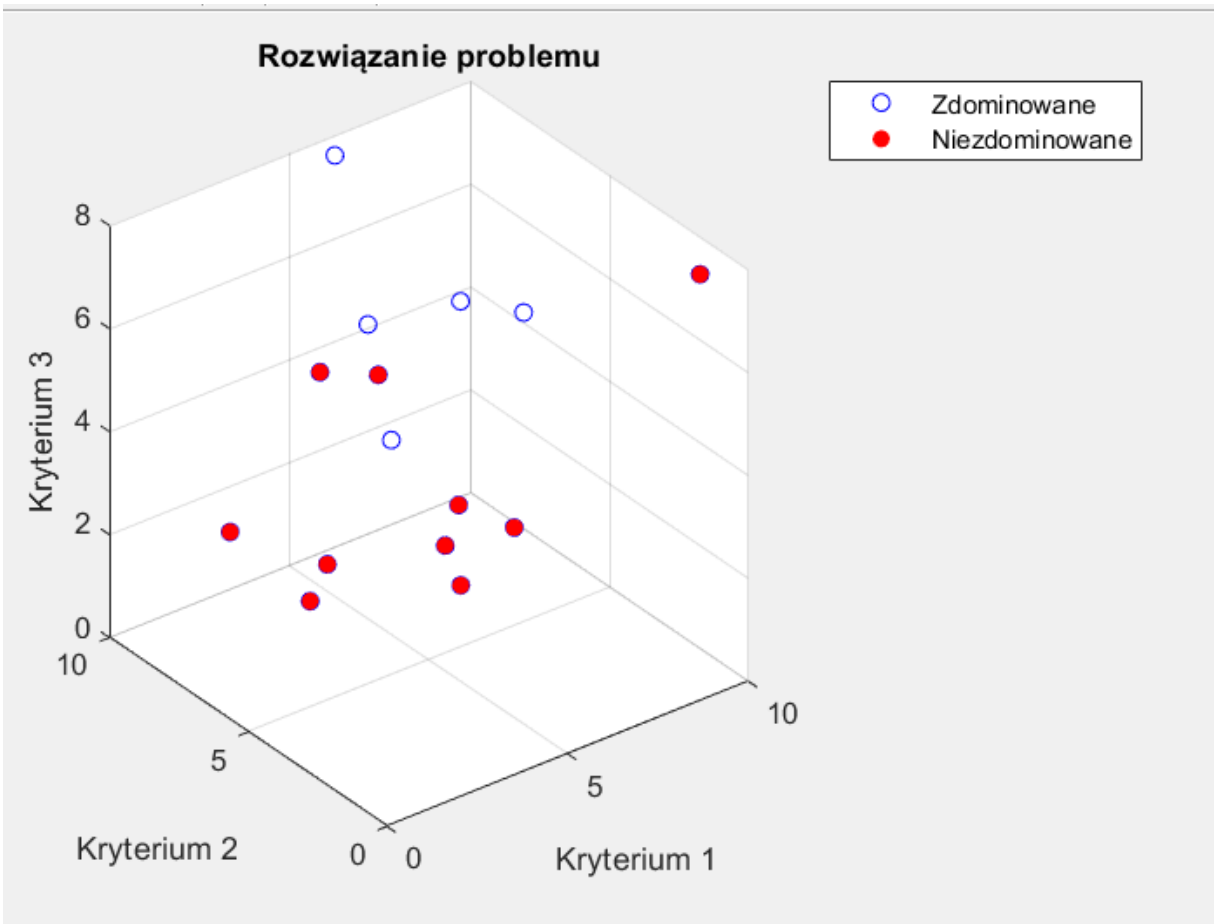

Benchmark: Execution Time
×10^-4
2.5
2
1.5
1
0.5
0
Execution Time (s)
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata

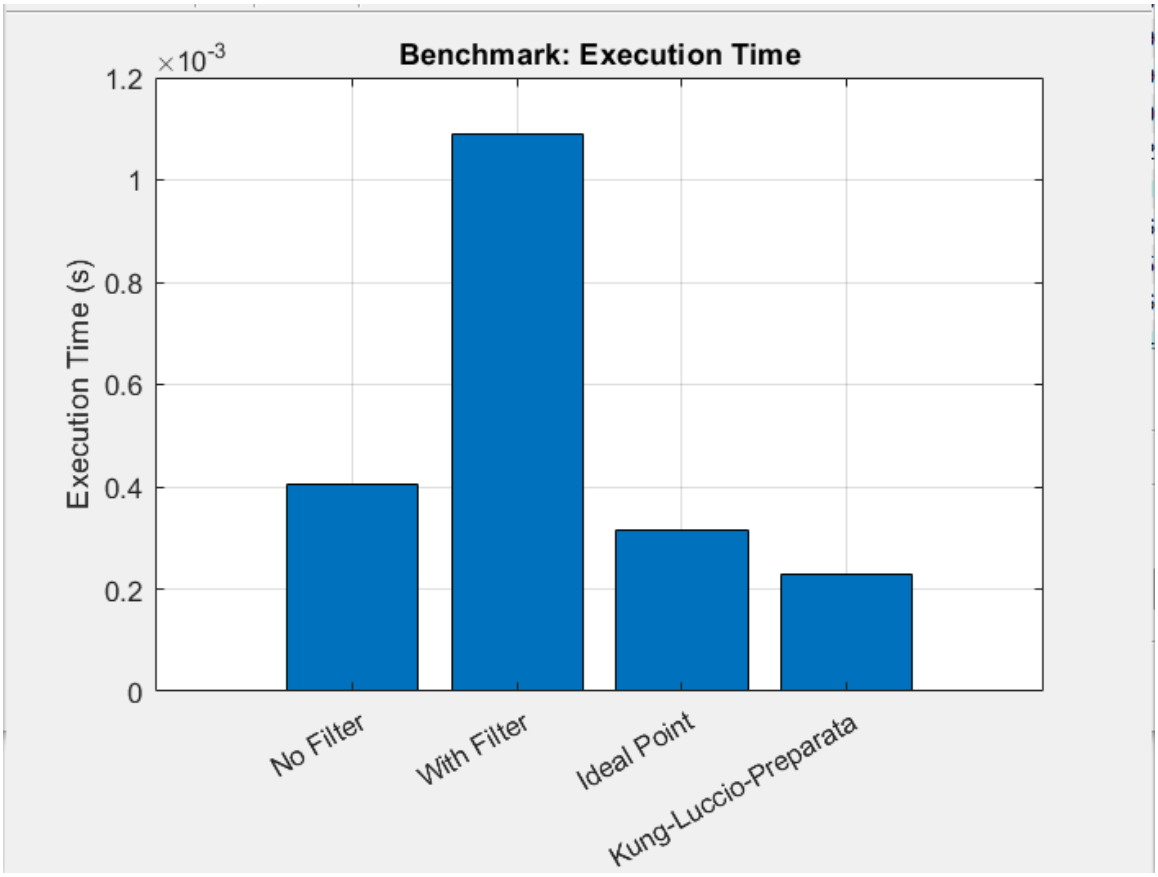

Benchmark: Point And Coordinates Comparisons
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata
700
600
500
400
300
200
100
0
Number of Points Comparisons
Number of Coordinates Comparisons

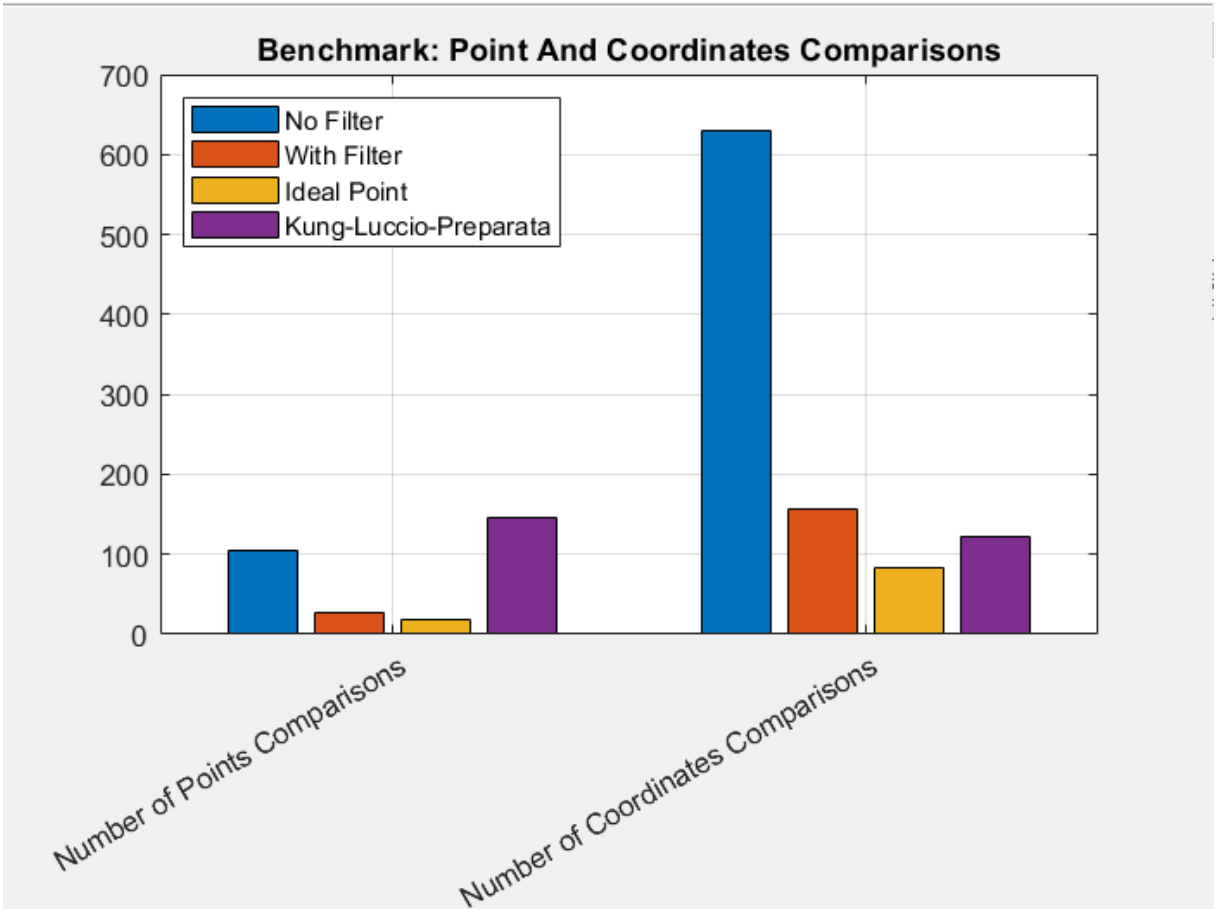

Distance to Ideal with Standard
Distance to Ideal
7
6
5
4
3
2
1
0
-1
No Filter
With Filter
Kung-Luccio-Preparata

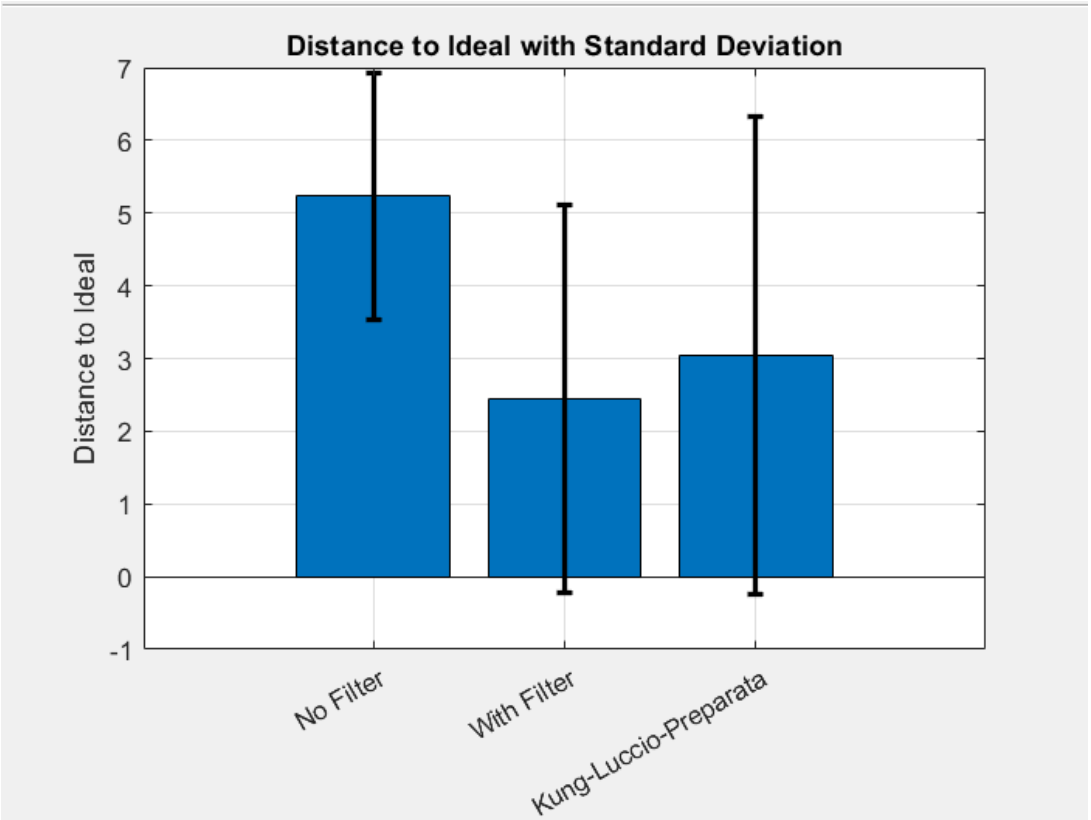
- Rozkład jednostajny, algorytm naiwny bez filtracji



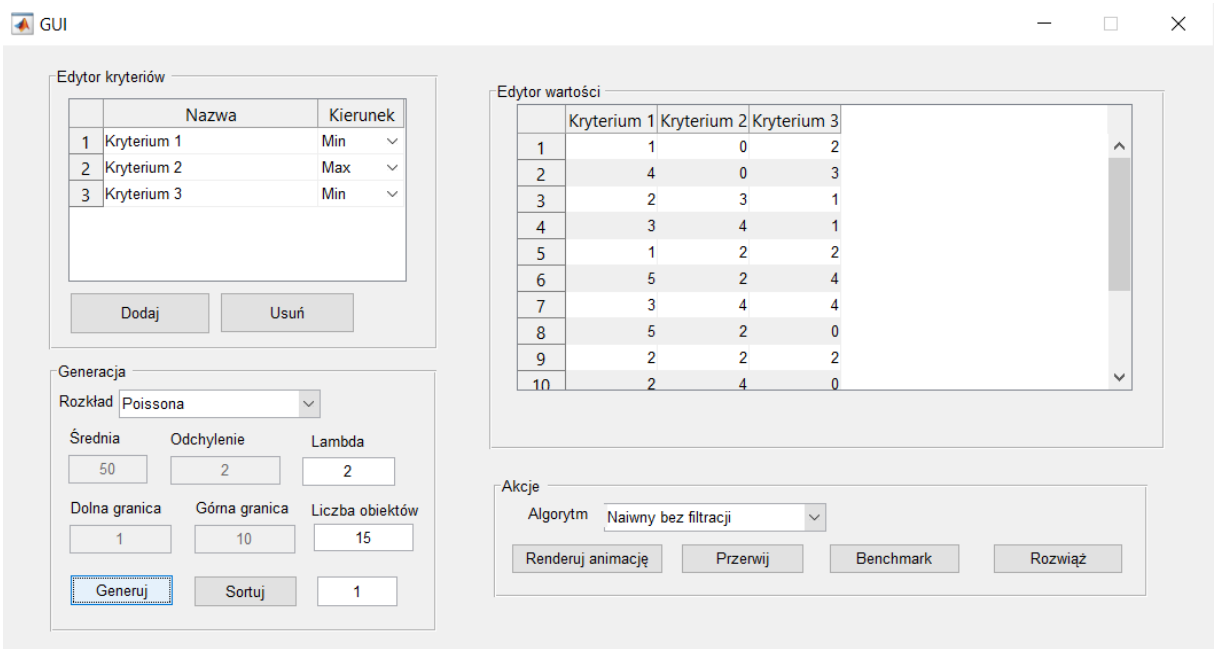


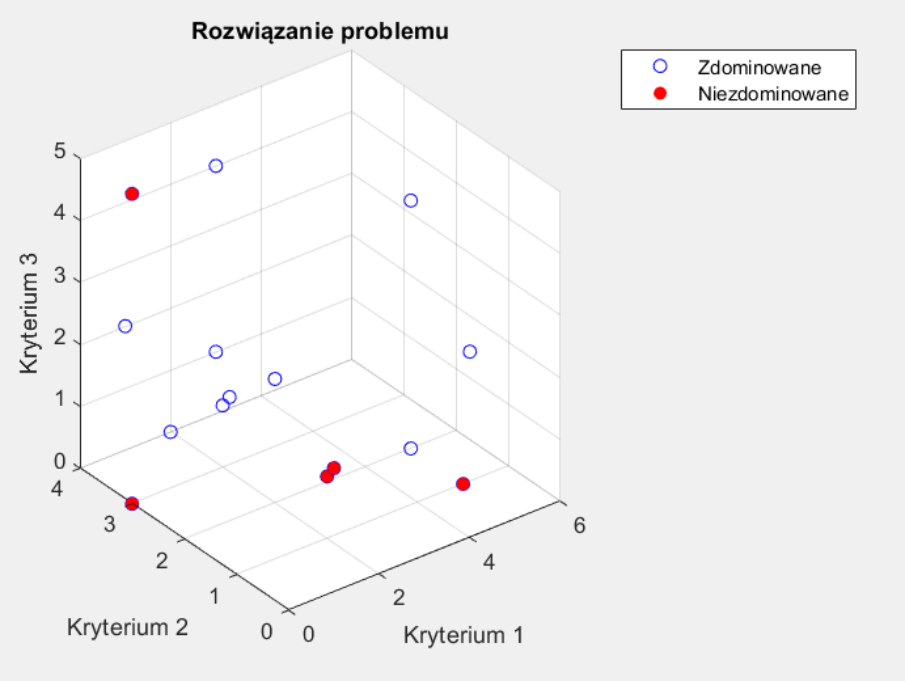

Benchmark: Execution Time
×10-3
1.2
1
0.8
0.6
0.4
0.2
0
Execution Time (s)
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata

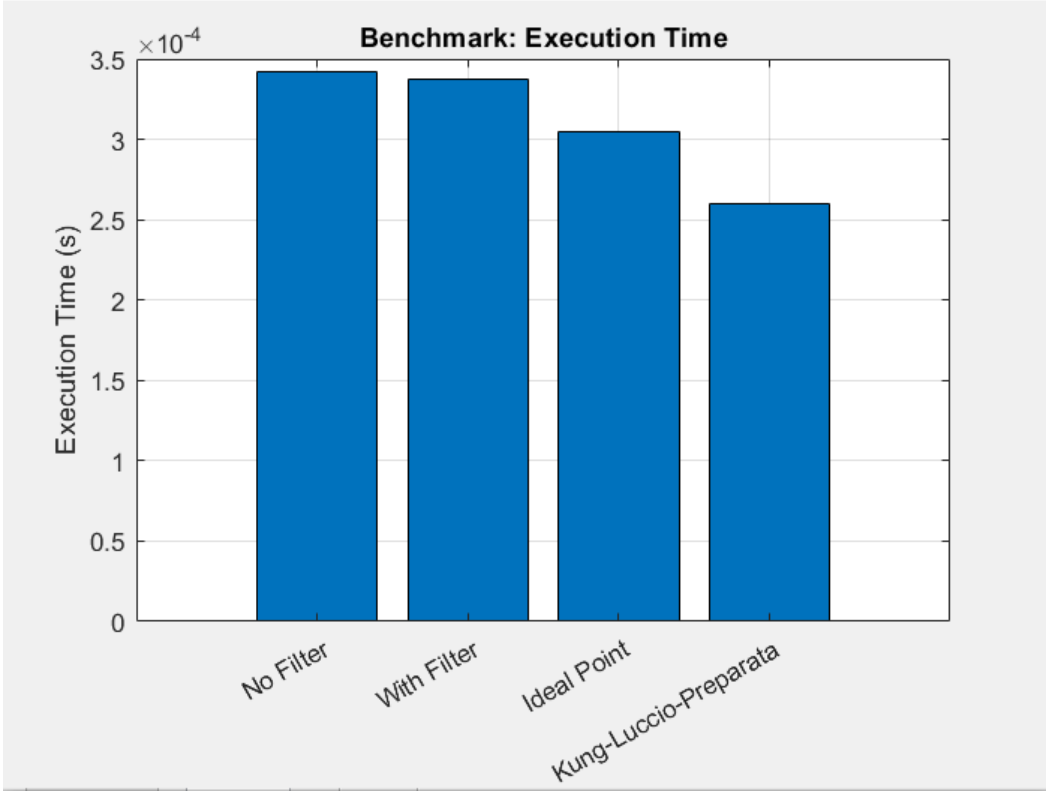

Benchmark: Point And Coordinates Comparisons
700
600
500
400
300
200
100
0
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata
Number of Points Comparisons
Number of Coordinates Comparisons

