

Górka Bartosz Zimniak Kajetan
127228 127229

Algorytmy ewolucyjne i metaheurystyki

Sprawozdanie 3

1. Opis etapu projektu

Celem kolejnego etapu projektu była implementacja rozszerzeń dla algorytmu *Steepest*. Były to zastosowanie ruchów kandydackich oraz cache.

W projekcie wykorzystano algorytm *Steepest* z bazowym przydziałem punktów do grup powstającym dzięki wykorzystaniu algorytmu naiwnego.

Jako naiwny algorytm wykorzystano algorytm przydziału punktu do grupy (bez żalu). Liczba grup pozostała bez zmian na poziomie 20.

Podobnie jak w poprzednich etapach, funkcja celu została zdefiniowana jako *minimalizacja średniej odległości wszystkich par obiektów umieszczonych w ramach tej samej grupy*.

W rozdziale 2 zaprezentowano pseudokody przygotowanych rozszerzeń algorytmu, natomiast w rozdziale 3 wyniki działania dla 100 iteracji. Ostatni rozdział dotyczy wizualizacji najlepszych uzyskanych rozwiązań.

2. Pseudokody rozszerzeń przeszukiwania lokalnego *Steepest*

2.1. Ruchy kandydackie

```
Dla każdej iteracji algorytmu Steepest wykonaj {  
    Dla grupy g1 z listy grup punktów {  
        Dla każdego punktu p1 w grupie g1 {  
            Wyznacz najbliższy punkt p2 różny od p1 należący do grupy g2 innej niż g1  
            Do listy możliwych ruchów dodaj przesunięcie punktu p1 z grupy g1 do grupy g2  
        }  
    }  
}
```

Na podstawie wyznaczonej listy możliwych przesunięć, algorytm *Steepest* wybiera to, które najbardziej minimalizuje wartość funkcji celu.

2.2. Cache

```
Dla każdego ruchu r z listy możliwych ruchów {  
    Jeżeli ruch r nie znajduje się w cache {  
        Oblicz deltę zmiany funkcji celu dla ruchu r
```

```

        Zapisz deltę i ruch w pamięci cache
    }

    Z pamięci cache użyj oszacowania funkcji celu
    Jeżeli delta poprawia wartość funkcji celu {
        Zapamiętaj ruch jako dotychczas najlepszy
        Dokonaj aktualizacji aktualnej wartości funkcji celu
    }
}

Po wybraniu najlepszego z ruchów i jego wykonaniu {
    Usuń z pamięci cache przesunięcia dla grupy początkowej i docelowej, które zostały
zmienione po wykonaniu ruchu

    Dla każdego z zapamiętanych ruchów dokonaj modyfikacji wartości
    poprzez zmianę liczby łuków i sumy ich długości.
}

```

3. Wyniki eksperymentów obliczeniowych

W tabeli 1 zaprezentowano wyniki eksperymentów obliczeniowych. Dokonano 100 powtórzeń obliczeń. Za każdym razem algorytmy zostały uruchomione dla rozwiązań startowych zbudowanych z wykorzystaniem algorytmu naiwnego.

Tabela 1. Wyniki eksperymentów obliczeniowych dla 100 iteracji

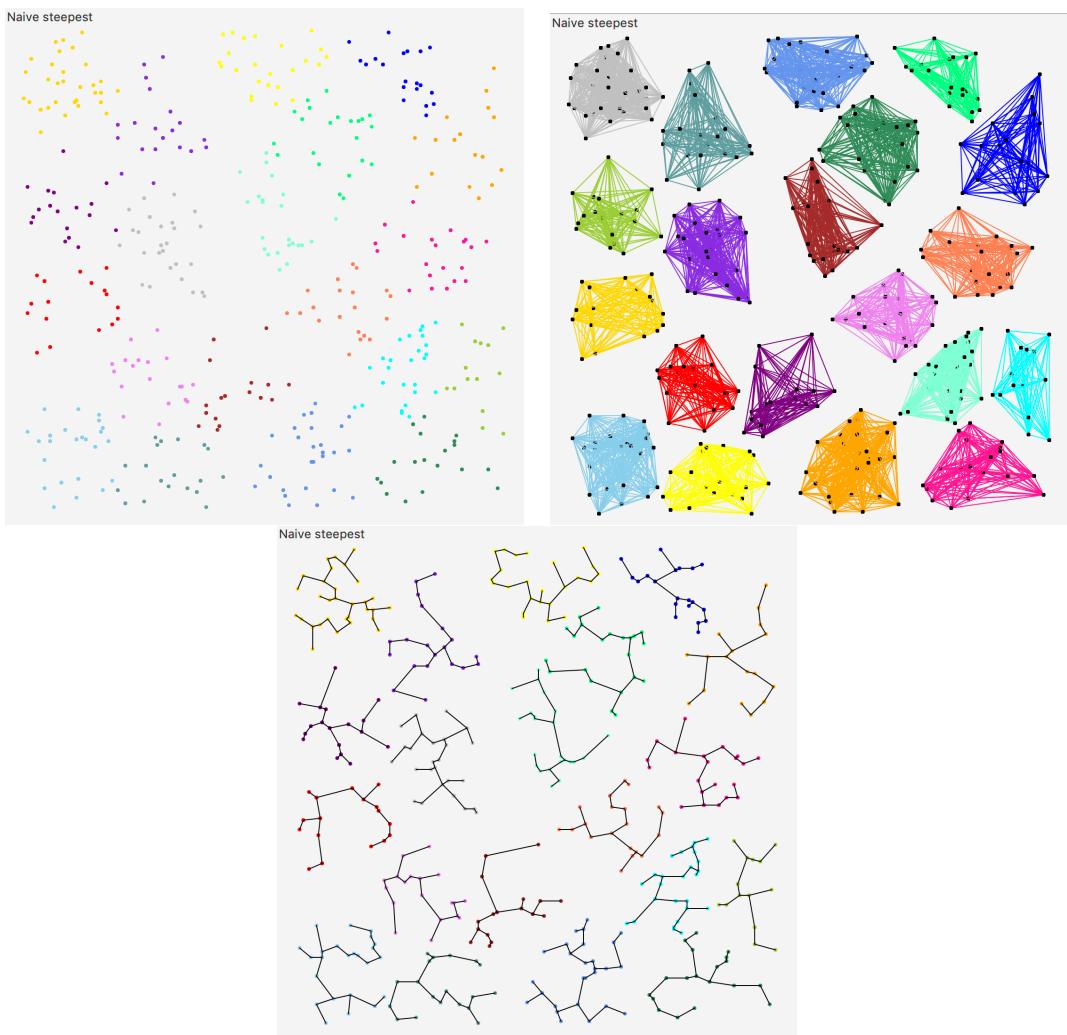
Cecha	Naive Steepest	Naive Steepest + Cache	Naive Steepest + Candidates	Naive Steepest with Cache + Candidates
Wartość minimalna funkcji celu	26.39	26.69	26.65	27.15
Wartość maksymalna funkcji celu	28.67	32.77	31.25	34.09
Wartość średnia funkcji celu	27.27	28.68	27.82	29.56
Wartość minimalna czasu obliczeń [sec]	0.81	6.57	0.55	0.13
Wartość maksymalna czasu obliczeń [sec]	2.32	40.16	2.05	2.52
Wartość średnia czasu obliczeń [sec]	1.40	23.49	0.96	0.68

Zgodnie z oczekiwaniami, zastosowanie mechanizmu łuków kandydackich pozwoliło przyspieszyć działanie algorytmu. Za zmianę czasu wykonania algorytmu musimy jednakże liczyć się z nieznacznym pogorszeniem wartości funkcji celu.

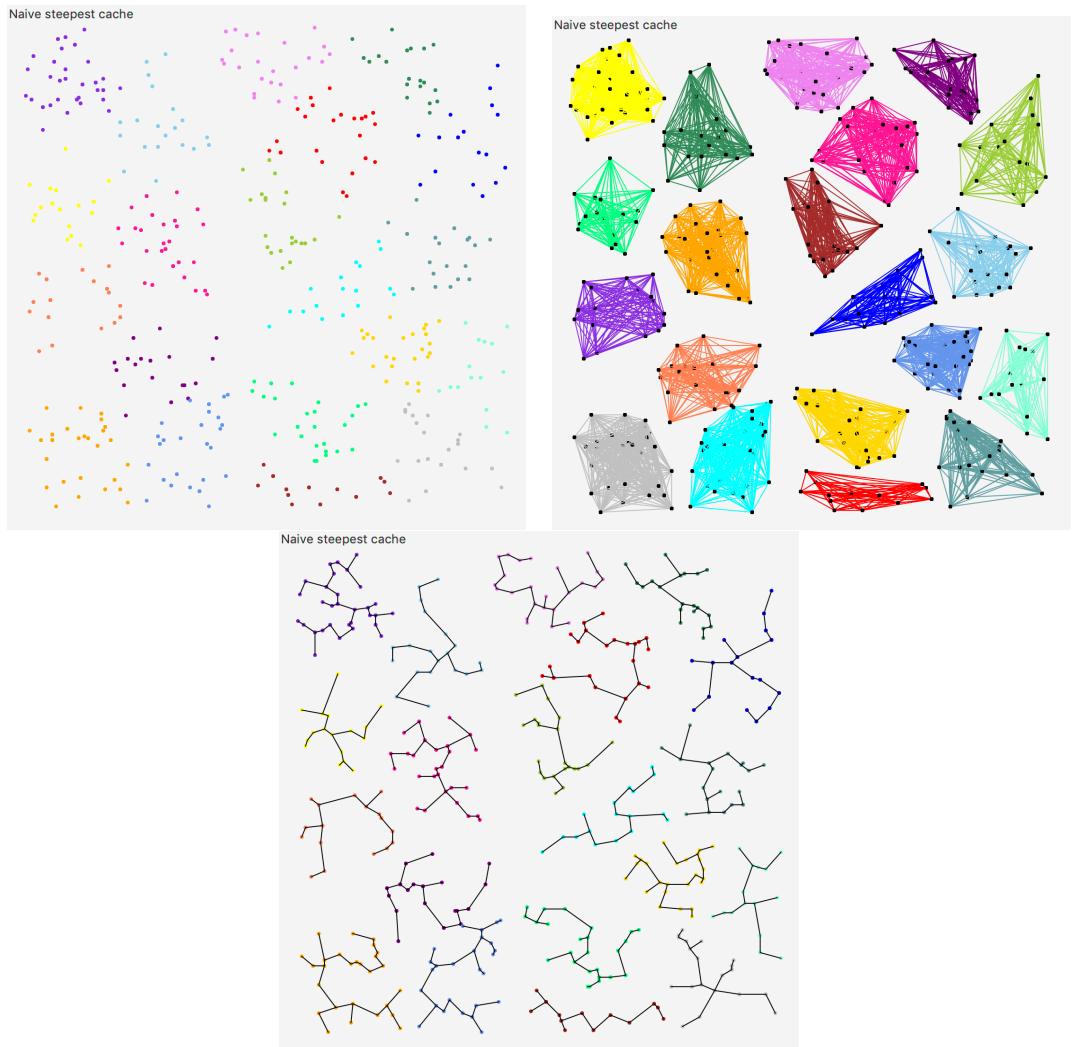
W przypadku wykorzystania mechanizmu pamięci (cache) czas realizacji jest wielokrotnie dłuższy, niż ten bez aktywnej pamięci. Jest to spowodowane aktualną funkcją celu i koniecznością aktualizacji każdego z przesunięcia jakie zostało zapamiętane. Algorytm *Steepest* z liczeniem funkcji celu w formie delty (bez budowania całego rozwiązania, liczenie zmian) jest znacznie szybszy niż wielokrotne aktualizacje pamięci.