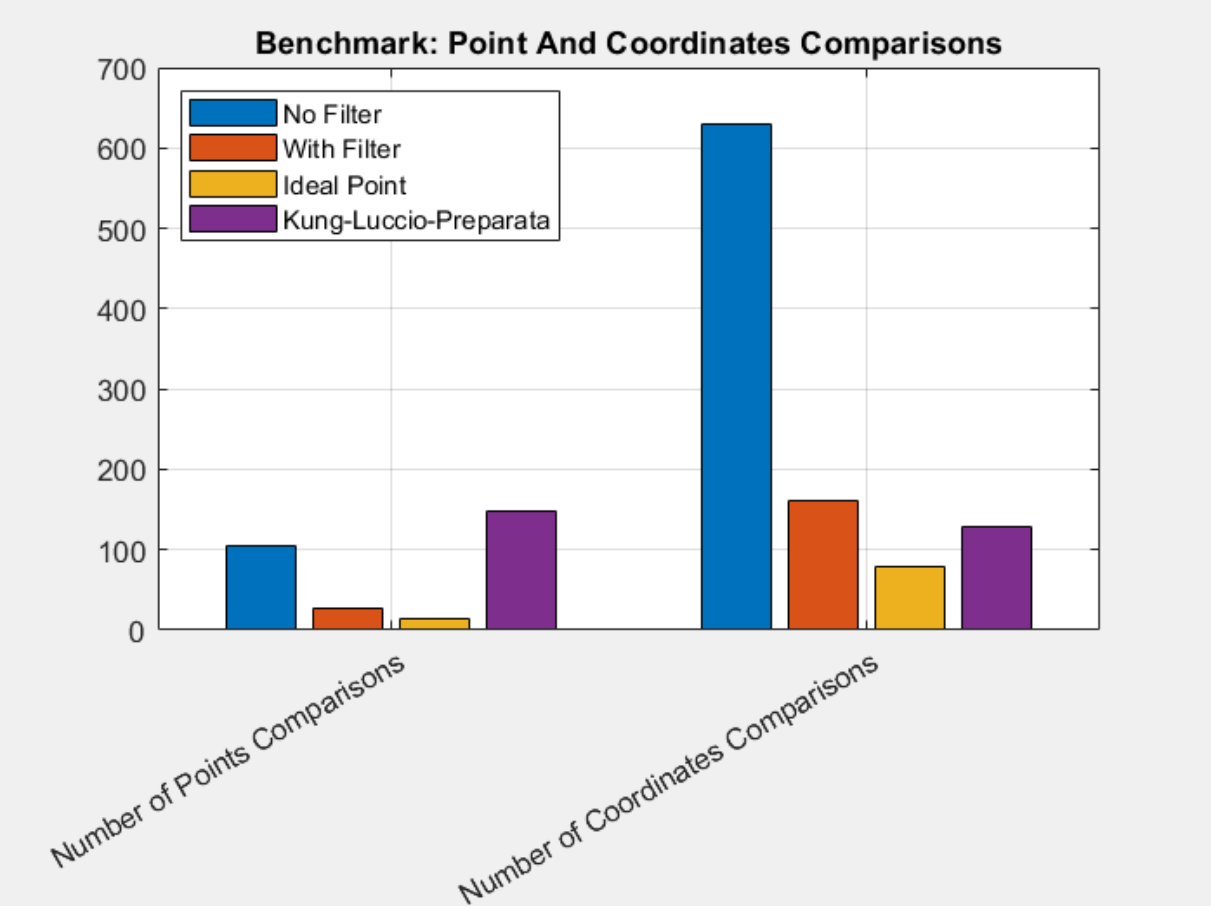


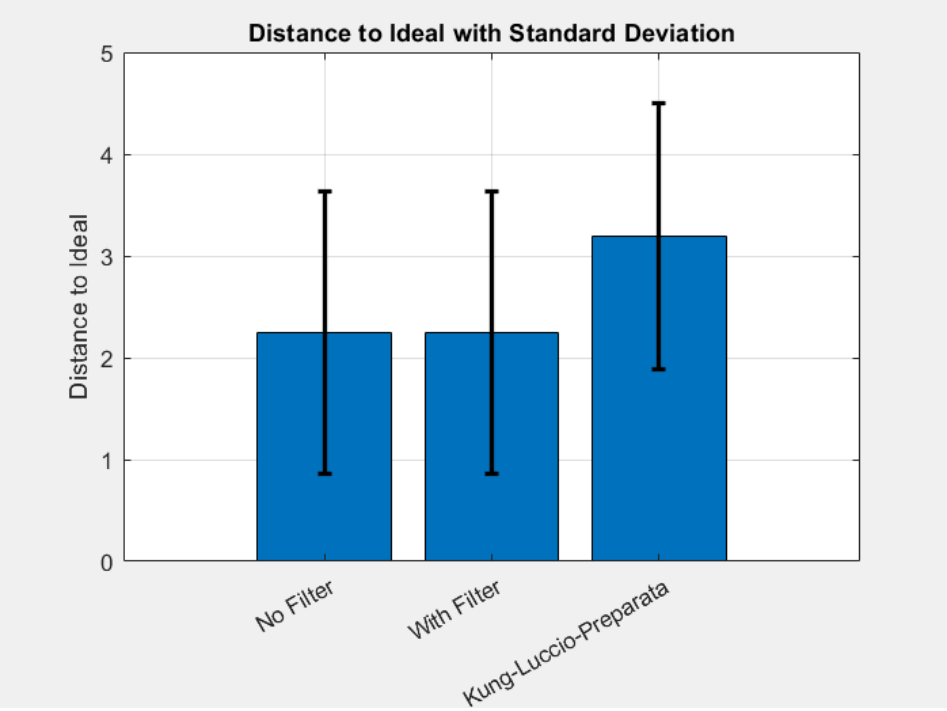
- Rozkład Poissona, algorytm naiwny bez filtracji



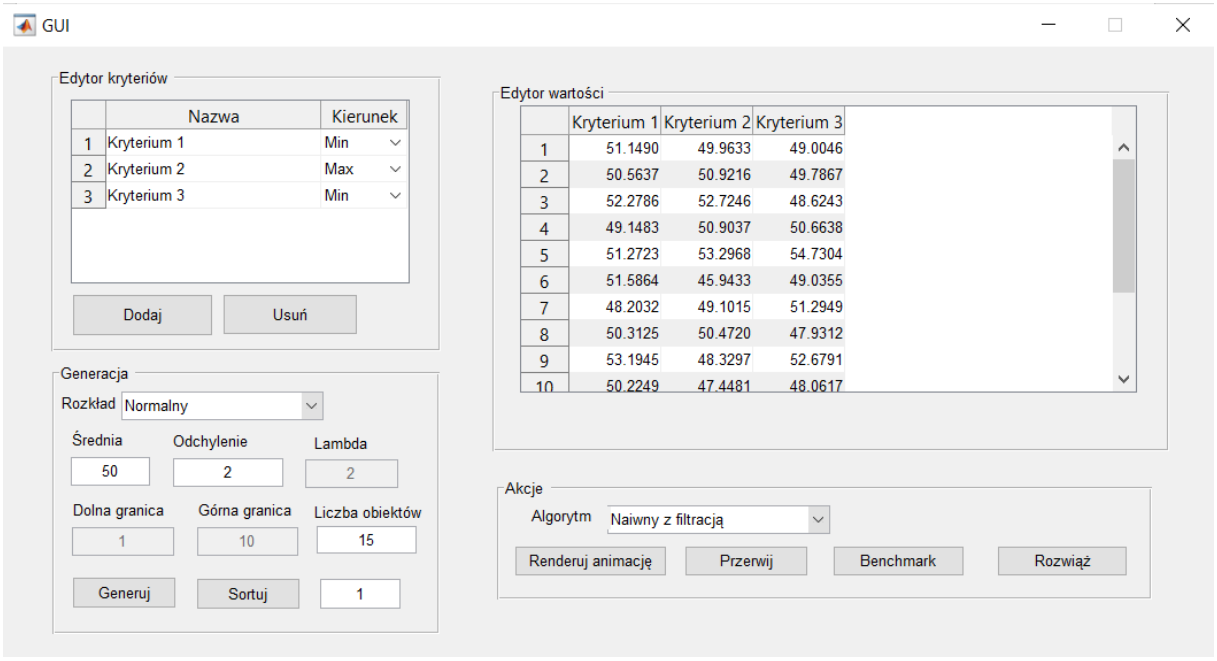

Rozwiązanie problemu
Zdominowane
Niezdominowane
Kryterium 3
Kryterium 2
Kryterium 1
0
1
2
3
4
5
6

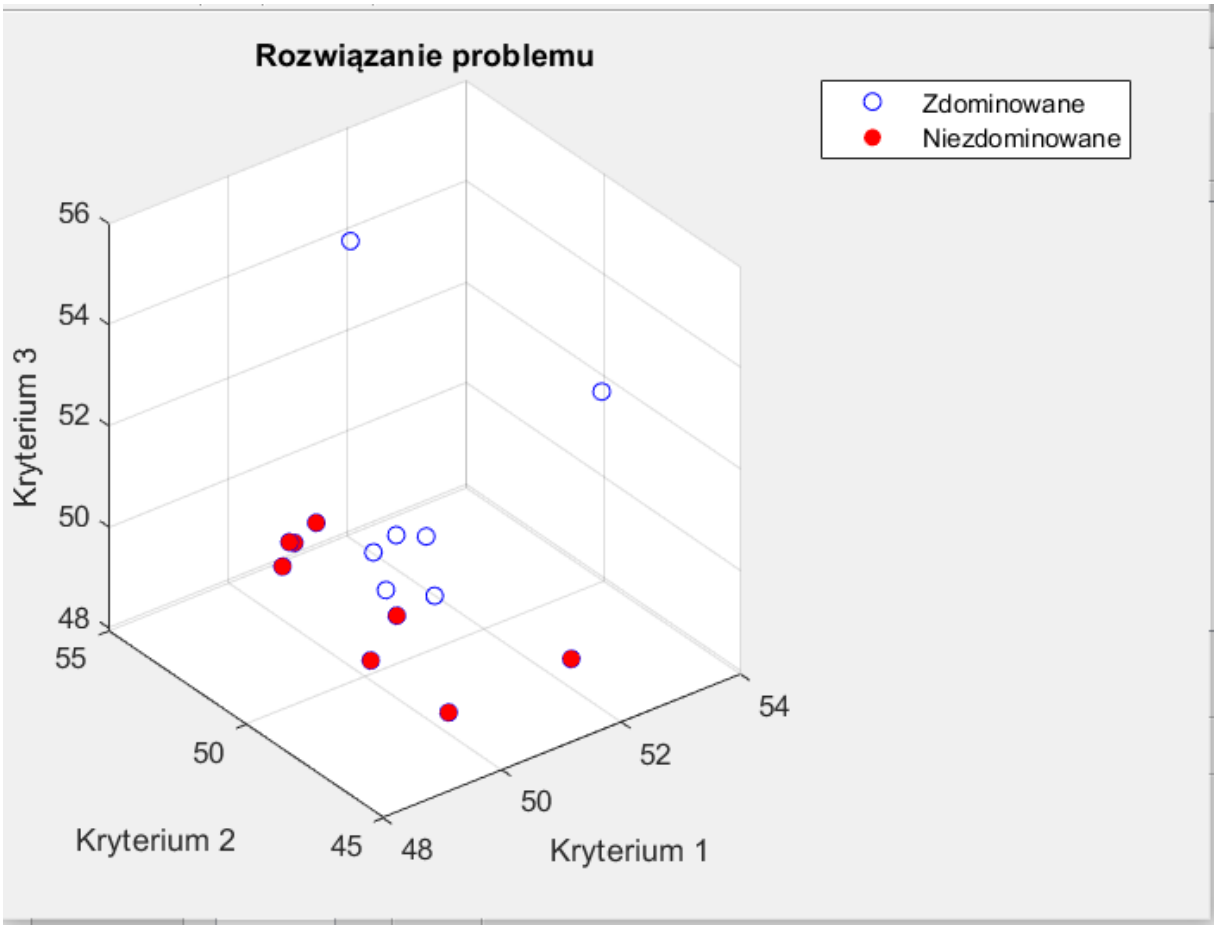

Benchmark: Execution Time
×10^-4
Execution Time (s)
0
0.5
1
1.5
2
2.5
3
3.5
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata

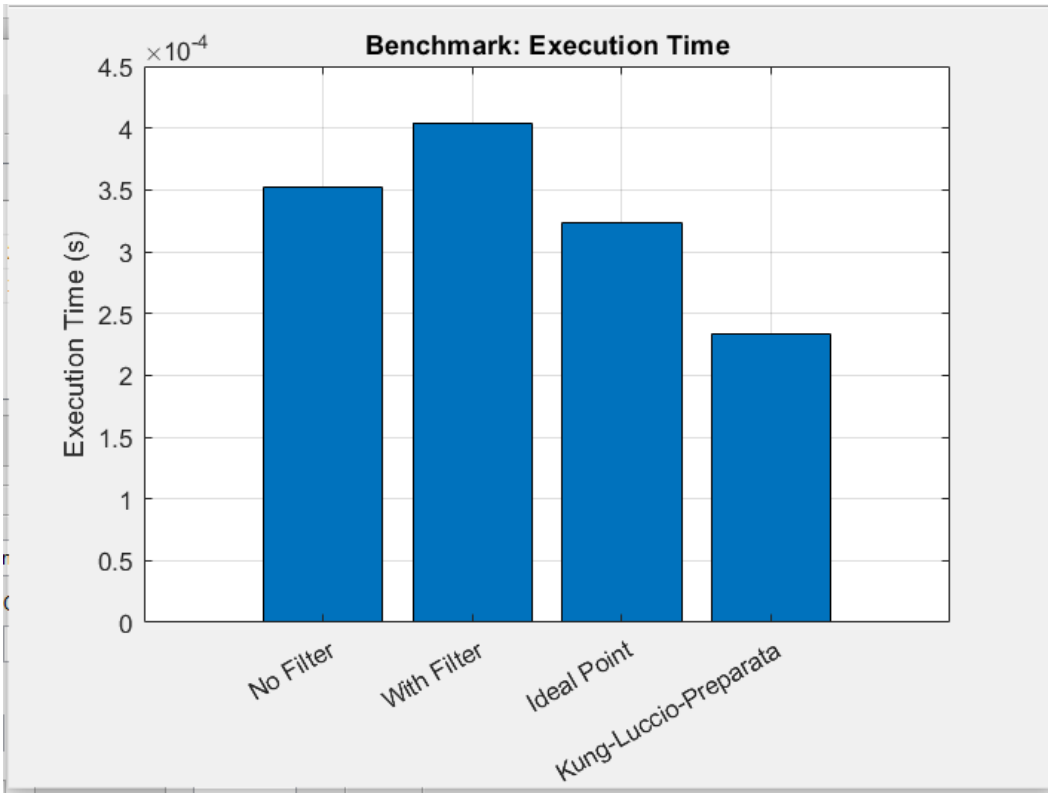

Benchmark: Point And Coordinates Comparisons
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata
700
600
500
400
300
200
100
0
Number of Points Comparisons
Number of Coordinates Comparisons

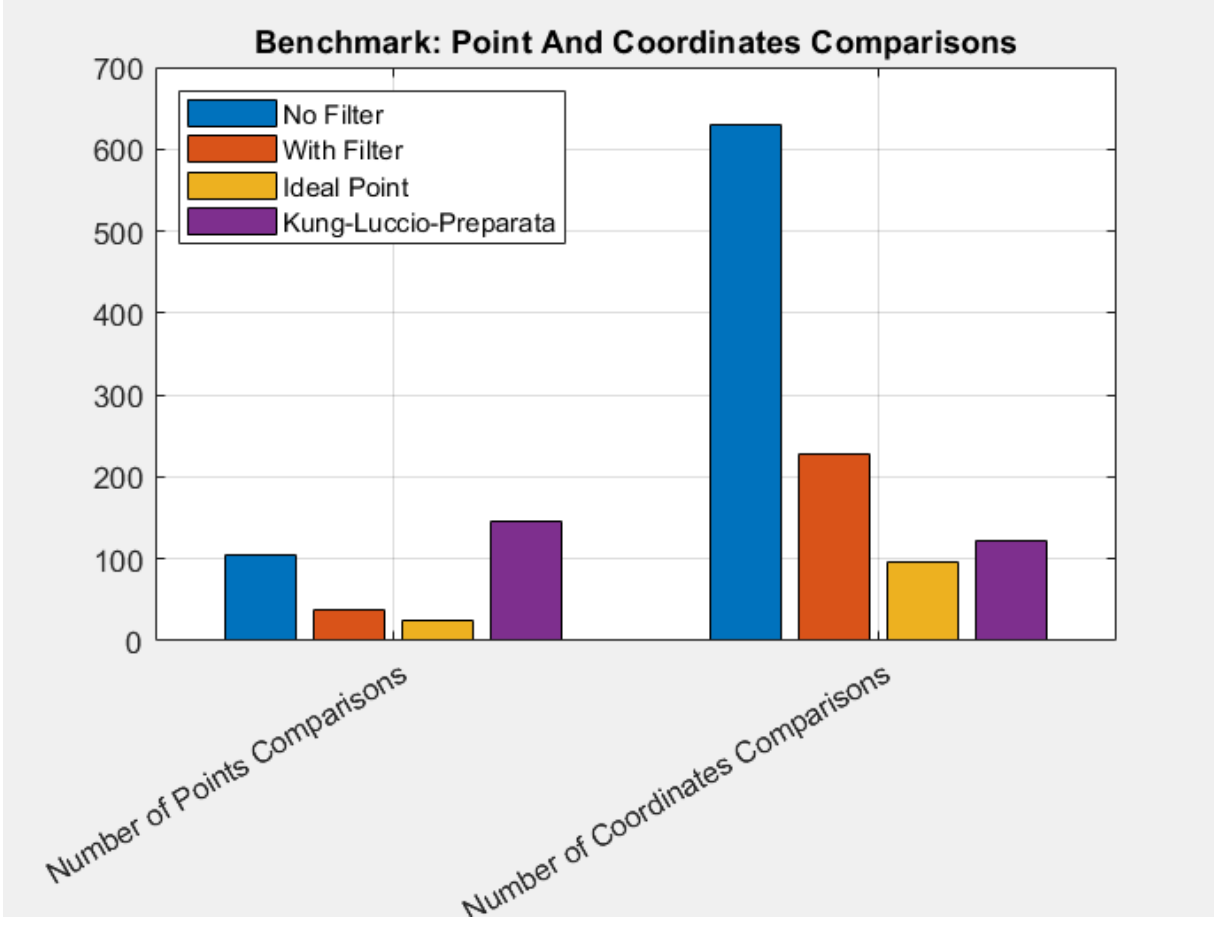

Distance to Ideal with Standard Deviation
Distance to Ideal
5
4
3
2
1
0
No Filter
With Filter
Kung-Luccio-Preparata

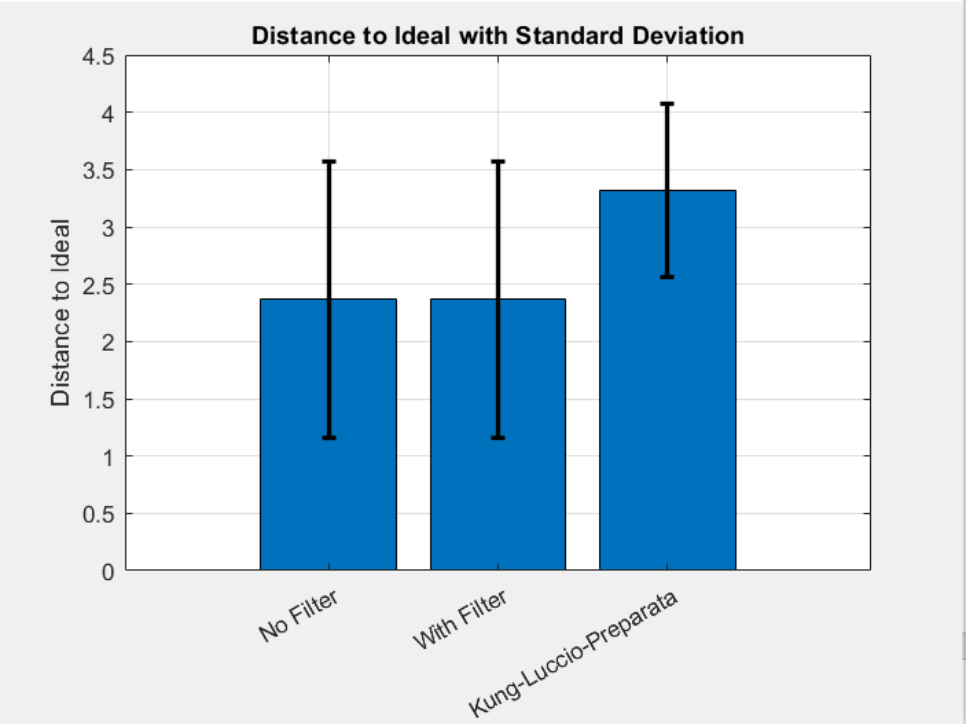
- Rozkład normalny, algorytm naiwny z filtracją



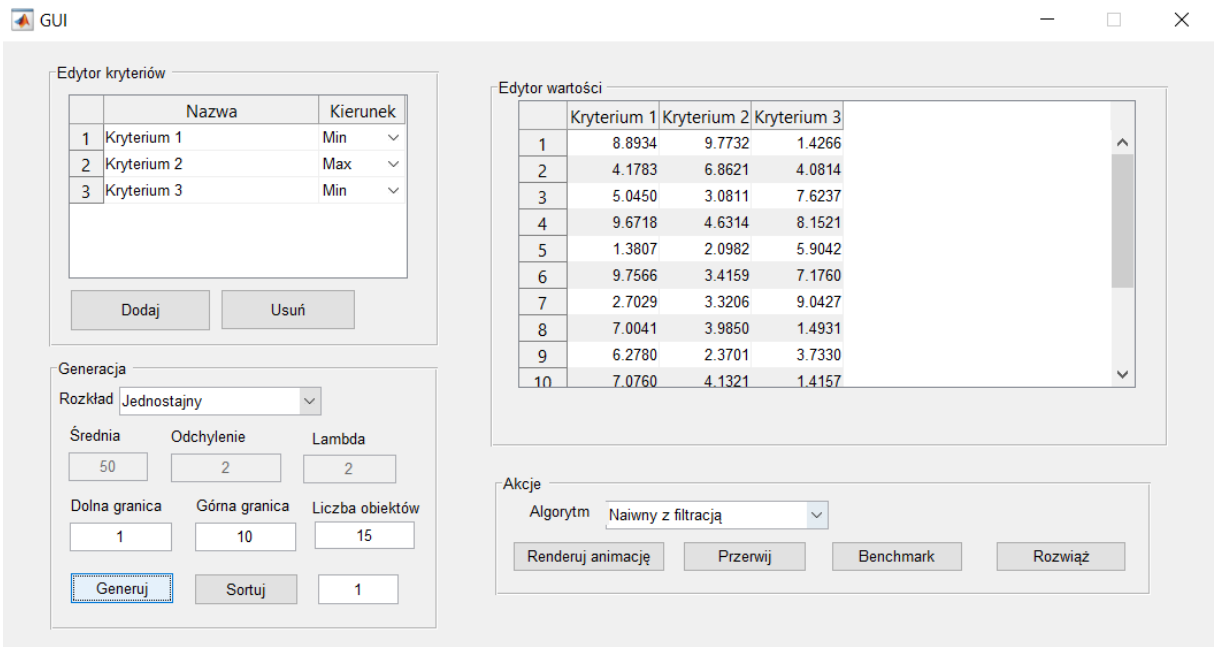


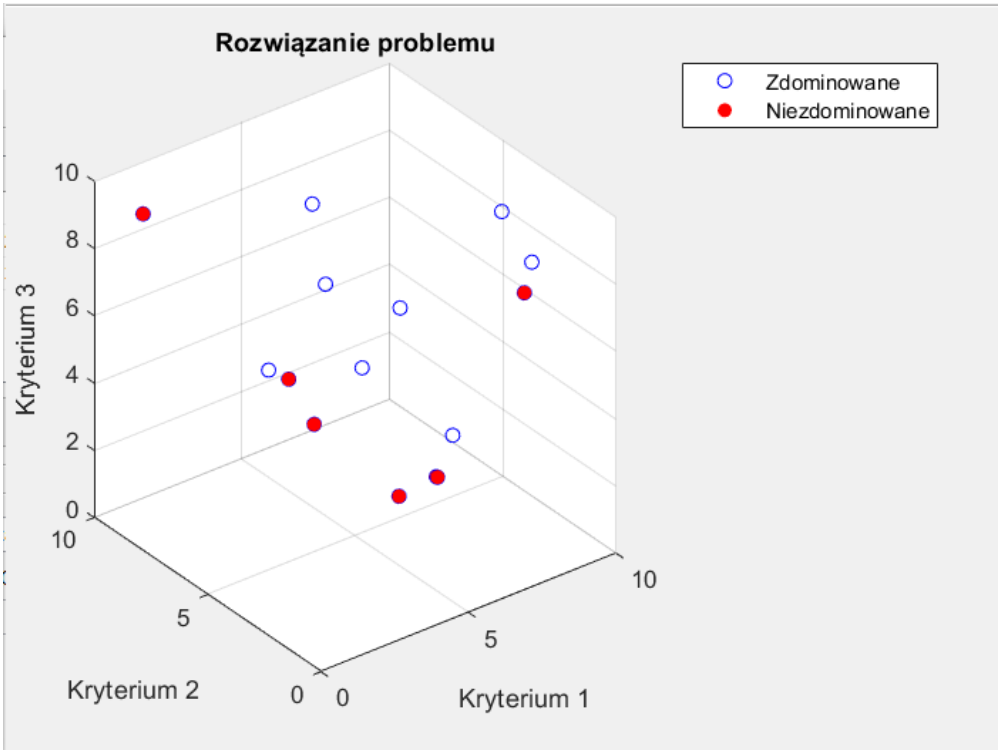

Benchmark: Execution Time
×10^-4
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Execution Time (s)
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata

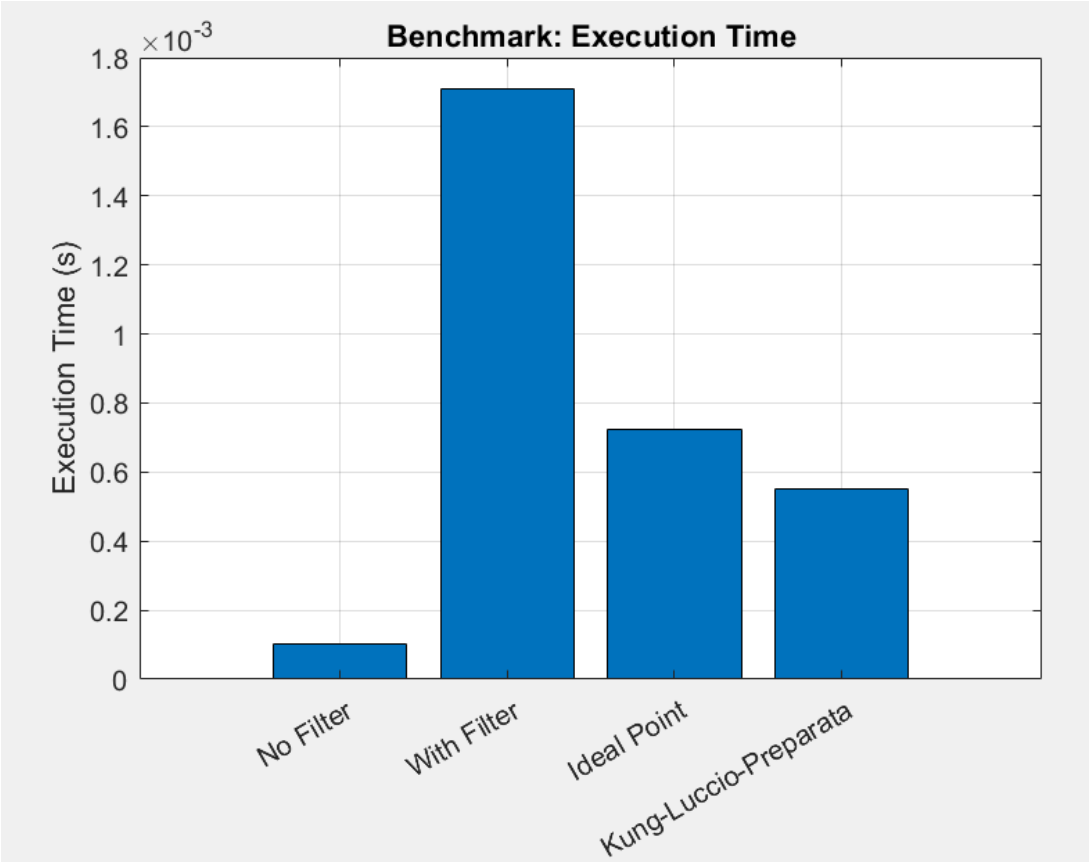

Benchmark: Point And Coordinates Comparisons
700
600
500
400
300
200
100
0
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata
Number of Points Comparisons
Number of Coordinates Comparisons

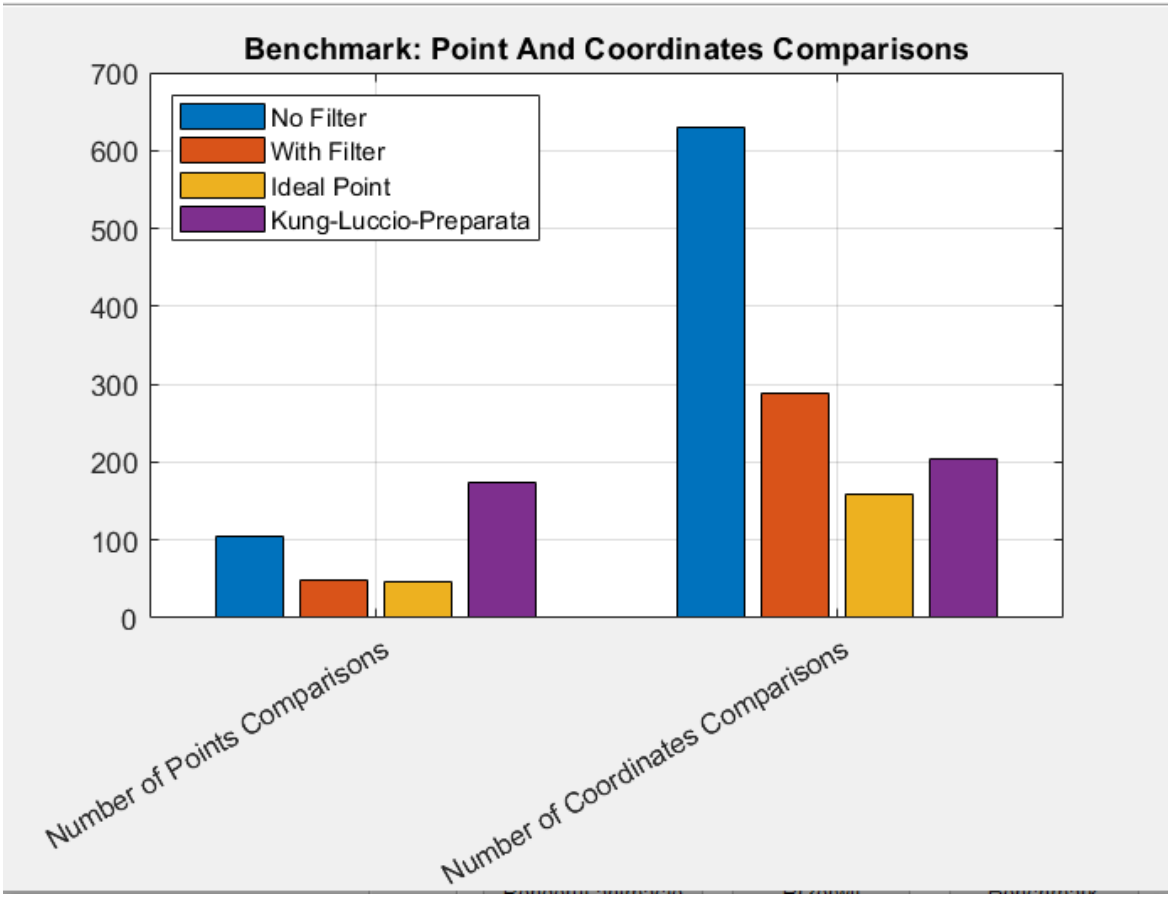


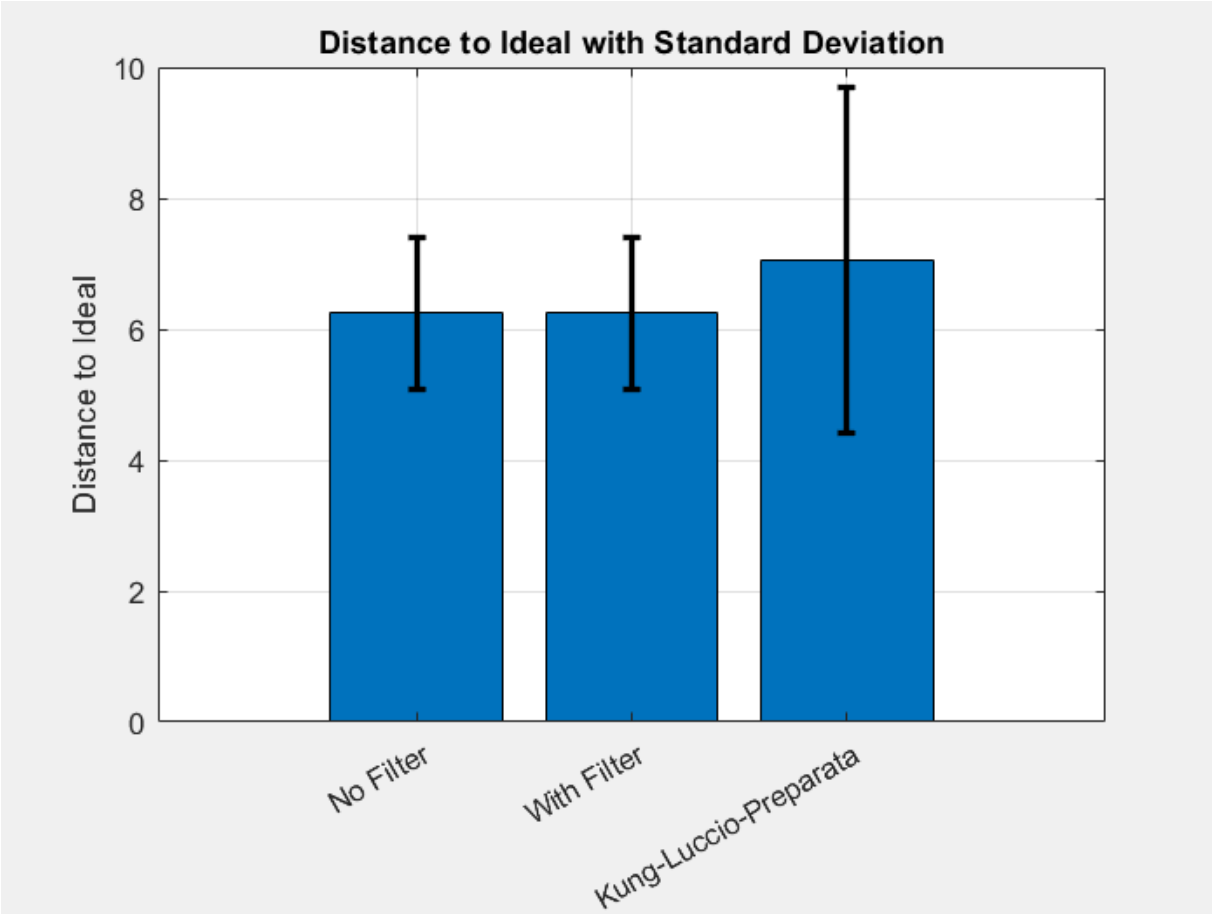
- Rozkład jednostajny, algorytm naiwny z filtracją



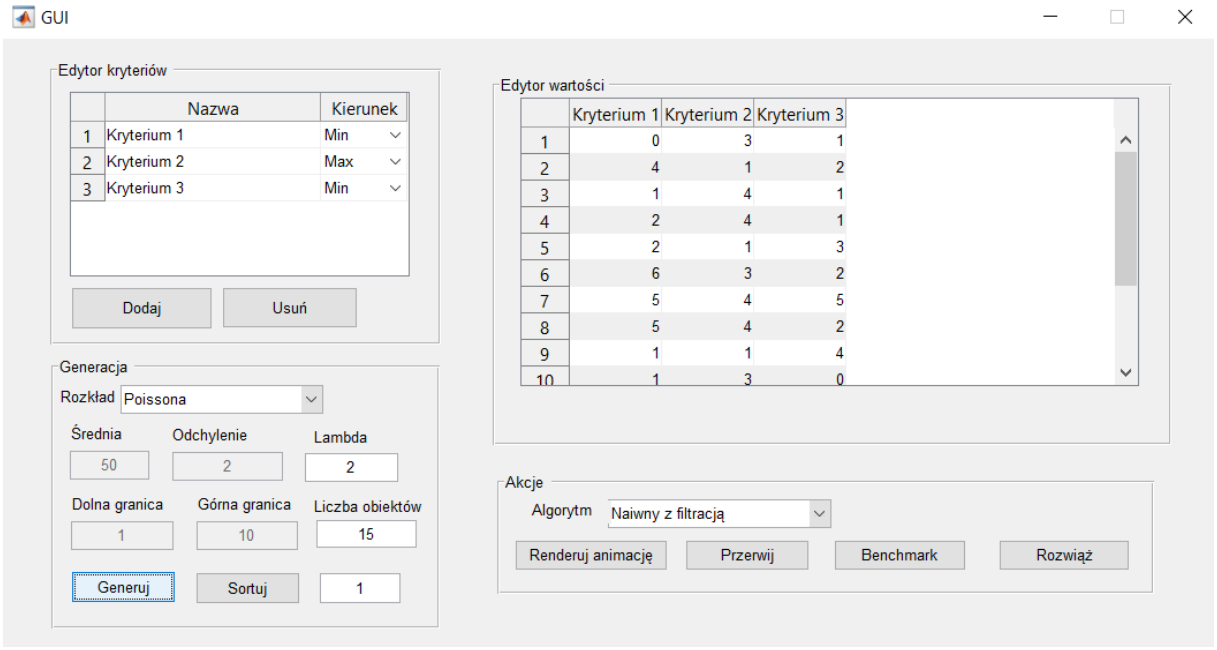

Rozwiązanie problemu
Zdominowane
Niezdominowane
Kryterium 3
Kryterium 2
Kryterium 1
0
2
4
6
8
10
0
5
10
0
5
10

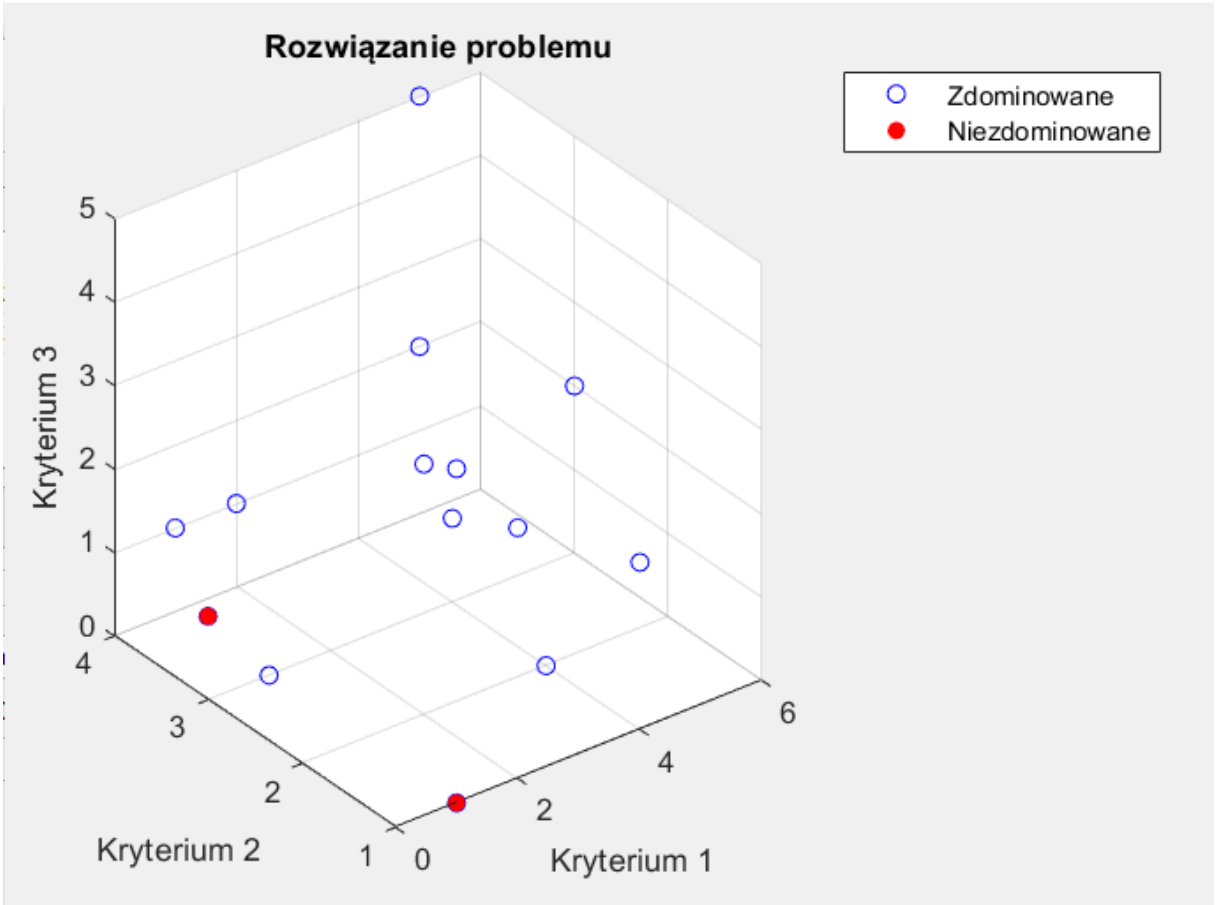

Benchmark: Execution Time
×10-3
Execution Time (s)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata

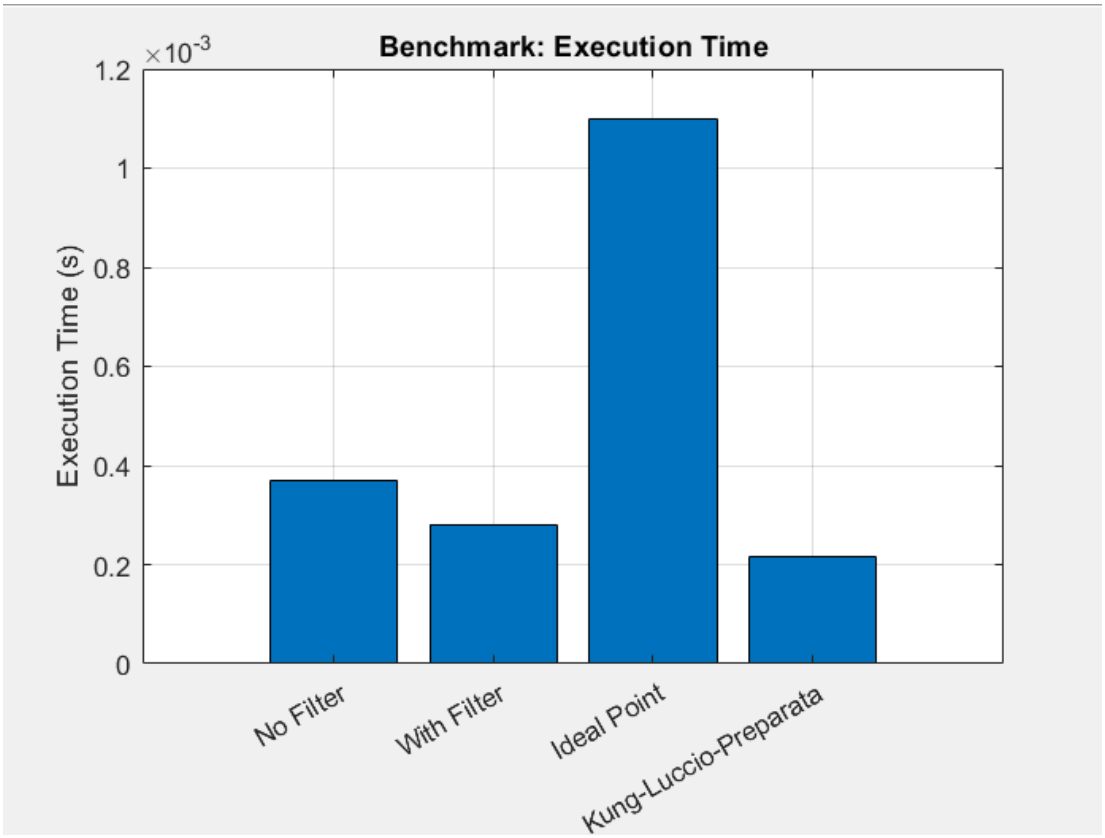

Benchmark: Point And Coordinates Comparisons
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata
700
600
500
400
300
200
100
0
Number of Points Comparisons
Number of Coordinates Comparisons