4. Wizualizacja najlepszych rozwiązań

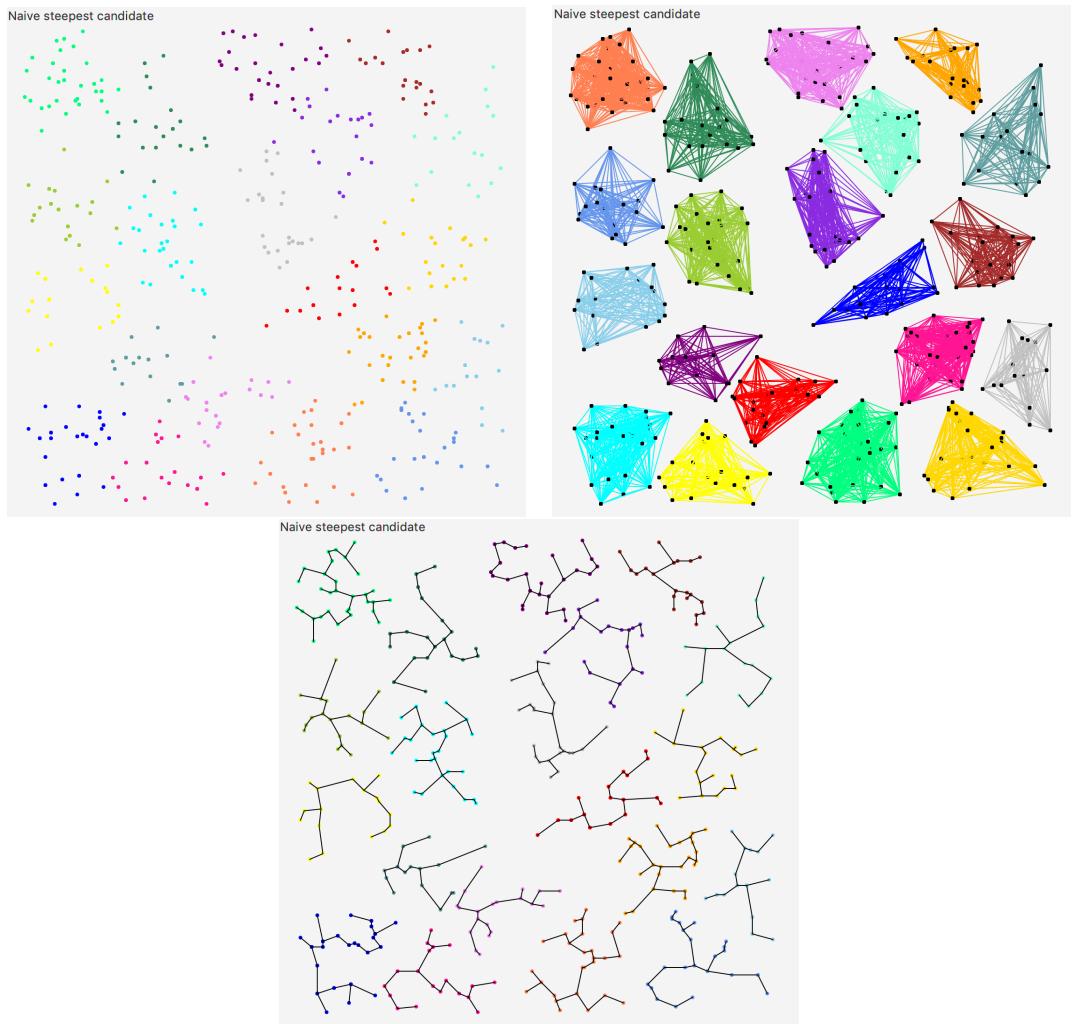
Podobnie jak w poprzednim etapie projektu, również w tym zastosowano wizualizację najlepszych rozwiązań na trzy sposoby. Pierwszy z nich to zaprezentowanie samych grup punktów, bez jakichkolwiek powiązań. Drugim sposobem jest prezentacja zgodna z funkcją celu, czyli zaprezentowanie powiązań pomiędzy punktami w ramach grupy. Ostatni sposób wykorzystuje minimalne drzewo rozpinające, które w przejrzysty sposób prezentuje przydział punktów do grup.