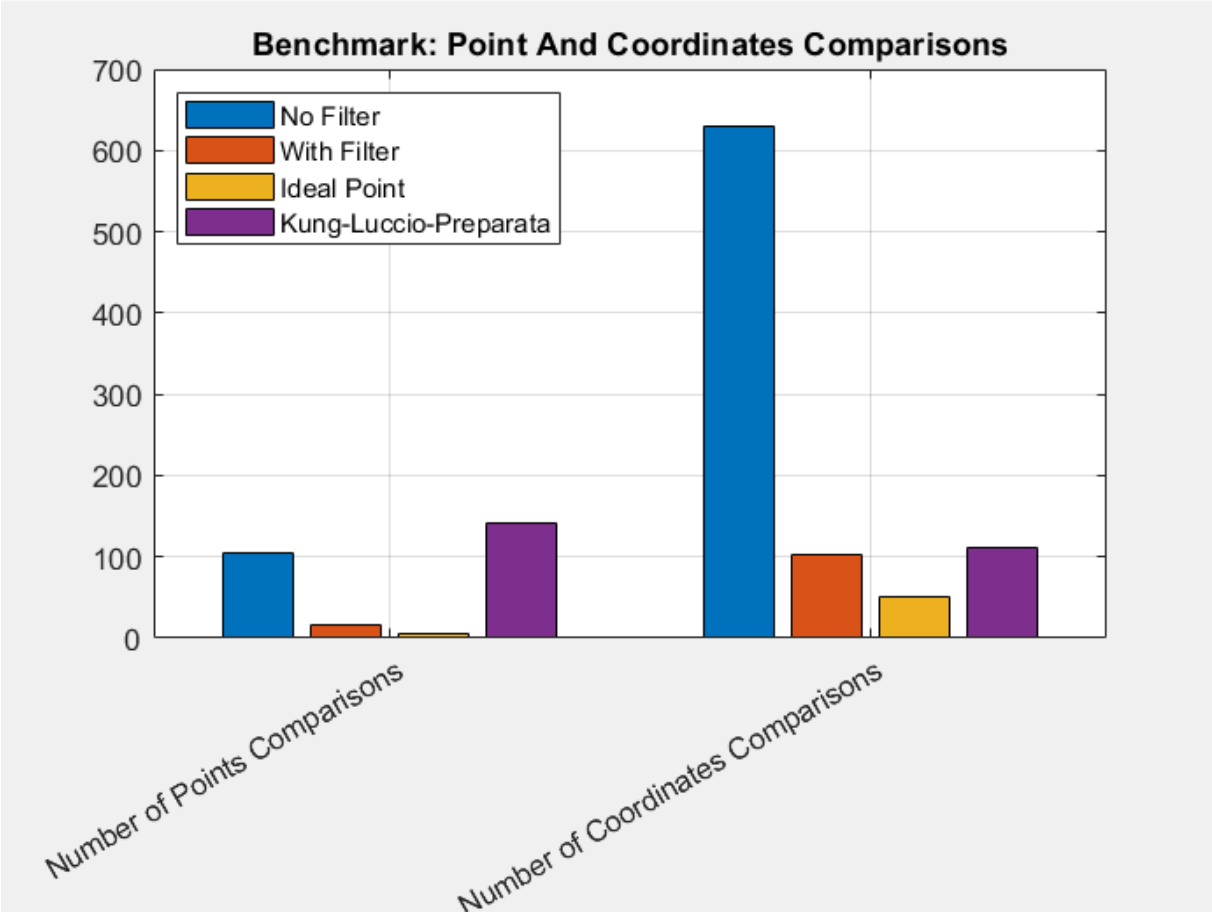

Distance to Ideal with Standard Deviation
Distance to Ideal
10
8
6
4
2
0
No Filter
With Filter
Kung-Luccio-Preparata

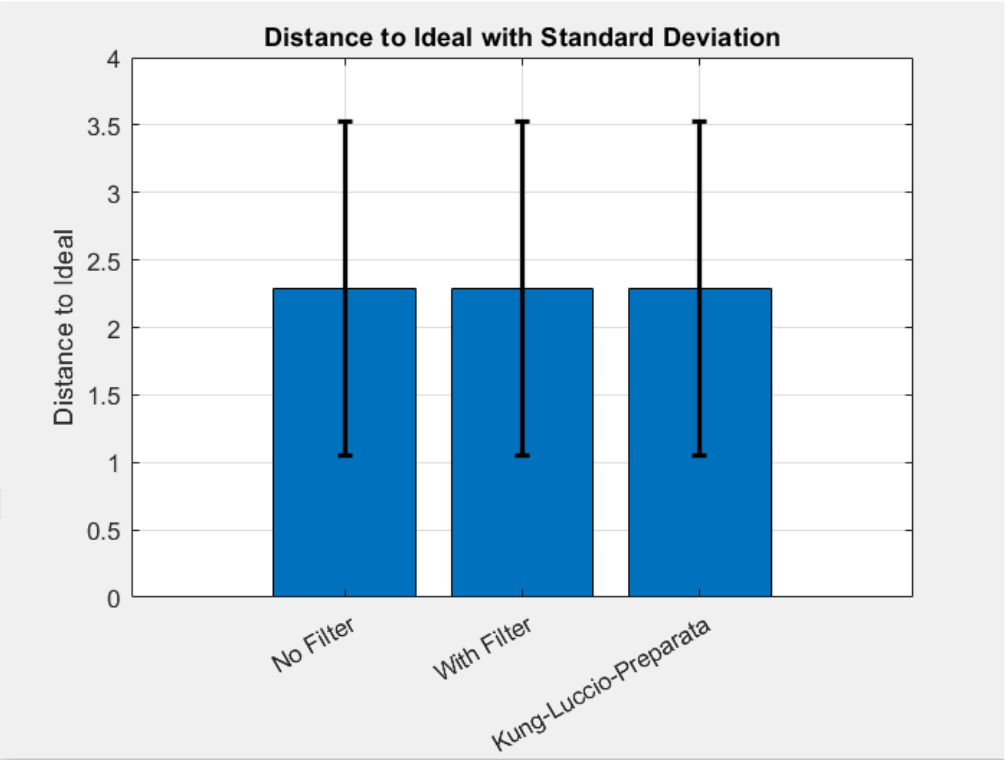
- Rozkład Poissona, algorytm naiwny z filtracją



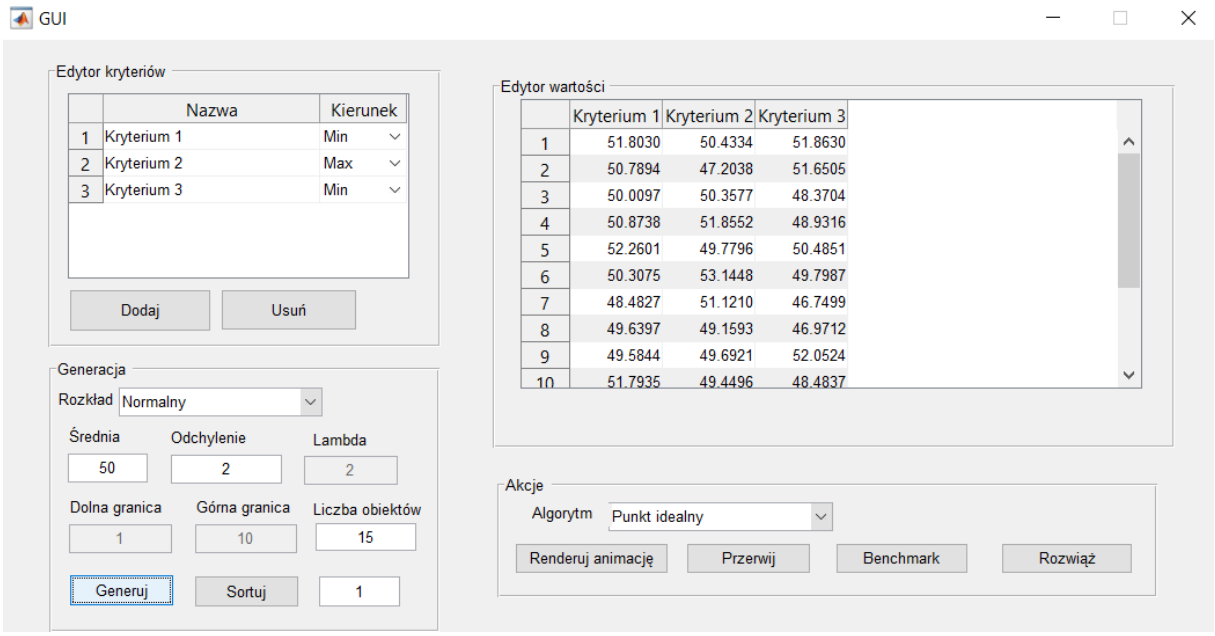


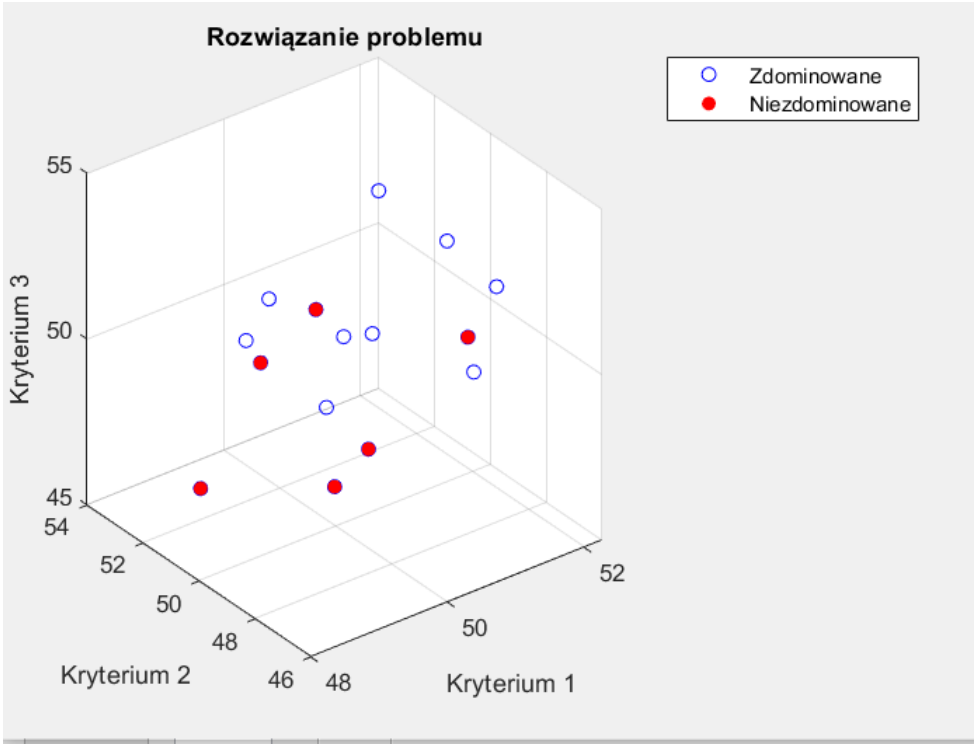

Benchmark: Execution Time
×10^-3
1.2
1
0.8
0.6
0.4
0.2
0
Execution Time (s)
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata

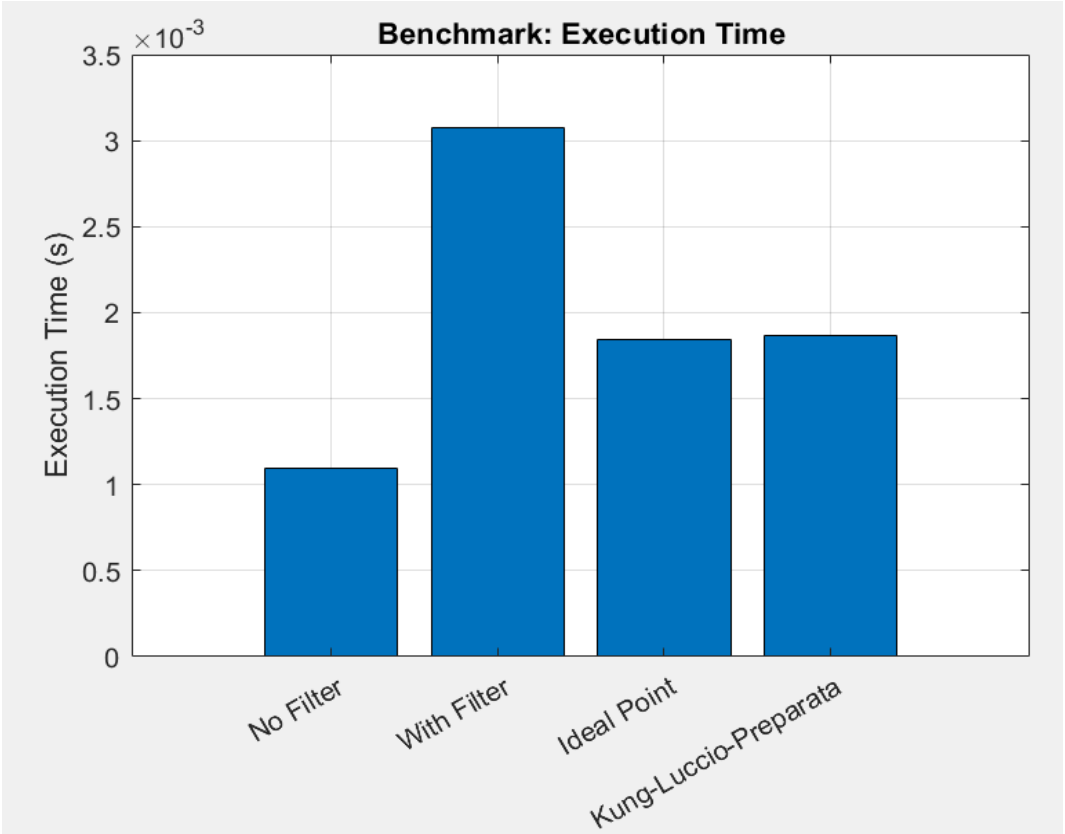

Benchmark: Point And Coordinates Comparisons
700
600
500
400
300
200
100
0
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata
Number of Points Comparisons
Number of Coordinates Comparisons

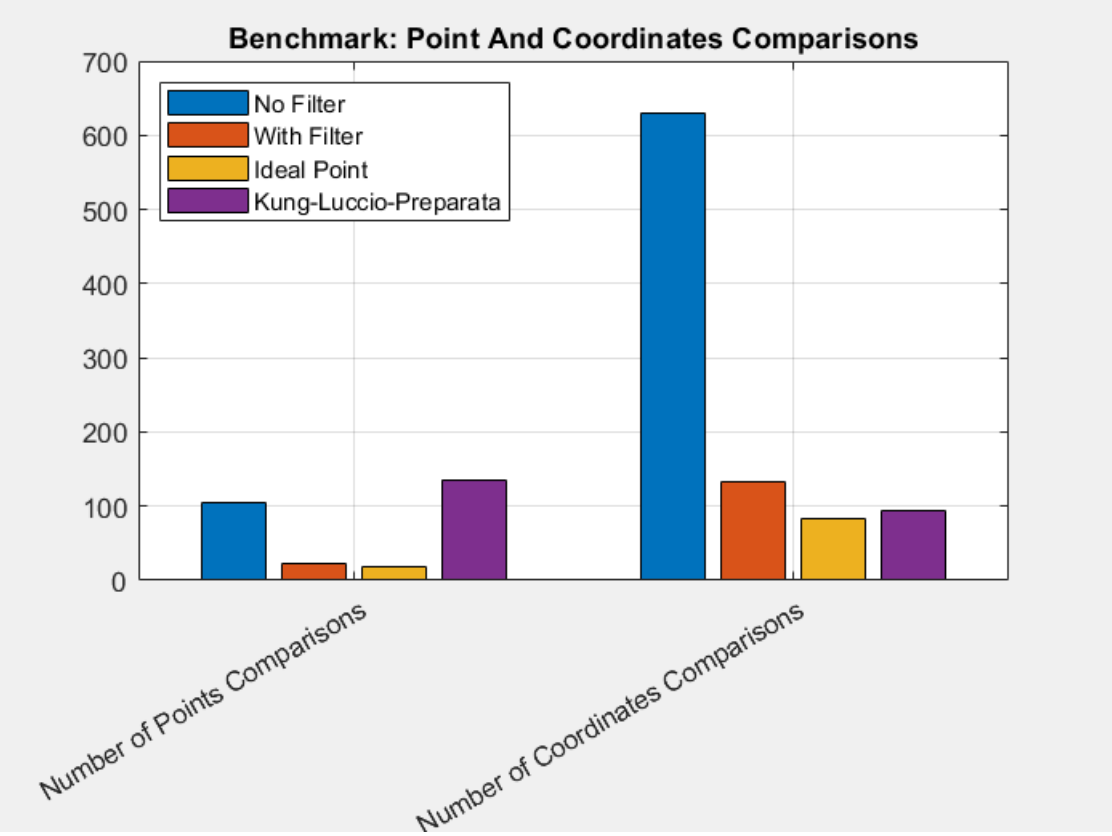


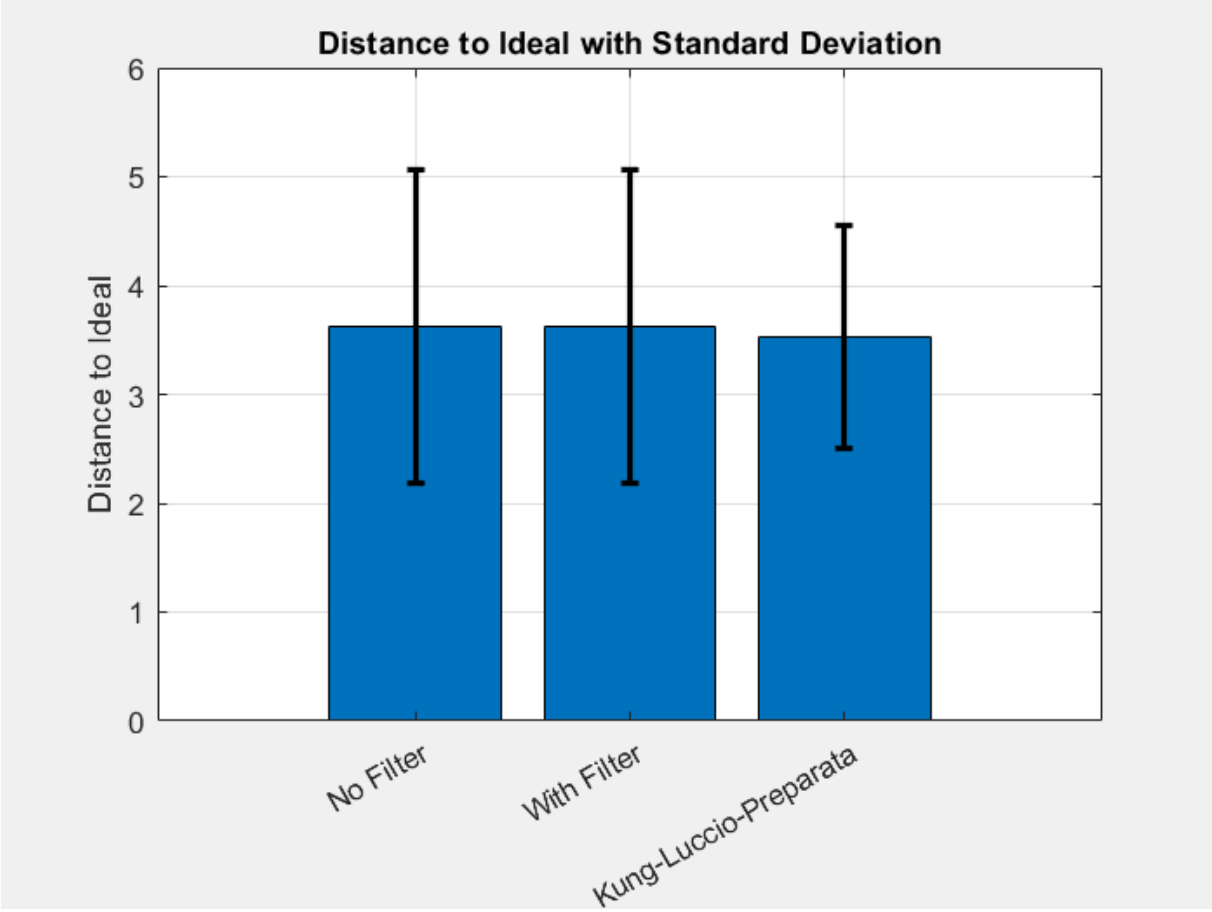
- Rozkład normalny, algorytm punkt idealny



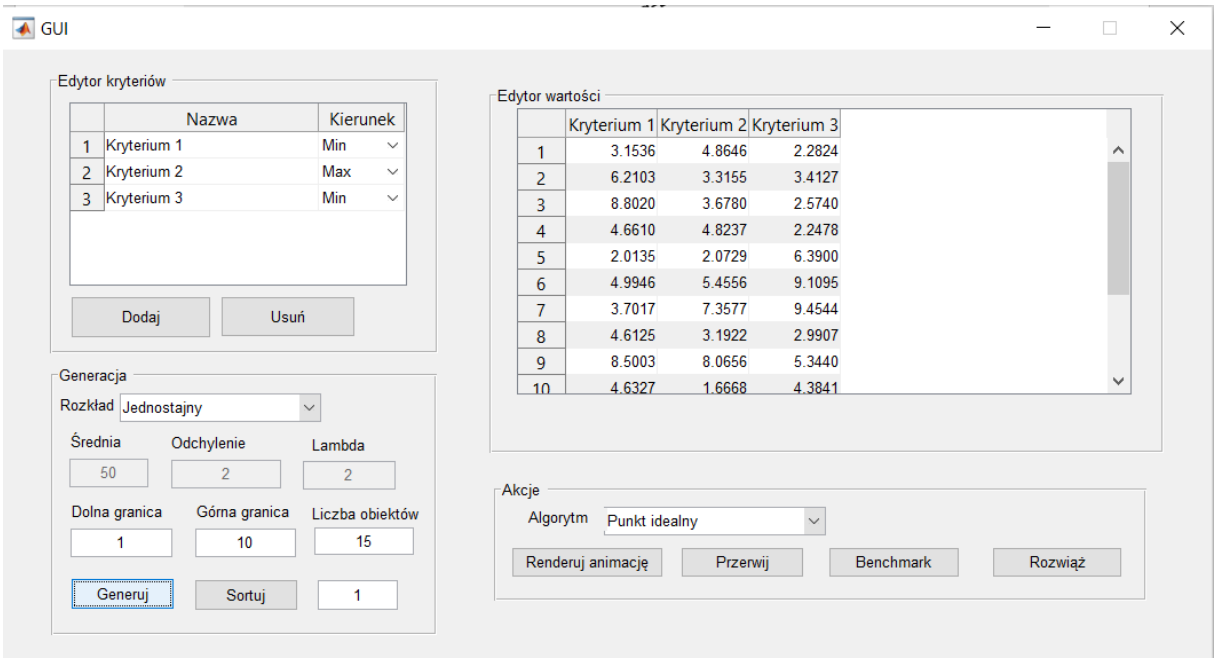

Rozwiązanie problemu
Zdominowane
Niezdominowane
Kryterium 3
55
50
45
54
52
50
48
46
Kryterium 2
48
50
52
Kryterium 1

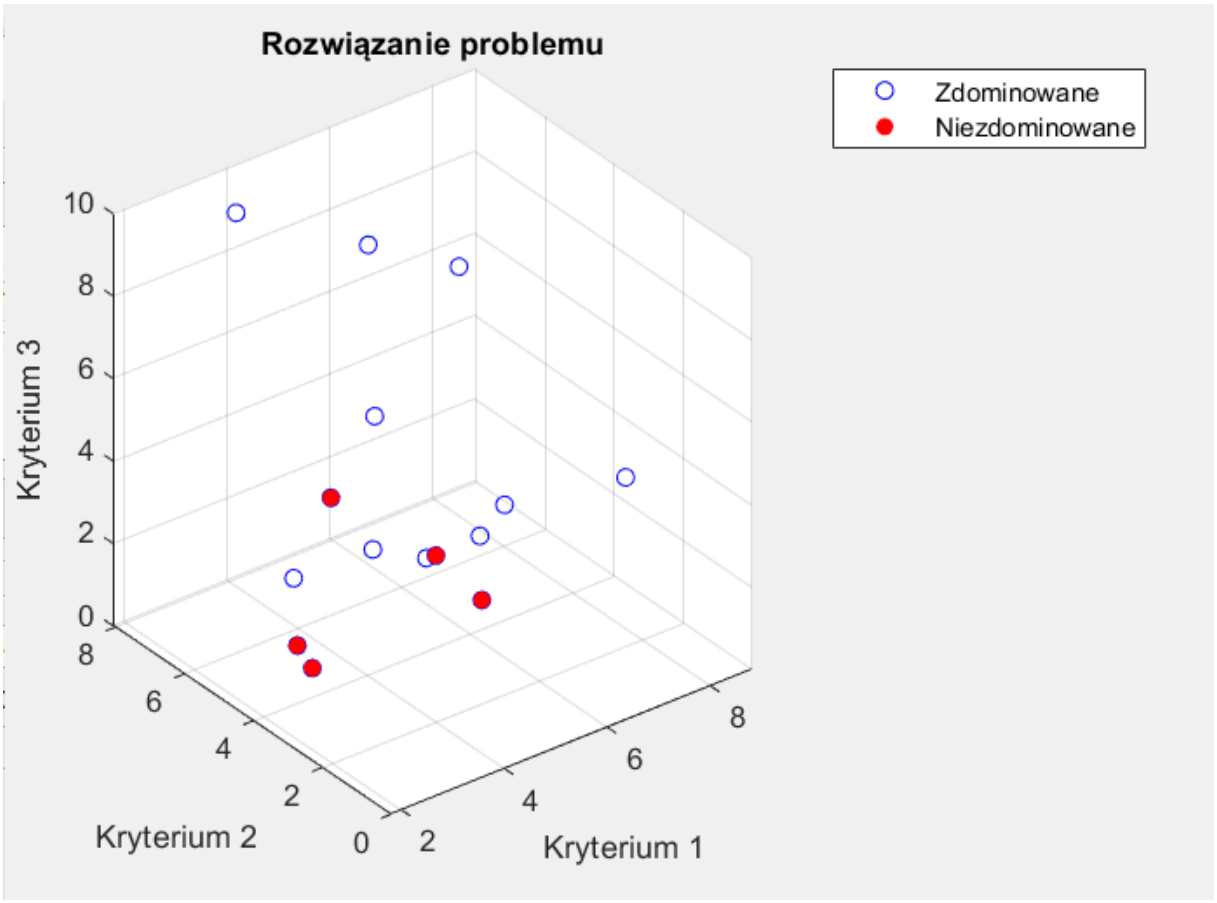

Benchmark: Execution Time
×10-3
Execution Time (s)
3.5
3
2.5
2
1.5
1
0.5
0
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata

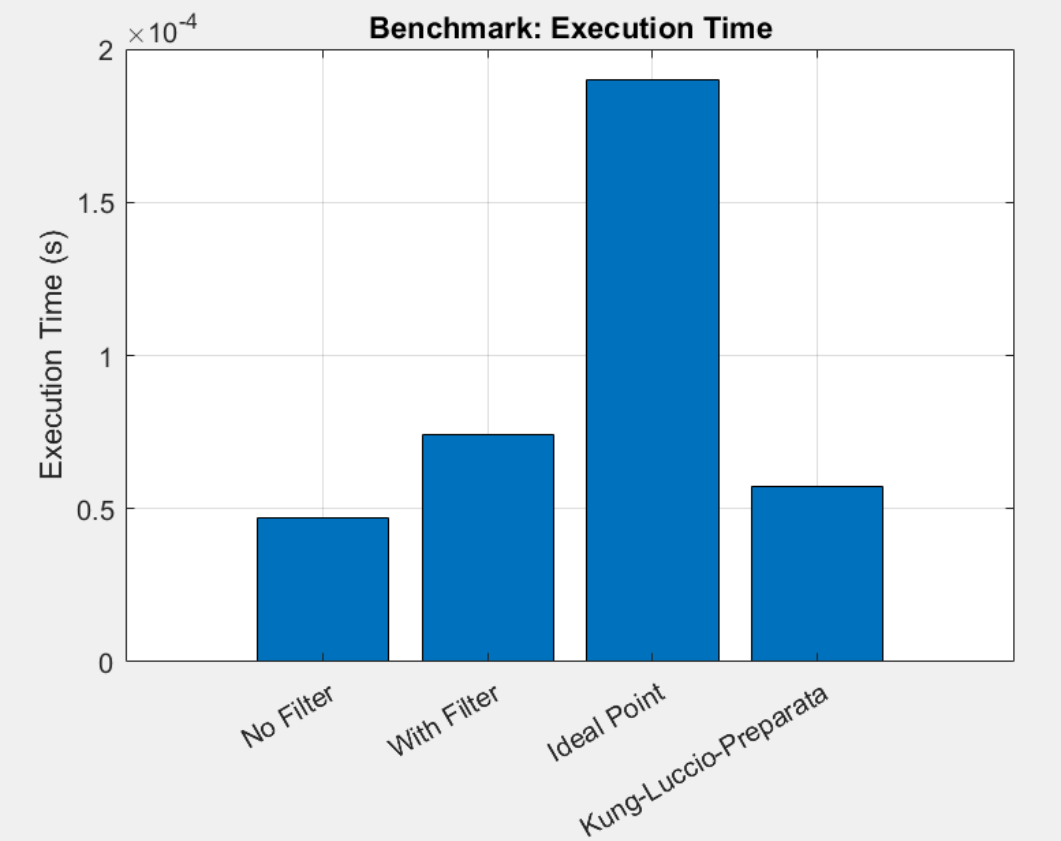

Benchmark: Point And Coordinates Comparisons
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata
700
600
500
400
300
200
100
0
Number of Points Comparisons
Number of Coordinates Comparisons

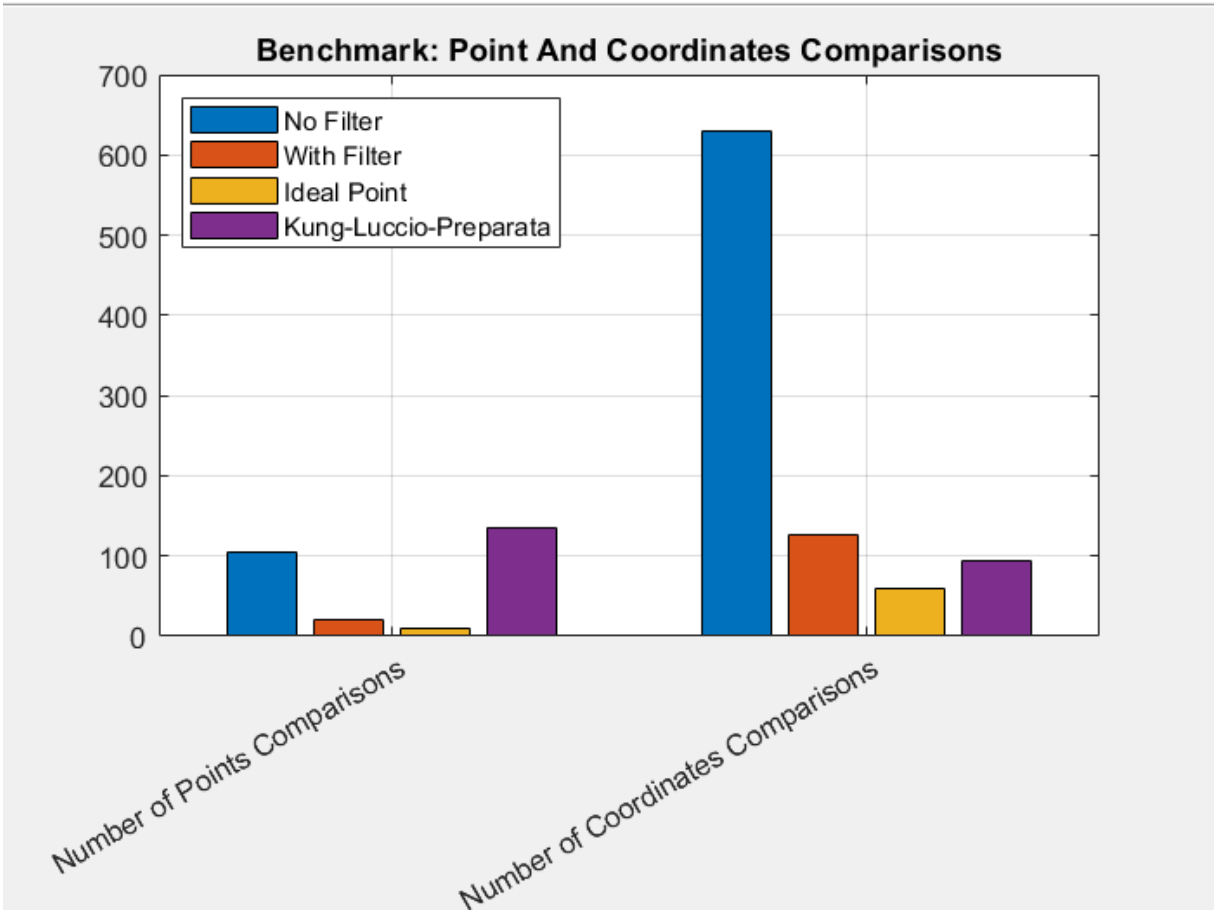

Distance to Ideal with Standard Deviation
Distance to Ideal
6
5
4
3
2
1
0
No Filter
With Filter
Kung-Luccio-Preparata

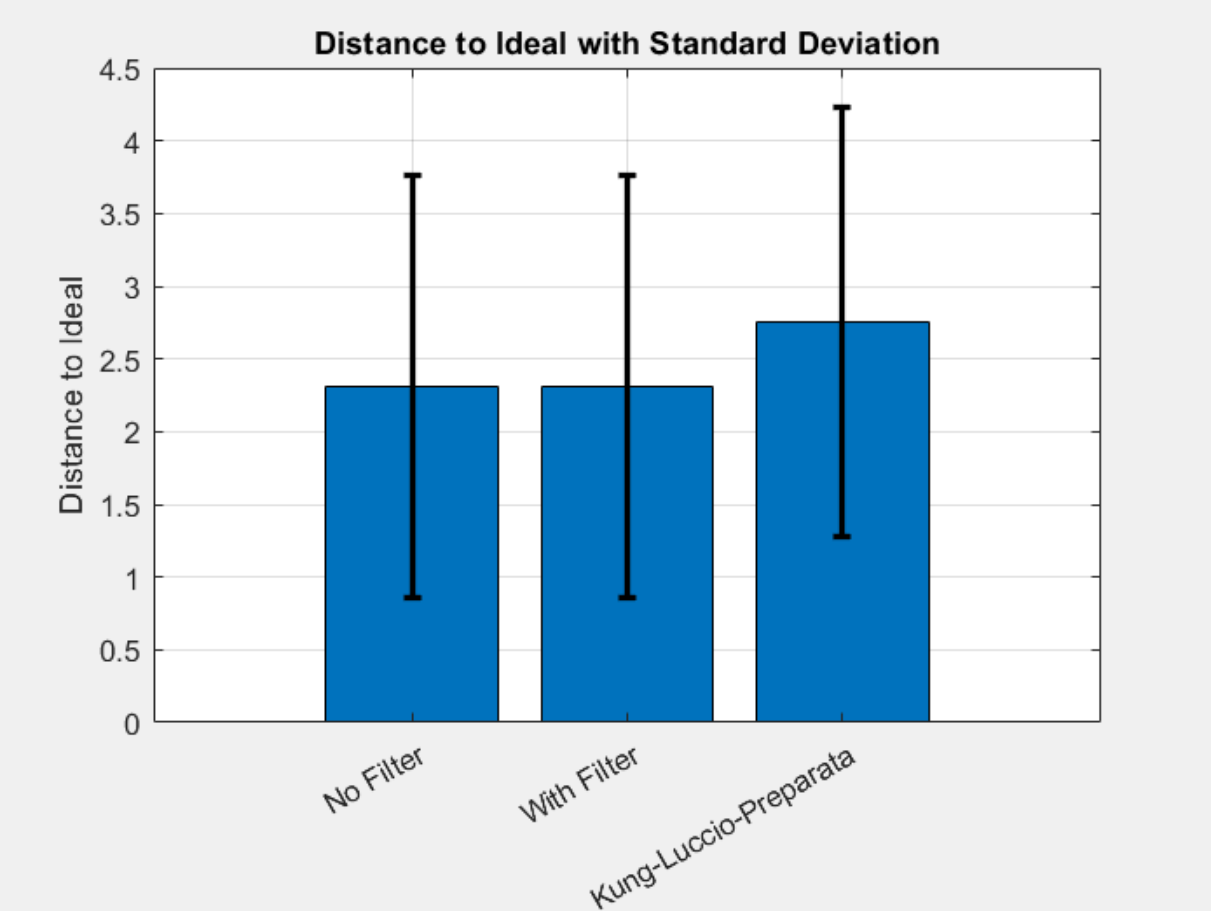
- Rozkład jednostajny, algorytm punkt idealny



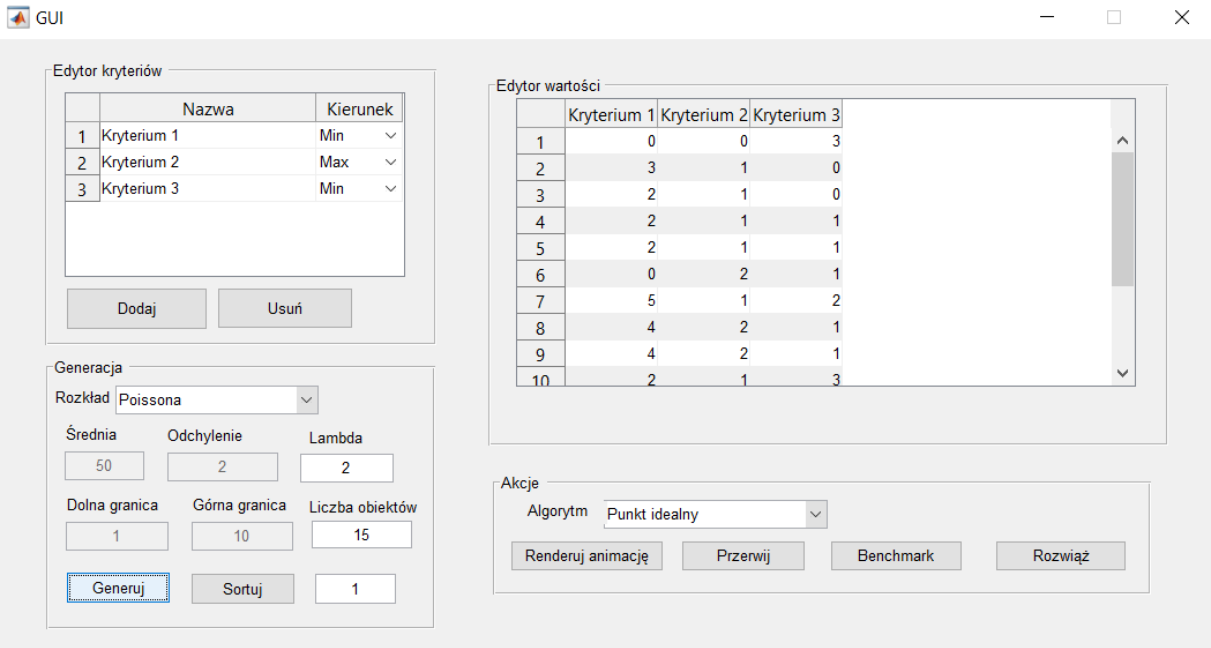


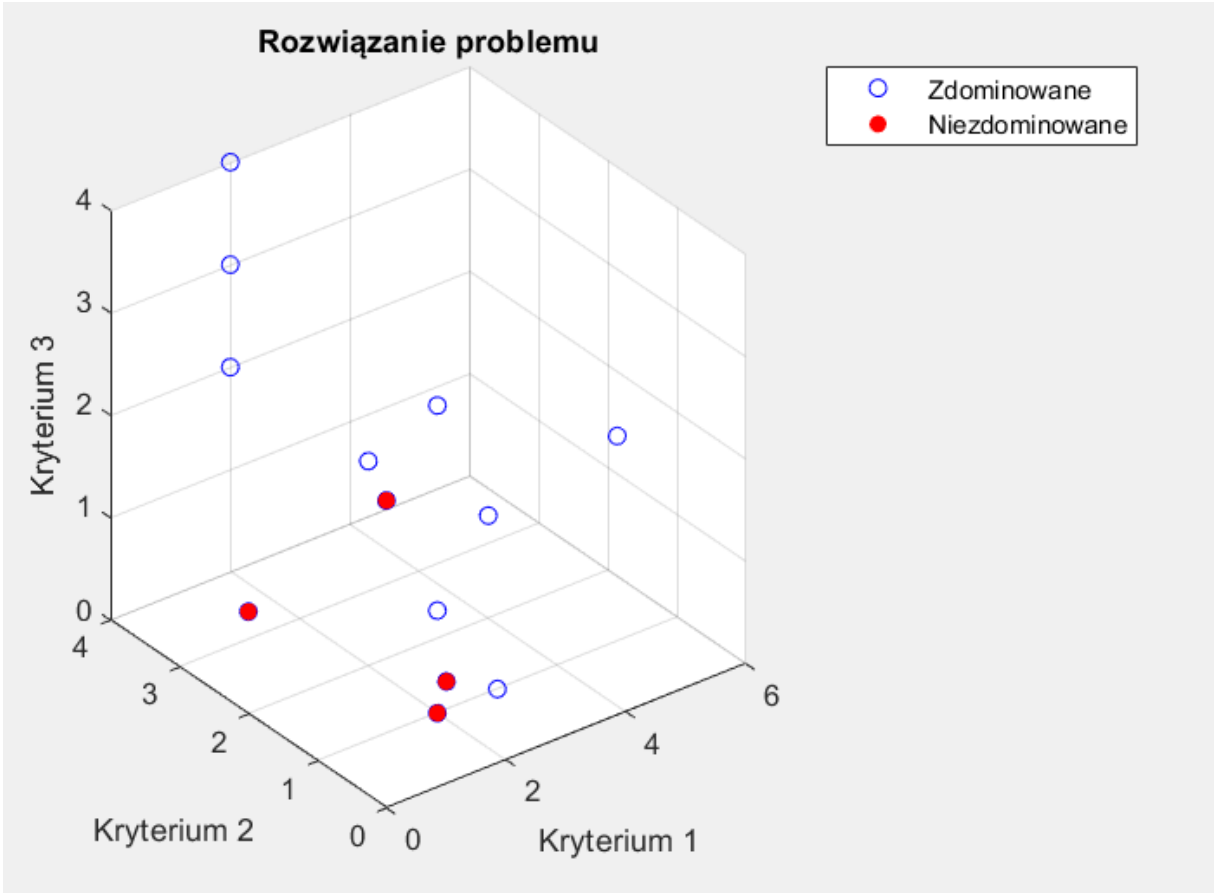

Benchmark: Execution Time
×10^-4
2
1.5
1
0.5
0
Execution Time (s)
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata