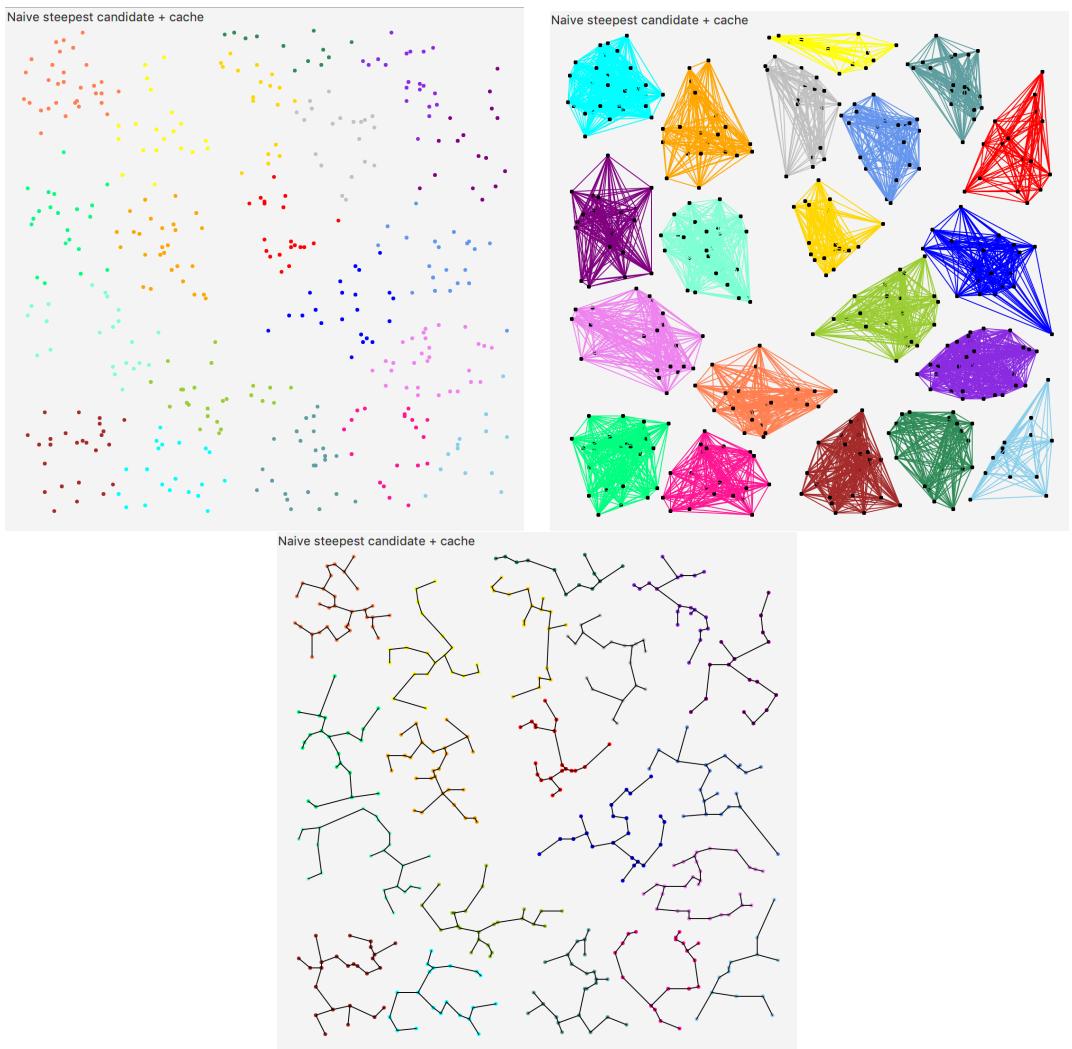
Rysunek 1. Naive Steepest Local Search



Rysunek 2. Naive Steepest Local Search + Cache



Rysunek 3. Naive Steepest Local Search + Candidates



Rysunek 4. Naive Steepest Local Search + Cache + Candidates

Górka Bartosz Zimniak Kajetan
127228 127229

Algorytmy ewolucyjne i metaheurystyki

Sprawozdanie 2

1. Opis etapu projektu

Celem kolejnego etapu projektu było wykorzystanie przygotowanego podczas ćwiczenia 1 środowiska testowego. W projekcie należało wykorzystać naiwny algorytm przydziału punktów do grup oraz algorytm losowy, które stanowiły podstawę tworzenia rozwiązania startowego dla algorytmów lokalnego przeszukiwania przygotowanych podczas tego etapu projektu.