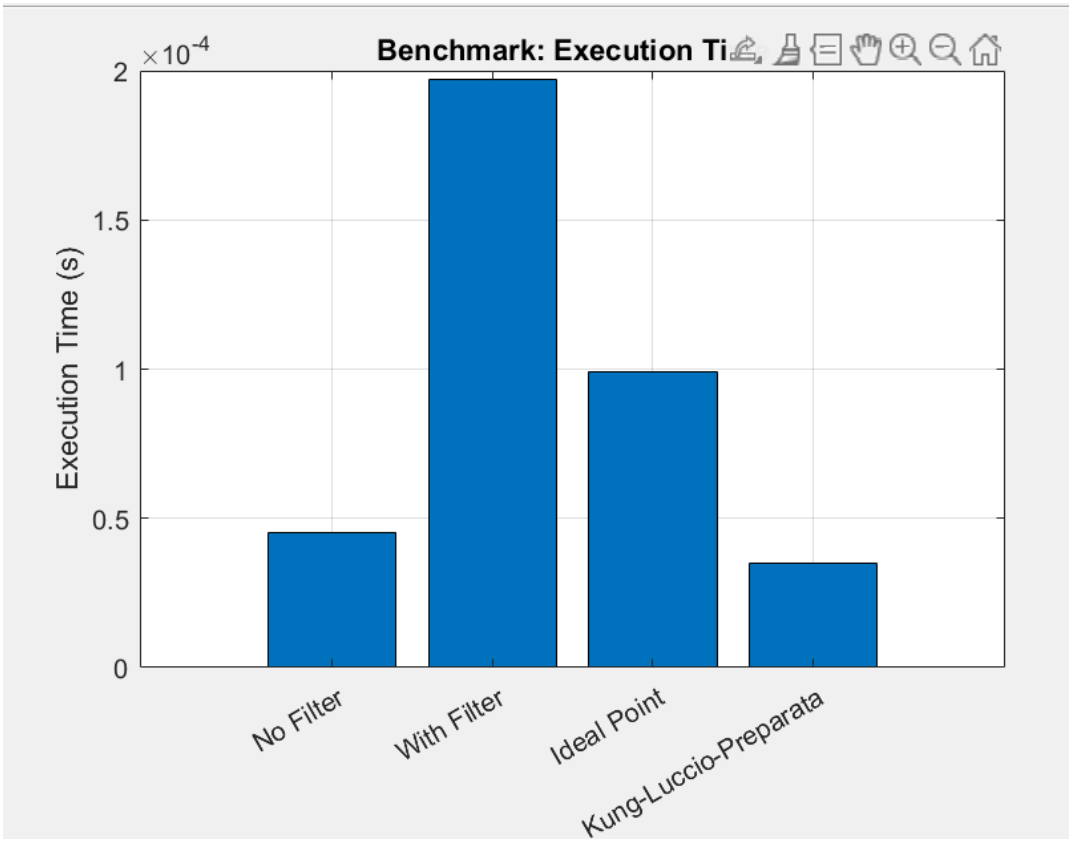

Benchmark: Point And Coordinates Comparisons
700
600
500
400
300
200
100
0
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata
Number of Points Comparisons
Number of Coordinates Comparisons

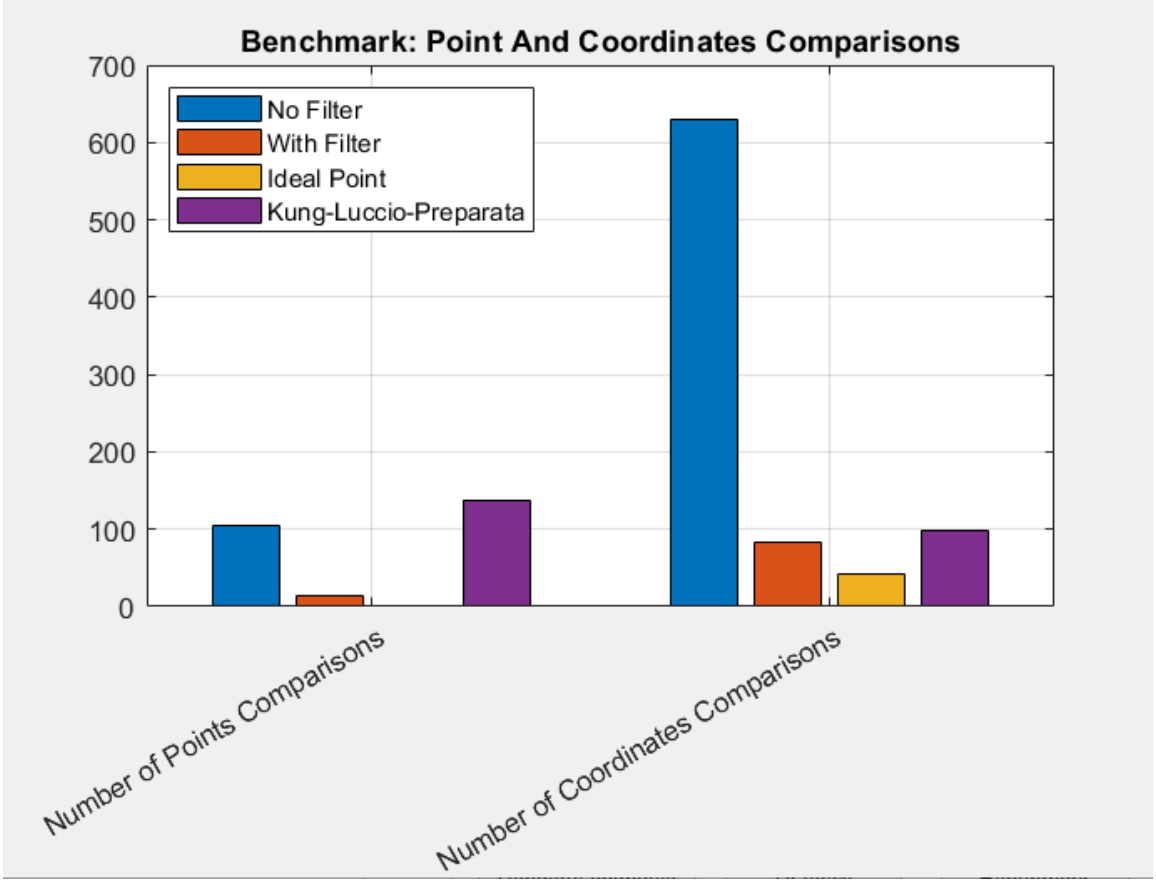


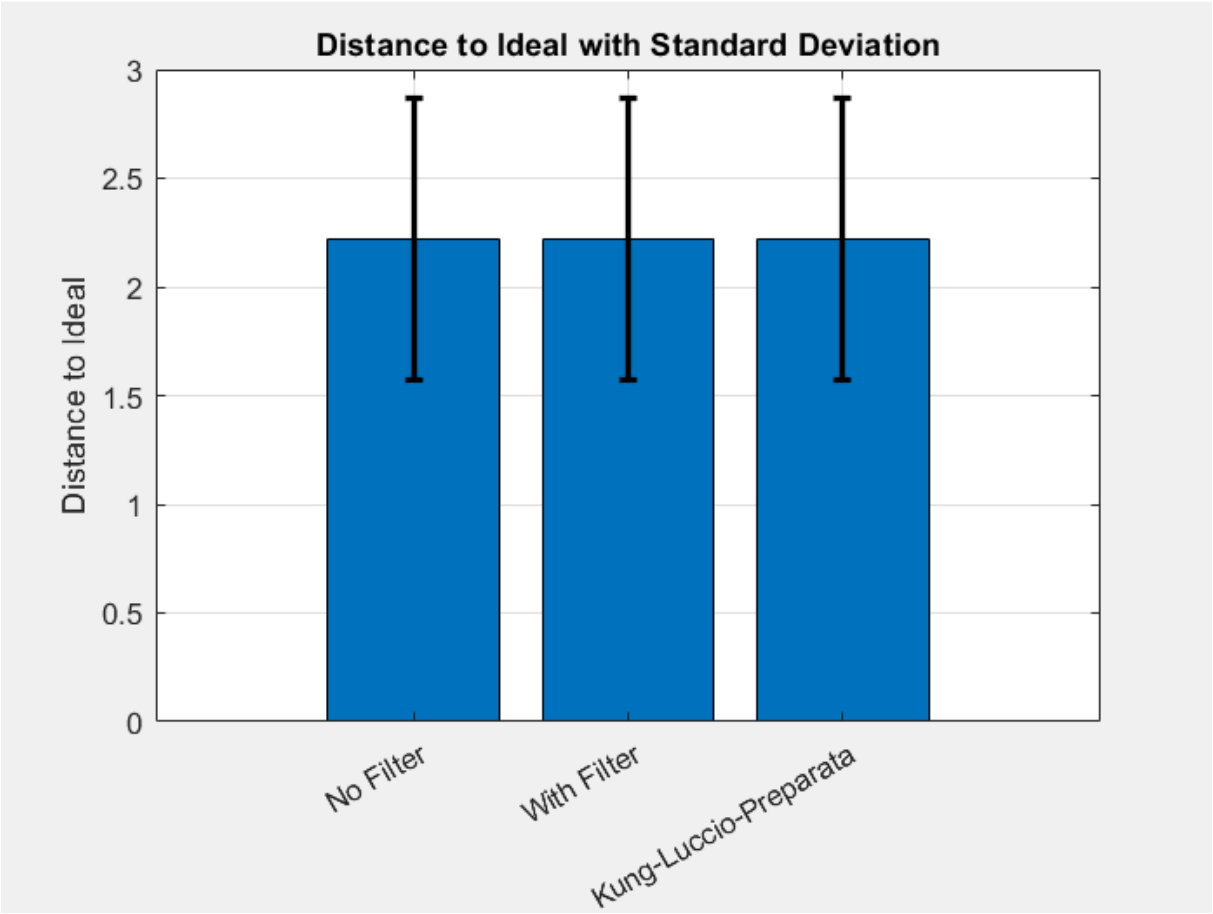
- Rozkład Poissona, algorytm punkt idealny



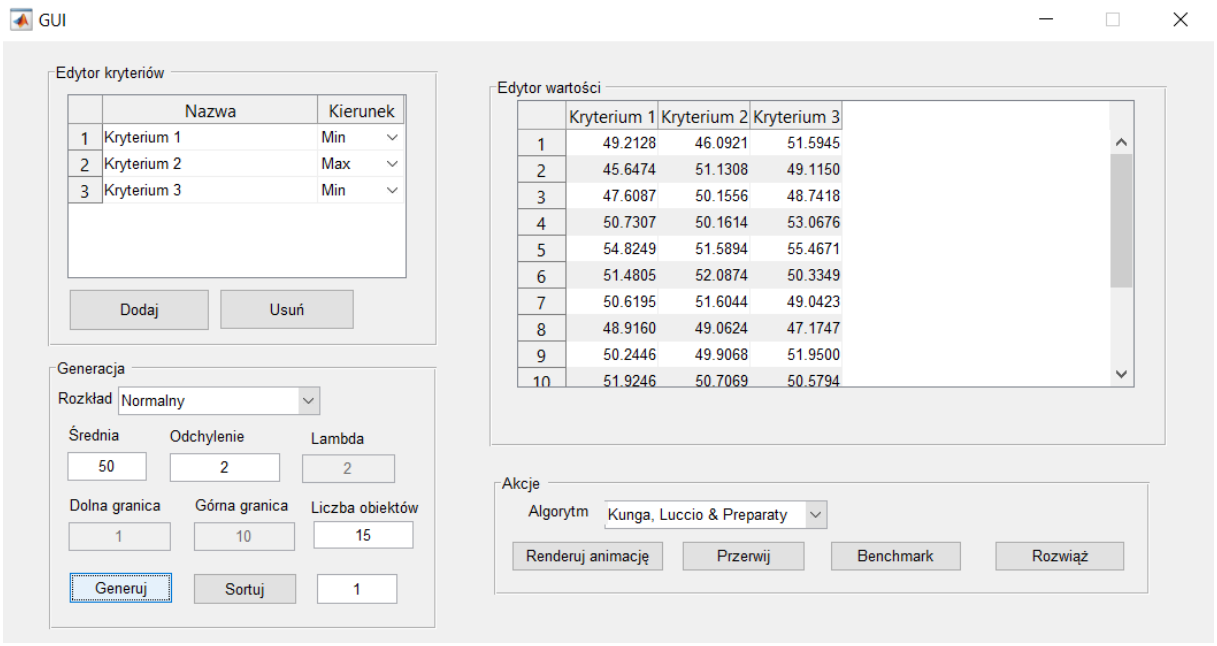

Rozwiązanie problemu
Zdominowane
Niezdominowane
Kryterium 3
Kryterium 2
Kryterium 1
0
1
2
3
4
0
2
4
6

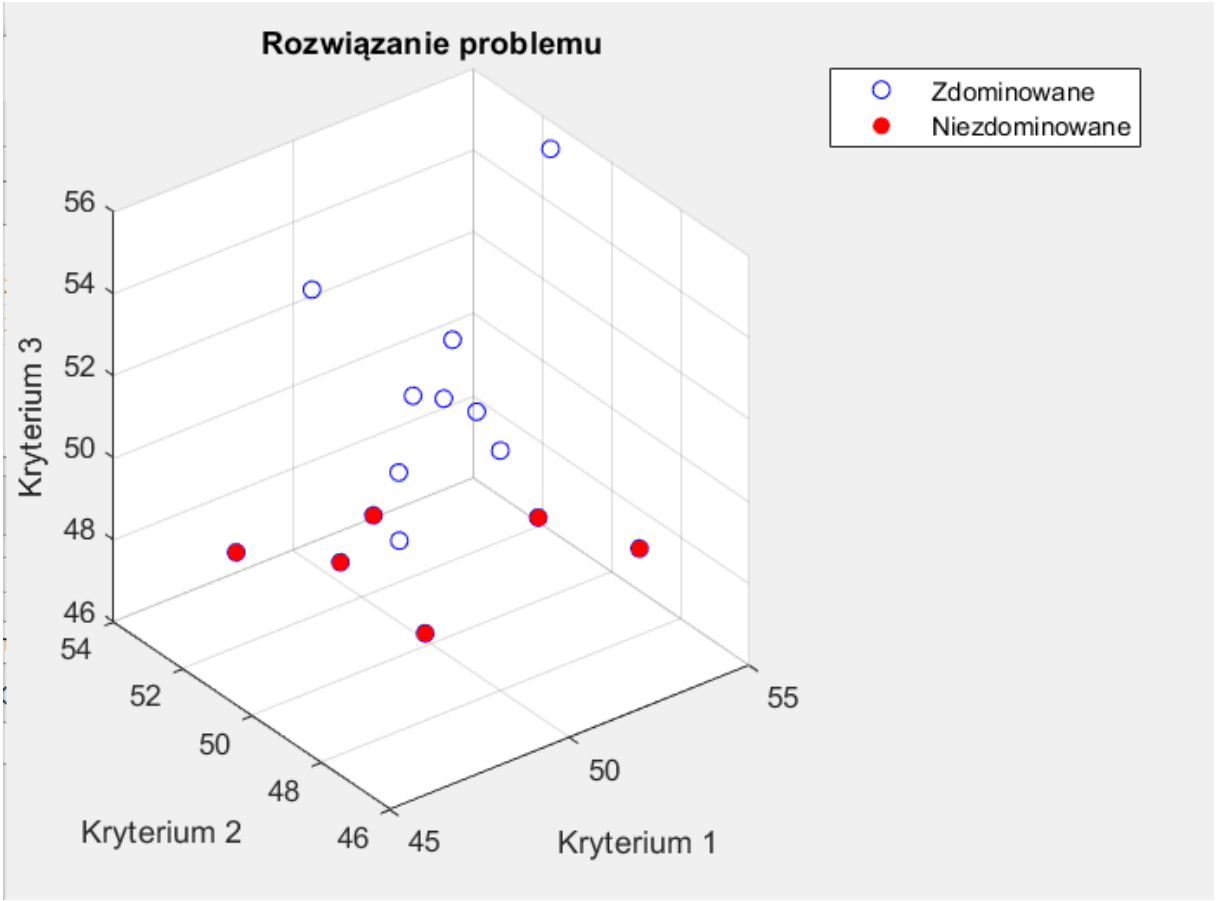

Benchmark: Execution Ti
×10-4
Execution Time (s)
0
0.5
1
1.5
2
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata

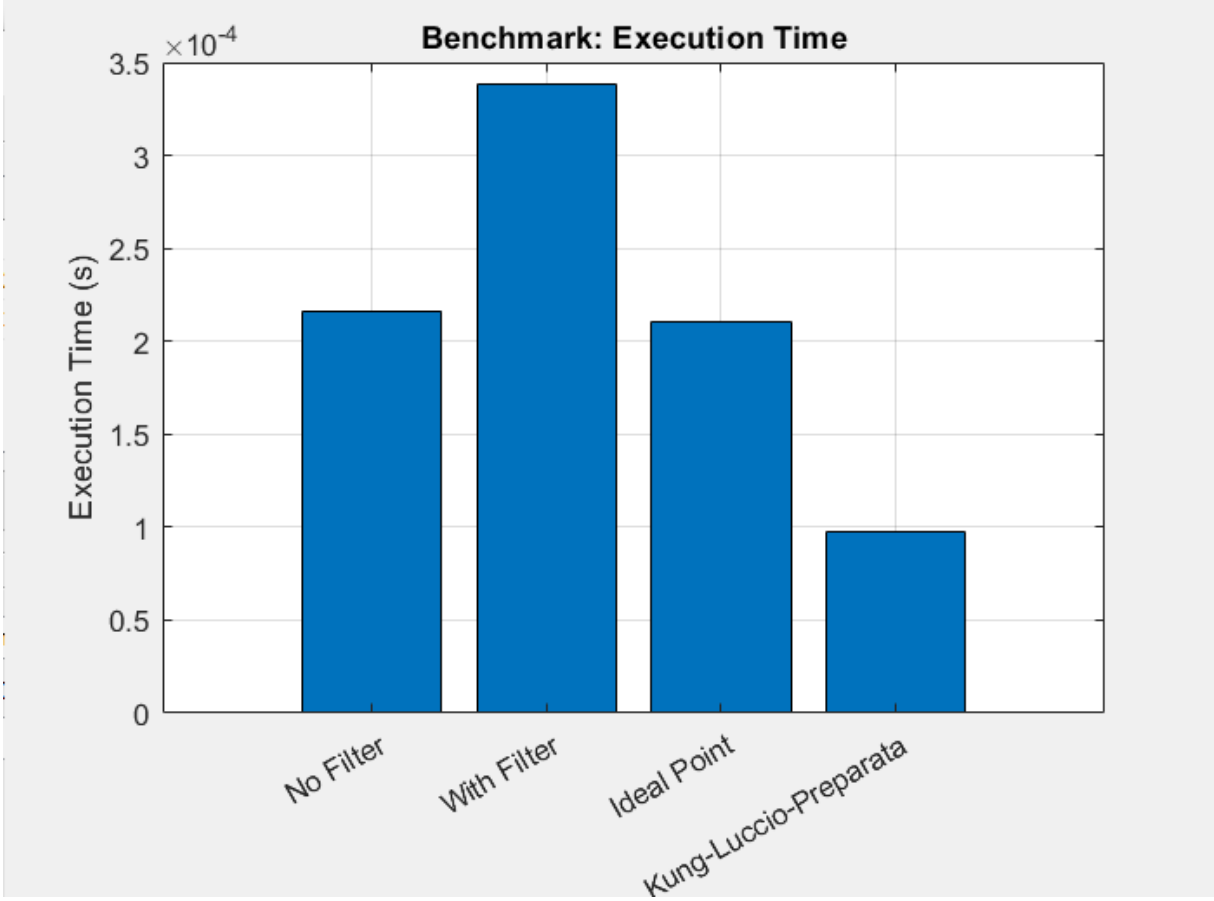

Benchmark: Point And Coordinates Comparisons
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata
700
600
500
400
300
200
100
0
Number of Points Comparisons
Number of Coordinates Comparisons