Jako naiwny algorytm wykorzystano algorytm przydziału punktu do grupy (bez żalu), natomiast algorytm losowy został oparty o całkowicie losowy przydział każdego punktu do jednej z 20 grup.

W ramach projektu przygotowano dwa algorytmy przeszukiwania lokalnego. Pierwszym z nich jest algorytm *Greedy Local Search*, natomiast drugim *Steepest Local Search*.

Podobnie jak w poprzednim etapie, funkcja celu została zdefiniowana jako *minimalizacja średniej odległości wszystkich par obiektów umieszczonych w ramach tej samej grupy*.

W rozdziale 2 zaprezentowano pseudokody przygotowanych algorytmów, natomiast w rozdziale 3 wyniki działania algorytmów dla 100 iteracji. Ostatni rozdział dotyczy wizualizacji najlepszych uzyskanych rozwiązań.

2. Pseudokody algorytmów przeszukiwania lokalnego

2.1. Generowanie listy ruchów

W algorytmach Greedy oraz Steepest Local Search wykorzystano metodę tworzącą listę możliwych ruchów dla aktualnego przydziału punktów do grup. Aby ułatwić zrozumienie algorytmów, wydzielono ją z pseudokodów i zaprezentowano osobno:

```
Zainicjalizuj listę ruchów jako pustą listę
Dla każdego punktu p z listy możliwych punktów {
    Dla każdej grupy g z listy grup {
        Jeżeli punkt p nie należy do grupy g {
            Do listy możliwych ruchów dodaj potencjalne przesunięcie punktu p
            z obecnej grupy do grupy g
        }
    }
}
Jako wynik metody zwróć listę ruchów
```

2.2. Greedy Local Search

```
Wykonuj dopóki przydział punktów do grup ulega zmianie {  
    Przygotuj listę możliwych ruchów dla obecnego sąsiedztwa  
    Dokonaj losowego posortowania listy ruchów  
  
    Dla każdego ruchu r z listy możliwych ruchów {  
        Oblicz deltę zmiany funkcji celu dla ruchu r  
        Jeżeli delta poprawia wartość funkcji celu {  
            Oznacz że dokonano zmiany przydziału punktów (kolejna iteracja możliwa)  
            Zaaplikuj ruch r  
            Dokonaj aktualizacji aktualnej wartości funkcji celu  
            Przerwij pętlę sprawdzania ruchów  
        }  
    }  
}
```

2.3. Steepest Local Search

```
Wykonuj dopóki przydział punktów do grup ulega zmianie {  
    Przygotuj listę możliwych ruchów dla obecnego sąsiedztwa  
  
    Dla każdego ruchu r z listy możliwych ruchów {  
        Oblicz deltę zmiany funkcji celu dla ruchu r  
        Jeżeli delta poprawia wartość funkcji celu {  
            Zapamiętaj ruch jako dotychczas najlepszy  
            Dokonaj aktualizacji aktualnej wartości funkcji celu  
        }  
    }  
  
    Jeżeli dokonano zapamiętania ruchu (zmieniono funkcję celu) {  
        Zaaplikuj najlepszy ruch (najbardziej poprawiający funkcję celu)  
        Oznacz że dokonano zmiany przydziału punktów (aby wykonać kolejną iterację)  
    }  
}
```

3. Wyniki eksperymentów obliczeniowych

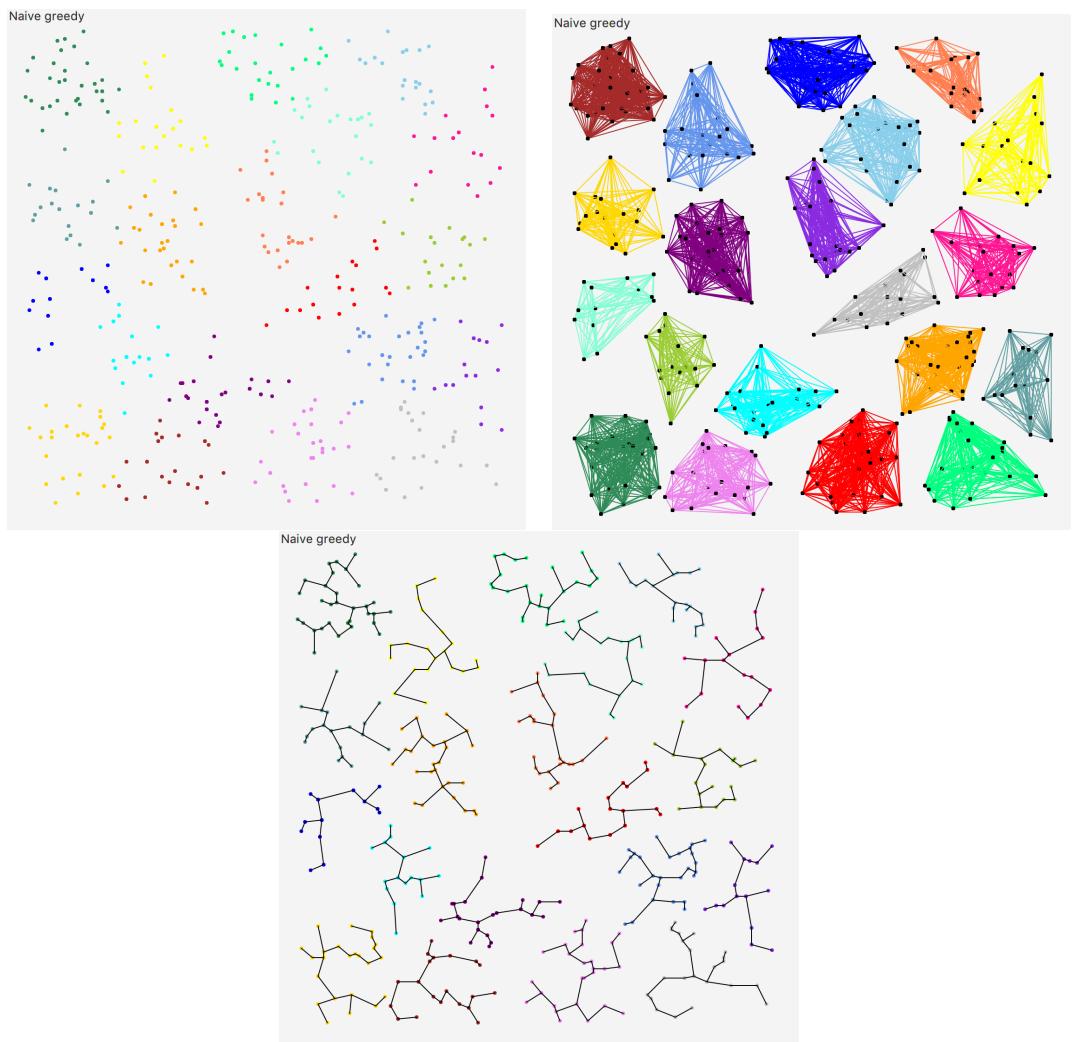
W tabeli 1 zaprezentowano wyniki eksperymentów obliczeniowych. Dokonano 100 powtórzeń obliczeń. Za każdym razem algorytmy zostały uruchomione dla dwóch rozwiązań startowych. Pierwszy z nich to wynik działania algorytmu naïwnego przygotowanego w poprzednim ćwiczeniu, a drugi to przydział przygotowany przez algorytm losowego przydziału punktów do grup.

Tabela 1. Wyniki eksperymentów obliczeniowych dla 100 iteracji

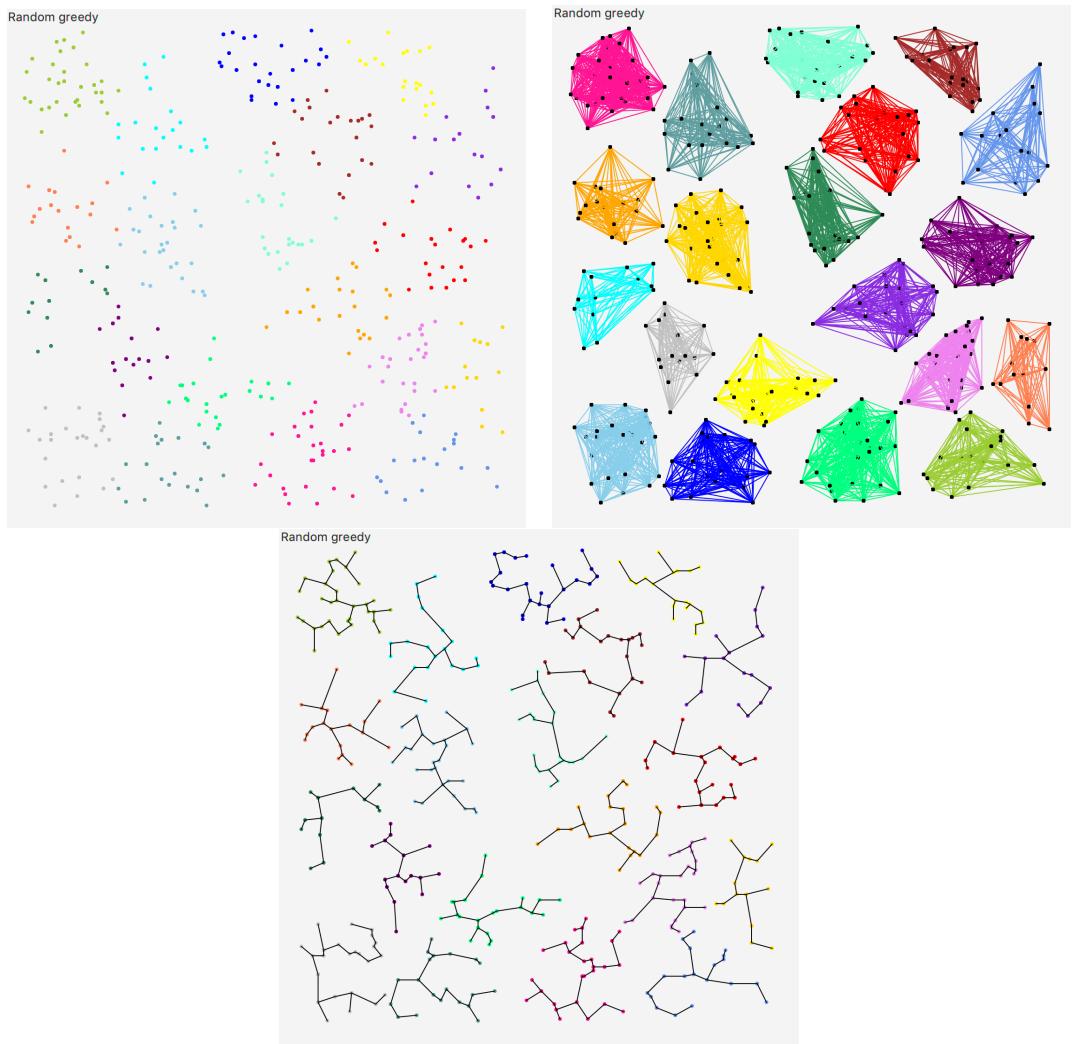
Cecha	Naive Greedy LS	Random Greedy LS	Naive Steepest LS	Random Steepest LS
Wartość minimalna funkcji celu	26.39	26.37	26.39	26.39
Wartość maksymalna funkcji celu	29.07	27.99	28.82	28.86
Wartość średnia funkcji celu	27.00	26.95	27.27	27.12
Wartość minimalna czasu obliczeń [sec]	0.11	0.34	0.83	4.50
Wartość maksymalna czasu obliczeń [sec]	0.60	0.71	3.28	7.51
Wartość średnia czasu obliczeń [sec]	0.19	0.41	1.48	5.52