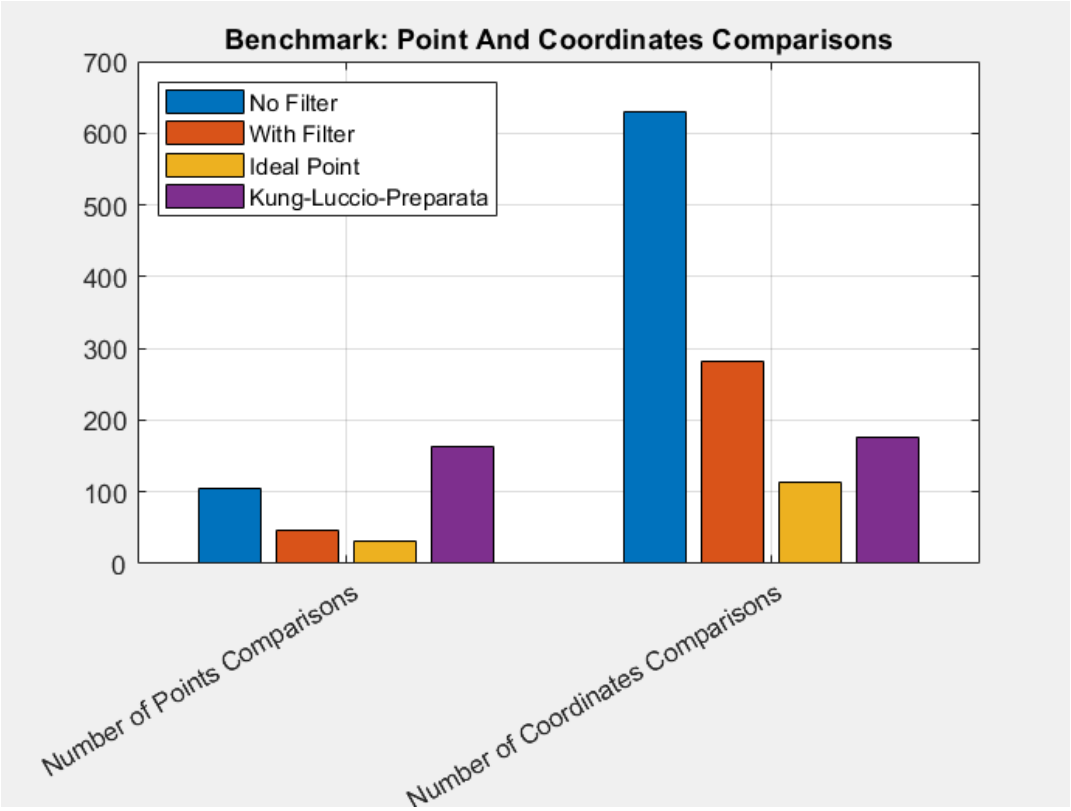

Distance to Ideal with Standard Deviation
Distance to Ideal
3
2.5
2
1.5
1
0.5
0
No Filter
With Filter
Kung-Luccio-Preparata

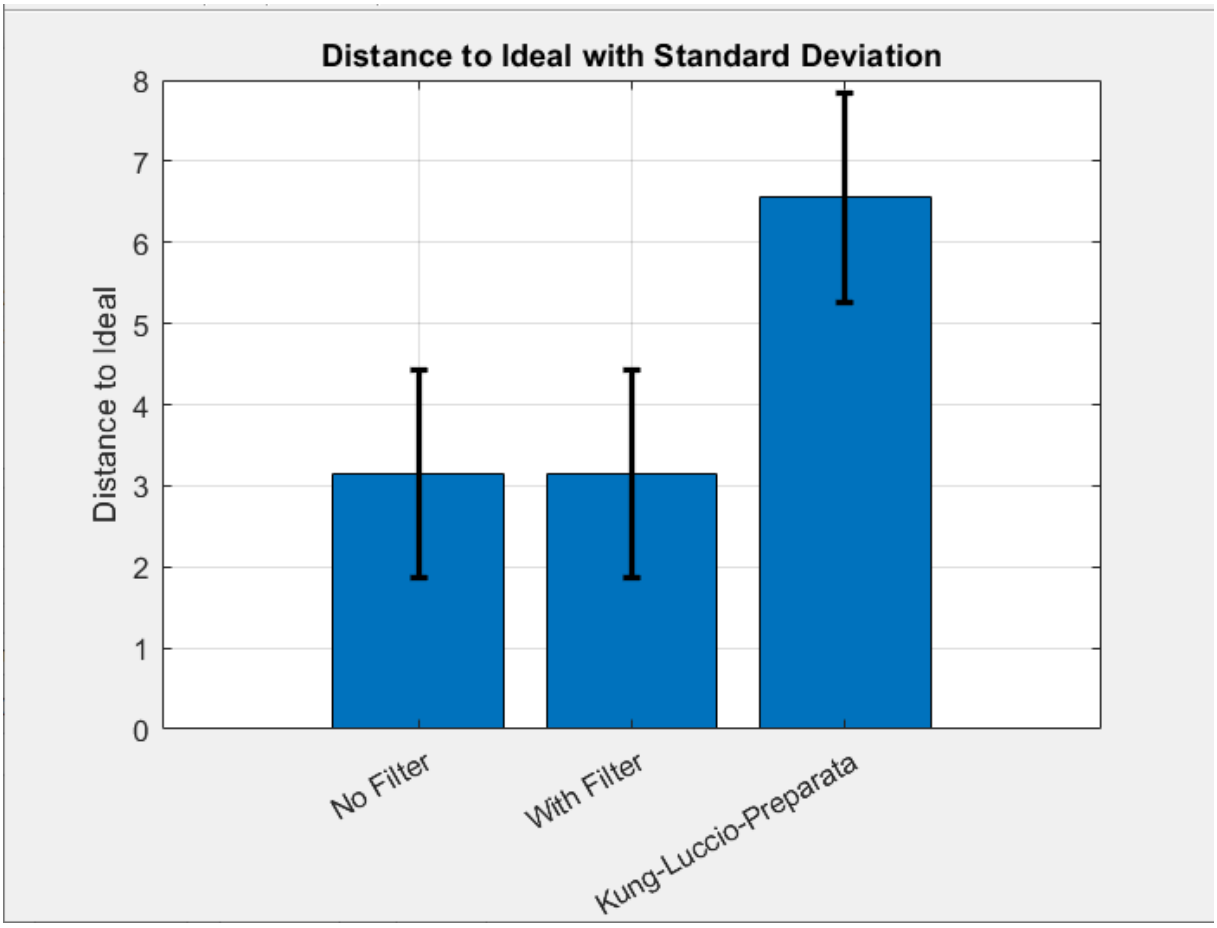
- Rozkład normalny, algorytm Kunga, Luccio & Preparaty



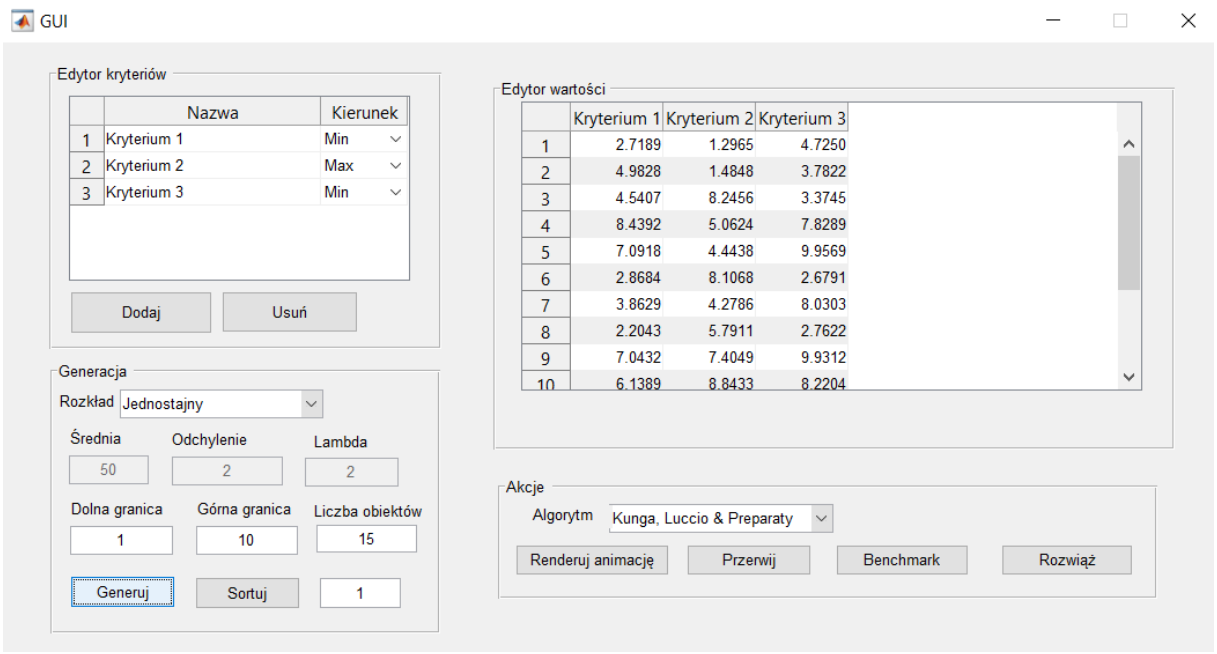


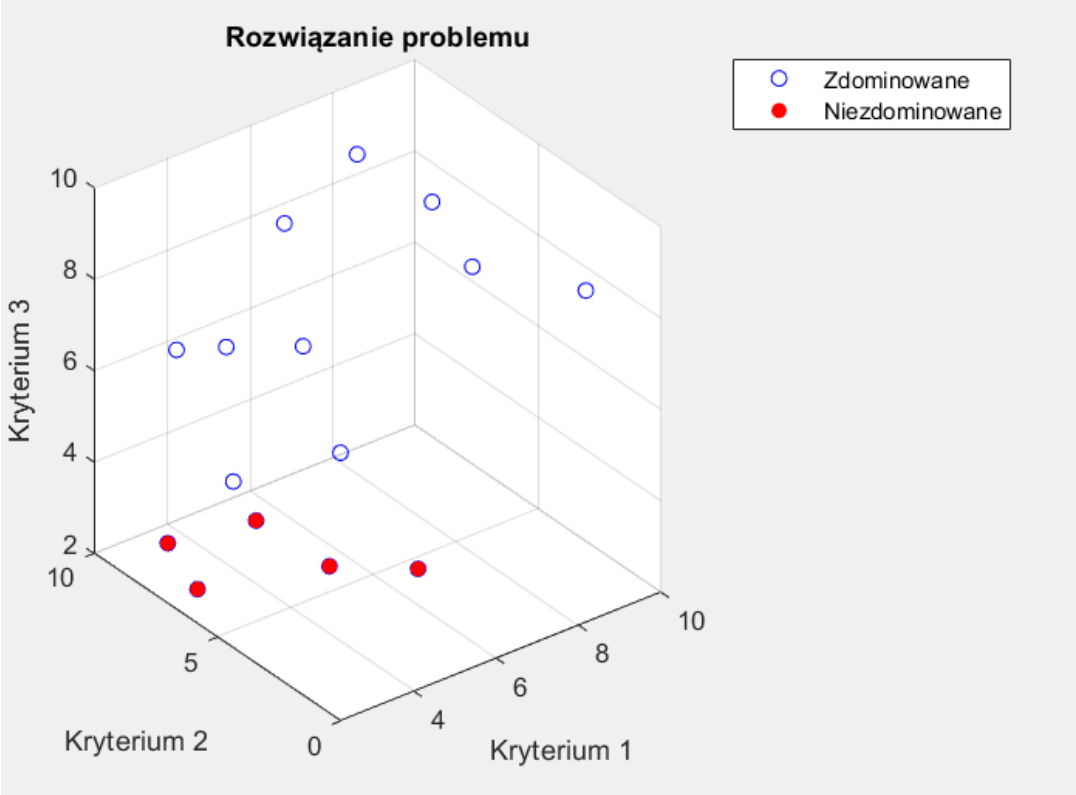

Benchmark: Execution Time
×10^-4
3.5
3
2.5
2
1.5
1
0.5
0
Execution Time (s)
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata

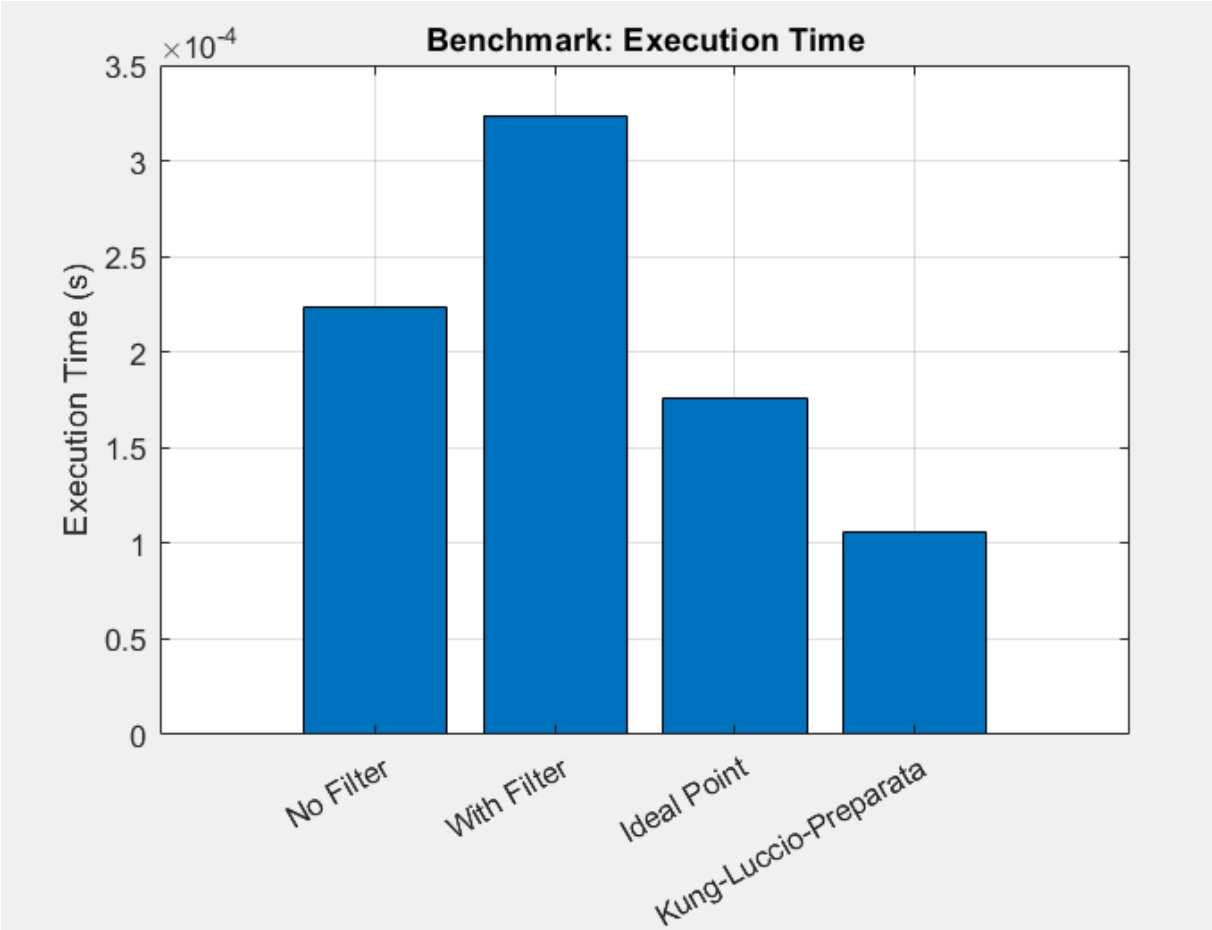

Benchmark: Point And Coordinates Comparisons
700
600
500
400
300
200
100
0
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata
Number of Points Comparisons
Number of Coordinates Comparisons

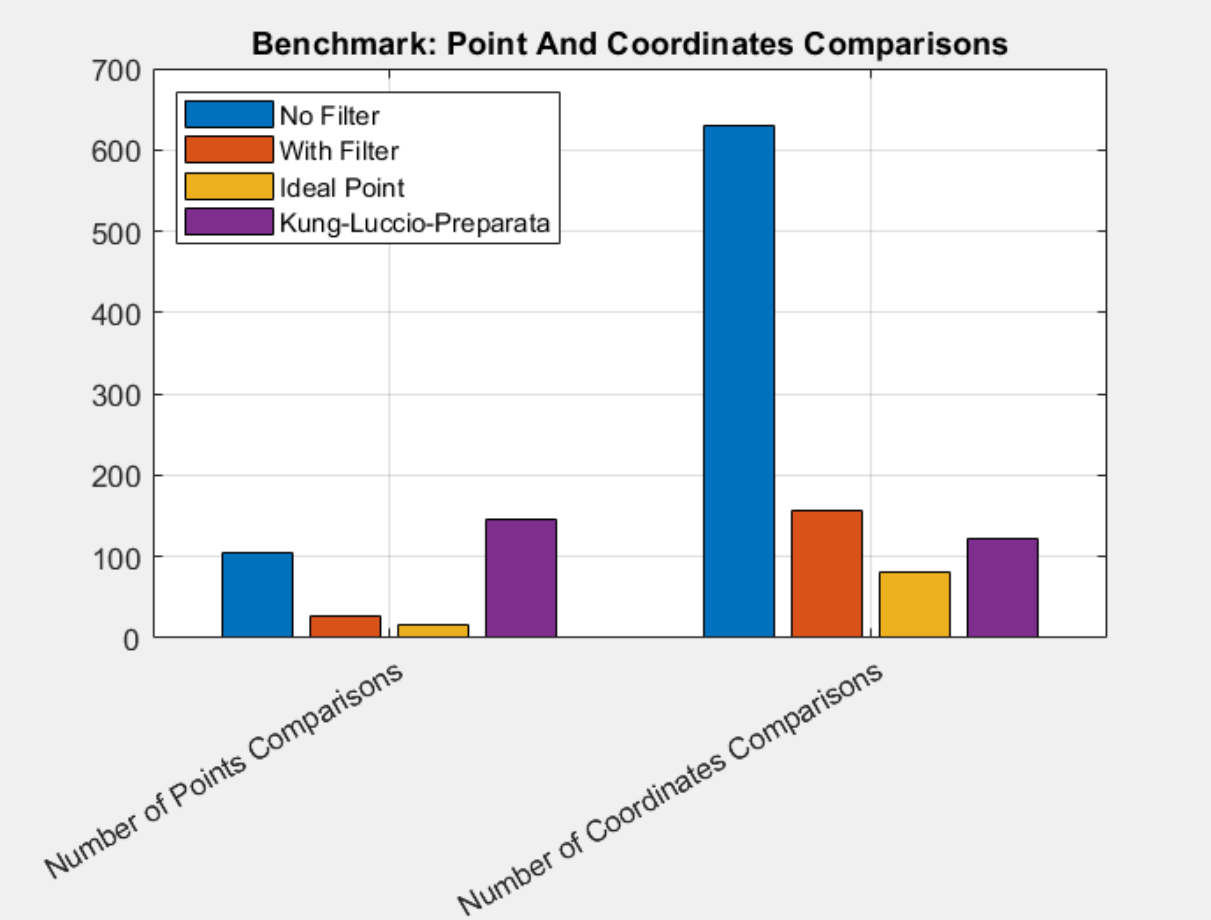


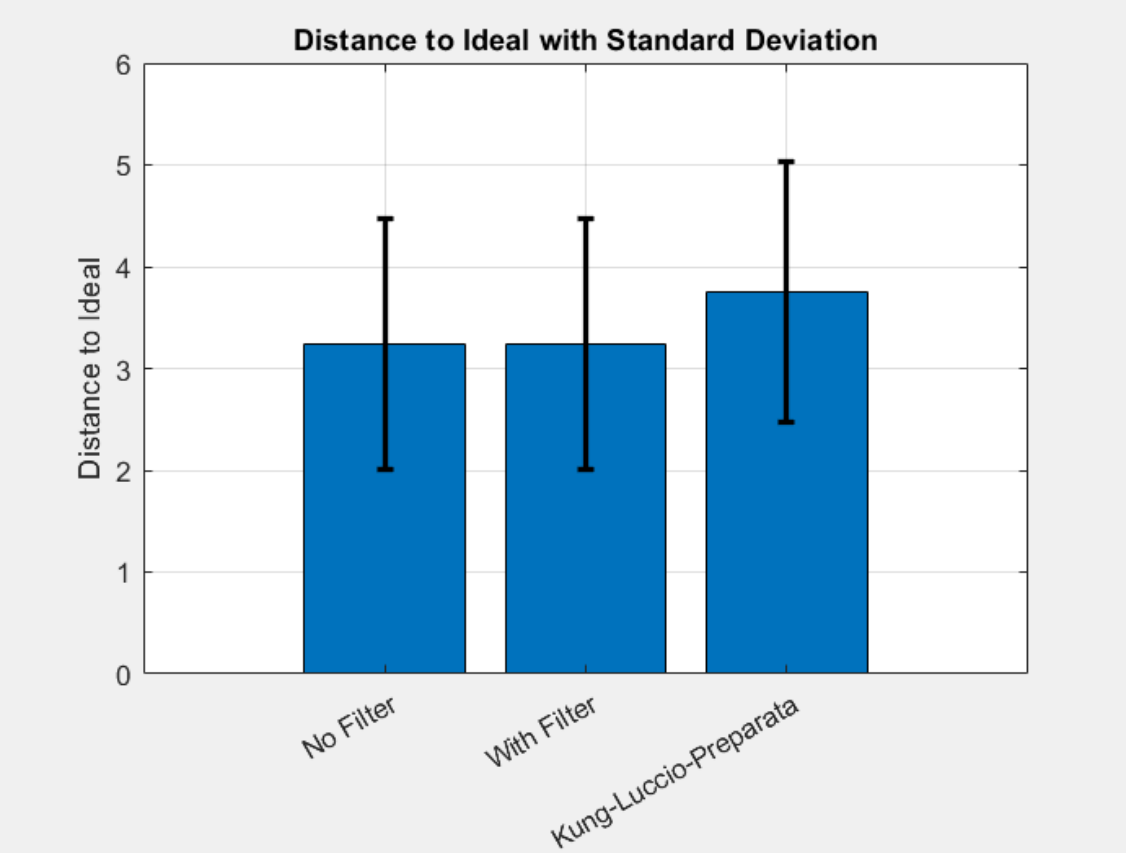
- Rozkład jednostajny, algorytm Kunga, Luccio & Preparaty



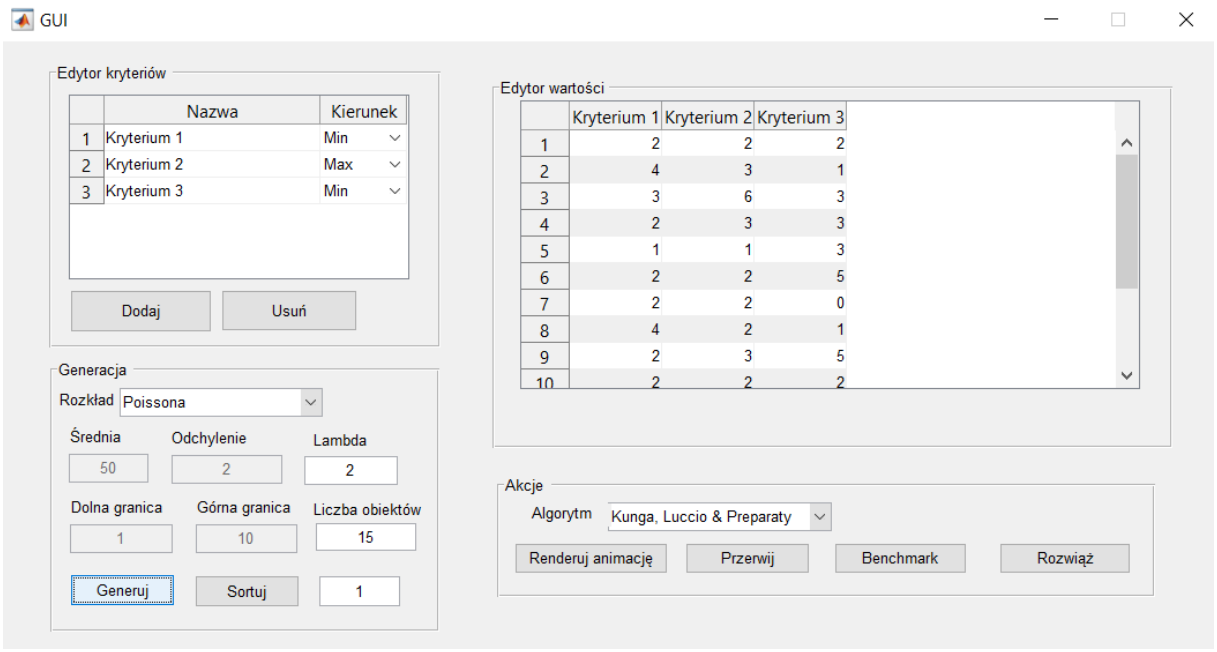

Rozwiązanie problemu
Zdominowane
Niezdominowane
Kryterium 3
10
8
6
4
2
10
5
0
Kryterium 2
4
6
8
10
Kryterium 1

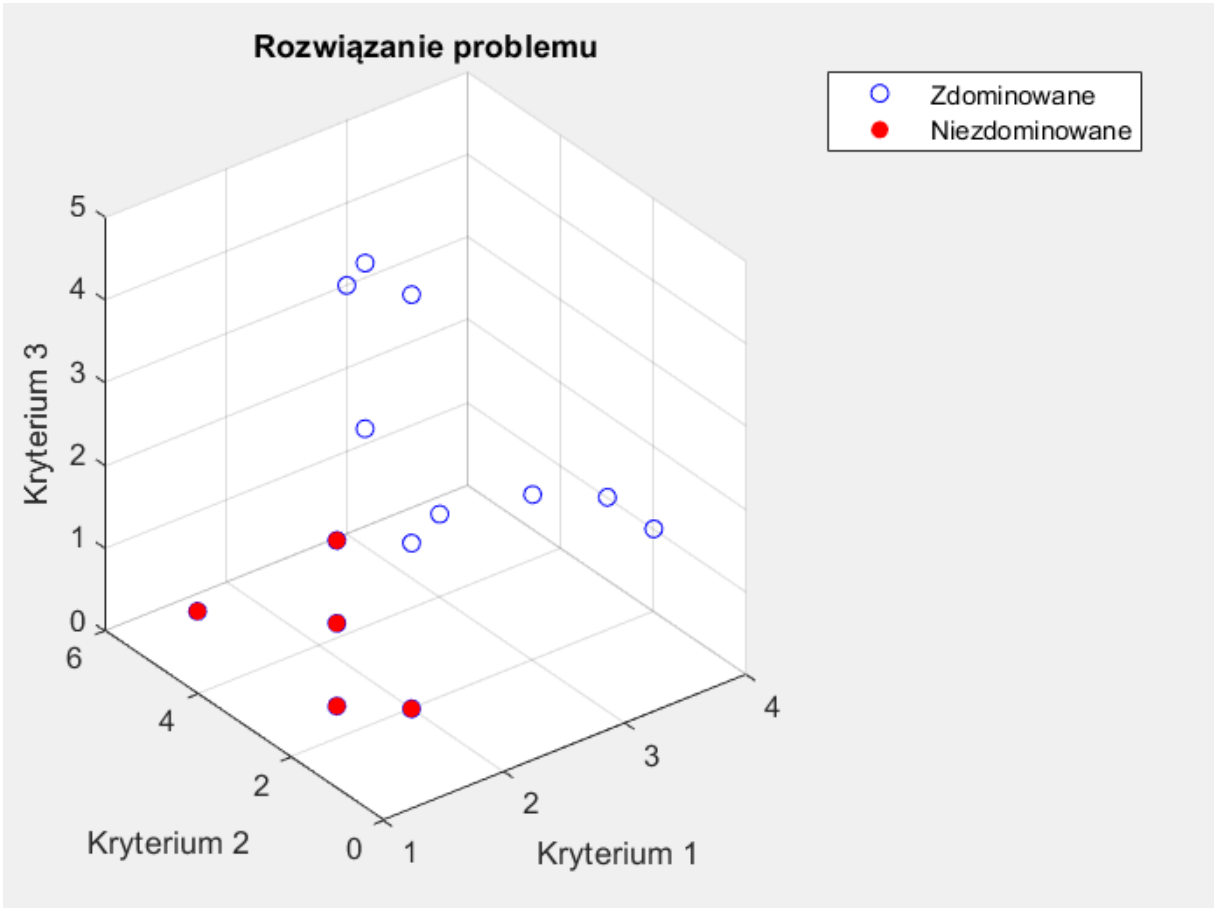

Benchmark: Execution Time
×10^-4
Execution Time (s)
3.5
3
2.5
2
1.5
1
0.5
0
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata

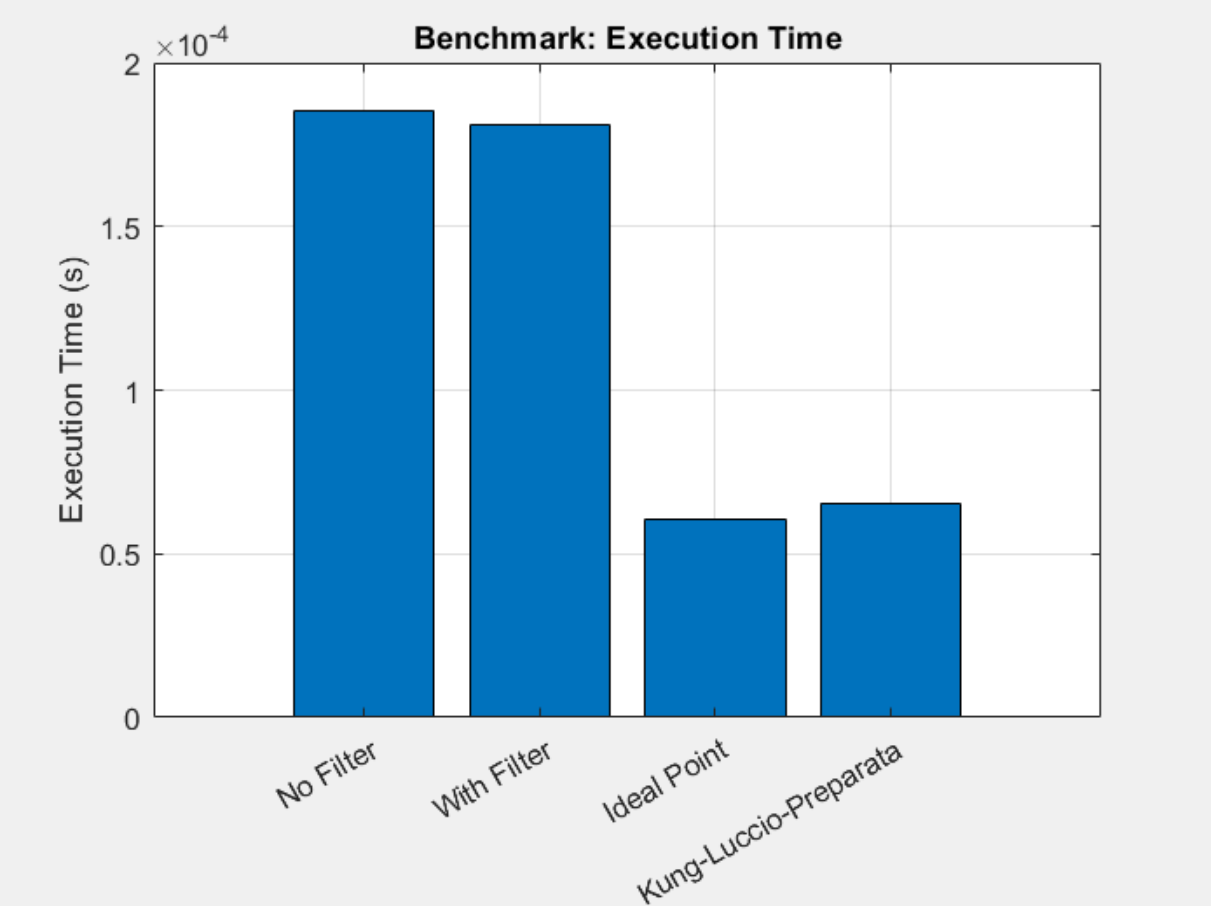

Benchmark: Point And Coordinates Comparisons
700
600
500
400
300
200
100
0
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata
Number of Points Comparisons
Number of Coordinates Comparisons

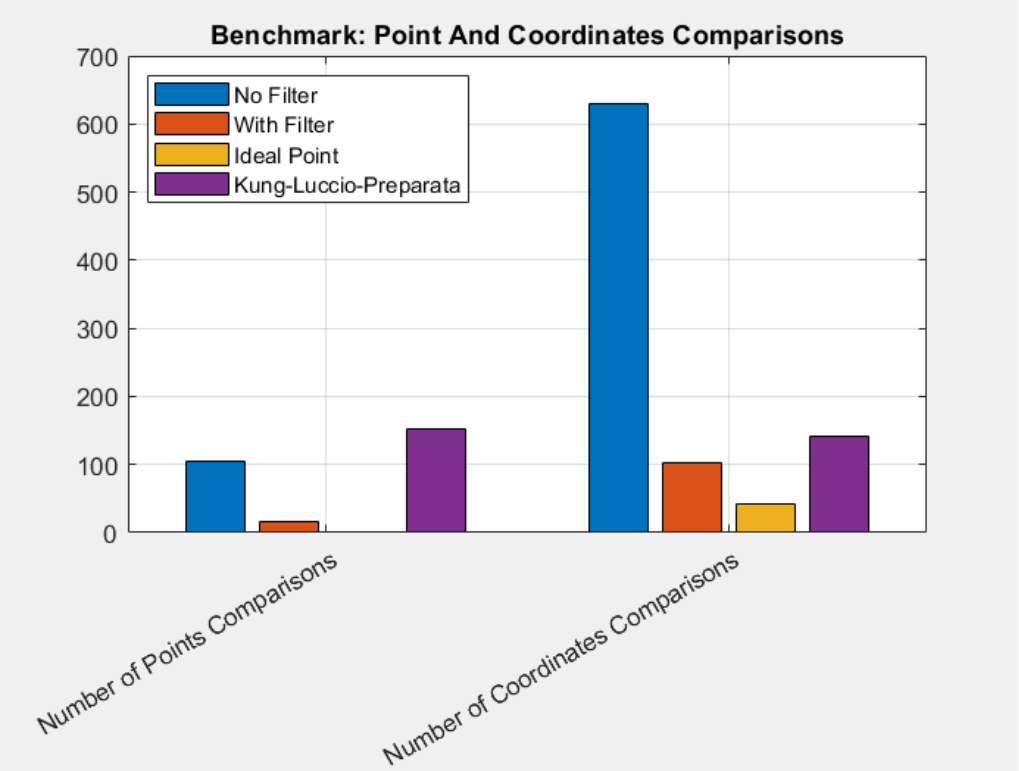

Distance to Ideal with Standard Deviation
Distance to Ideal
6
5
4
3
2
1
0
No Filter
With Filter
Kung-Luccio-Preparata

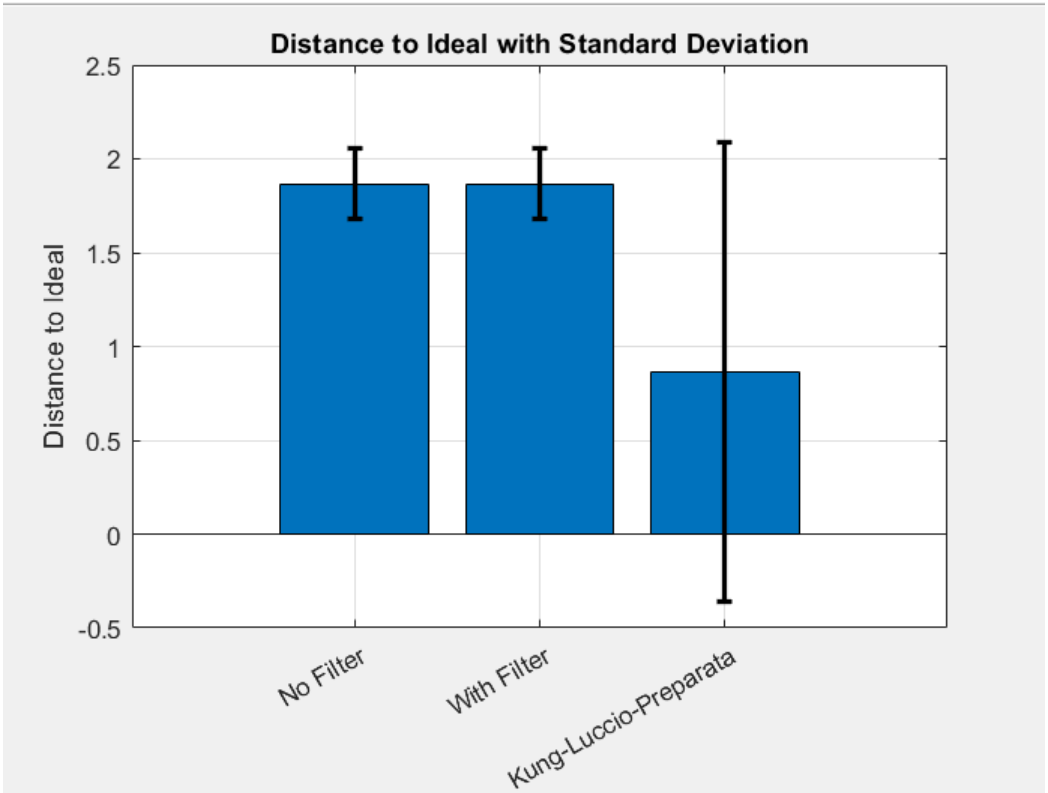
- Rozkład Poissona, algorytm Kunga, Luccio & Preparaty






Benchmark: Execution Time
×10^-4
2
1.5
1
0.5
0
Execution Time (s)
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata


Benchmark: Point And Coordinates Comparisons
700
600
500
400
300
200
100
0
No Filter
With Filter
Ideal Point
Kung-Luccio-Preparata
Number of Points Comparisons
Number of Coordinates Comparisons



Wnioski:

Analiza wyników uzyskanych podczas badań nad metodami wielokryterialnej optymalizacji dyskretnej pozwala na wyciągnięcie istotnych wniosków dotyczących efektywności oraz przydatności zastosowanych algorytmów w kontekście zdefiniowanych scenariuszy. Przeprowadzone eksperymenty obejmowały zbiory danych o różnych charakterystykach, reprezentujące rozkłady normalne, jednostajne i Poissona, co umożliwiło zbadanie skuteczności każdej z metod w warunkach odmiennych rozkładów danych.

Pierwszym analizowanym algorytmem był **algorytm naiwny bez filtracj**i, którego prostota konstrukcji pozwalała na szybkie wdrożenie i testowanie. Jednakże jego zastosowanie w praktyce ujawniło poważne ograniczenia wynikające z braku optymalizacji procesu obliczeniowego. Algorytm ten wymagał porównania każdego punktu w zbiorze z każdym innym, co w przypadku większych zbiorów danych prowadziło do znacznego wzrostu liczby obliczeń, a co za tym idzie, do wydłużenia czasu analizy. W efekcie, choć algorytm ten może być stosowany dla małych i prostych zbiorów danych, jego zastosowanie w bardziej złożonych przypadkach okazuje się niepraktyczne.

Z kolei **algorytm naiwny z filtracją** wprowadził istotne udoskonalenia w postaci mechanizmu redukcji liczby punktów wymagających porównania. Dzięki zastosowaniu tej techniki znacząco zmniejszono ilość obliczeń, co przełożyło się na skrócenie czasu realizacji analizy. Szczególnie wyraźnie korzyści z tego podejścia zaobserwowano w przypadku danych o rozkładzie jednostajnym, gdzie liczba punktów była największa. Algorytm naiwny z filtracją pozwalał na bardziej efektywne przetwarzanie danych, co czyniło go bardziej

uniwersalnym i przydatnym narzędziem w analizie wielokryterialnej niż jego wersja bez filtracji.

Porównanie algorytmów naiwnych wykazało znaczącą różnicę w efektywności obliczeń. Mechanizm filtracji w drugim algorytmie znacząco zmniejszył liczbę porównań punktów, co przełożyło się na szybsze wyniki. To doświadczenie potwierdziło praktyczne znaczenie optymalizacji przy analizie dużych zbiorów danych.

Trzecim rozważanym podejściem był **algorytm punktu idealnego**, który opierał się na koncepcji wyznaczania hipotetycznego punktu reprezentującego najbardziej optymalne rozwiązanie w przestrzeni wielokryterialnej. Rozwiązanie to okazało się szczególnie skuteczne w przypadkach, gdy dane wykazywały duże zróżnicowanie, co miało miejsce przy rozkładach normalnych i Poissona. Algorytm ten charakteryzował się relatywnie stabilnym czasem obliczeń, jednocześnie dostarczając wyników, które były zgodne z oczekiwaniami praktycznymi w zakresie wyboru najbardziej optymalnych rozwiązań. Z tego powodu algorytm punkt idealny może być szczególnie przydatny w sytuacjach, gdzie jakość wyników jest istotniejsza niż czas ich uzyskania.

Największe wyzwania oraz duże ilości błędów pojawiły się w trakcie implementacji złożonego algorytmu **Kung-Luccio-Preparata**, którego zaawansowana struktura teoretyczna wiązała się z trudnym procesem integracji z pozostałymi komponentami aplikacji. Pomimo początkowych trudności technicznych algorytm ten wykazał potencjał w analizie przypadków wymagających bardziej zaawansowanego podejścia do selekcji optymalnych rozwiązań.

Oprócz analizy algorytmów, istotnym elementem projektu było stworzenie intuicyjnego i funkcjonalnego graficznego interfejsu użytkownika (GUI). Interfejs ten umożliwił użytkownikom zarządzanie kryteriami, wizualizację wyników oraz generowanie danych wejściowych w sposób przejrzysty i przystępny, nawet dla osób bez zaawansowanej wiedzy technicznej. Rozwiązanie to znaczną miarą przyczyniło się do sukcesu projektu, podnosząc wartość aplikacji jako narzędzia praktycznego.

Pomimo napotkanych trudności technicznych, stworzona aplikacja z powodzeniem spełniła założenia projektowe i stanowi solidną podstawę do dalszego rozwoju narzędzi wspierających wielokryterialną analizę dyskretną.

Implementacja projektu pokazała, jak ważne jest gruntowne testowanie każdego elementu kodu. Dzięki systematycznemu testowaniu udało się szybko zidentyfikować problematyczne miejsca, takie jak niezgodność liczby zwracanych wyników przez algorytmy lub błędy związane z niewłaściwymi danymi wejściowymi. Pomogło to zwłaszcza podczas tworzenia algorytmu Luccio-Preparata, bez porządkowania kodu oraz systematycznego testowania ukończenie go nie byłoby możliwe

Realizacja projektu była cennym doświadczeniem z zakresu pracy zespołowej. Podział zadań pomiędzy członków grupy pozwolił na efektywniejsze wykorzystanie czasu, a wzajemna weryfikacja pracy oraz pomoc przy zadaniach innych członków zespołu pozwoliła na płynną współpracę i zapewniła wyższą jakość końcowego produktu.

Podsumowując, wyniki przeprowadzonych badań dowodzą, że wybór algorytmu optymalizacyjnego powinien być ściśle uzależniony od charakterystyki danych wejściowych oraz wymagań użytkownika co do czasu obliczeń i precyzji wyników. Projekt dostarczył cennych doświadczeń w zakresie implementacji, testowania i optymalizacji algorytmów oraz integracji ich w przyjaznym dla użytkownika środowisku.