4. Wizualizacja najlepszych rozwiązań

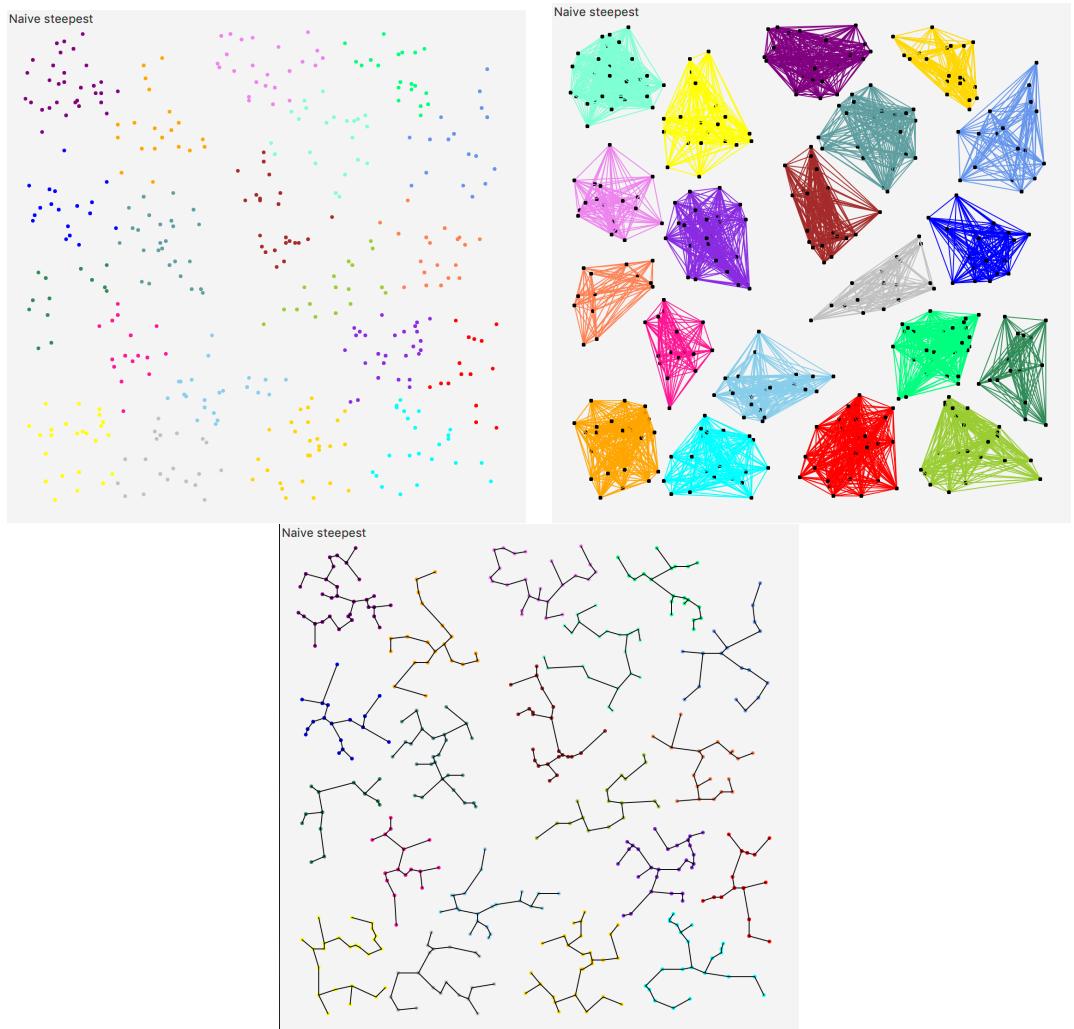
W wizualizacji najlepszych rozwiązań wykorzystano trzy sposoby prezentacji rozwiązań. Pierwszy z nich to zaprezentowanie samych grup punktów, bez jakichkolwiek powiązań. Drugim sposobem jest prezentacja zgodna z funkcją celu, czyli zaprezentowanie powiązań pomiędzy punktami w ramach grupy. Ostatni sposób wykorzystuje minimalne drzewo rozpinające, które w przejrzysty sposób prezentuje przydział punktów do grup.



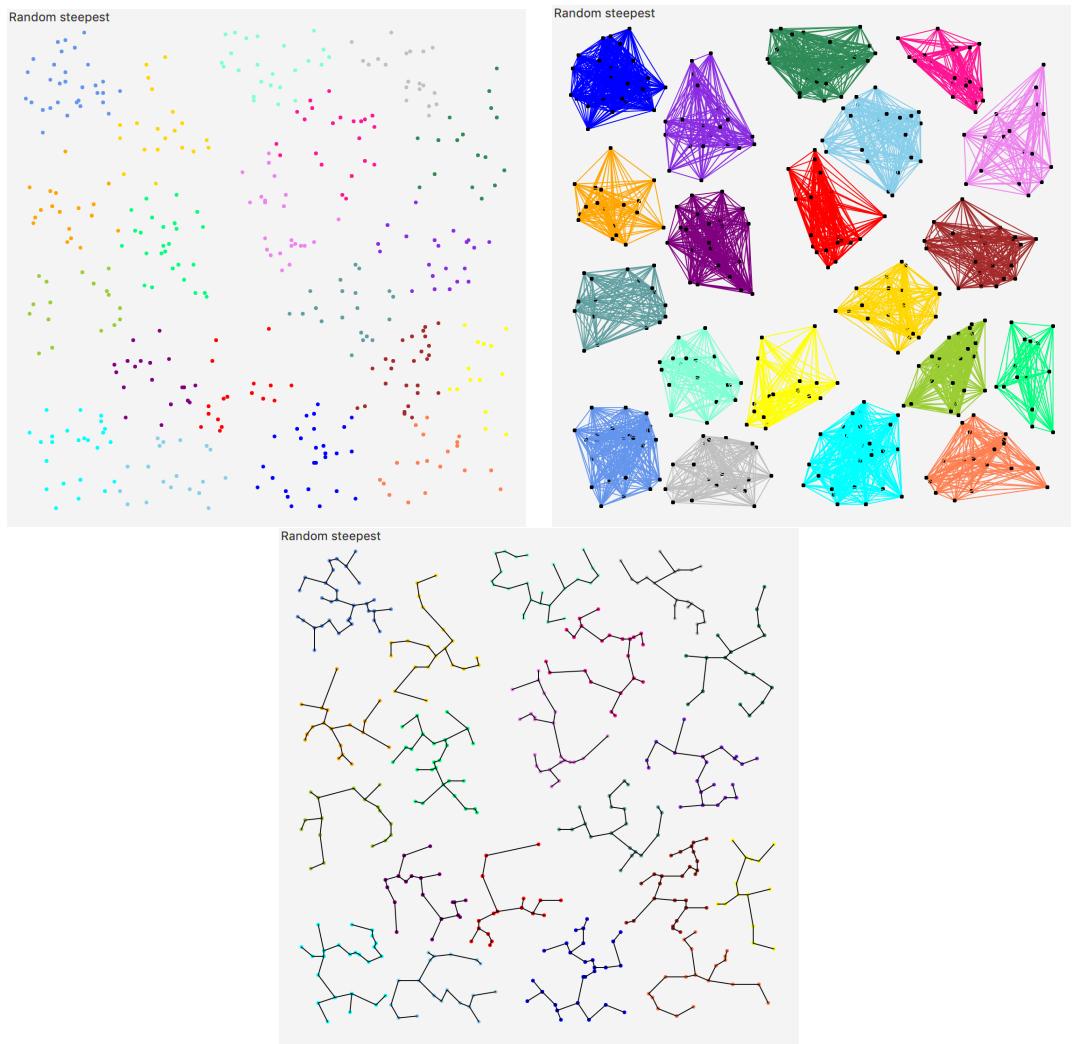
Rysunek 1. Naive Greedy Local Search



Rysunek 2. Random Greedy Local Search



Rysunek 3. Naive Steepest Local Search



Rysunek 4. Random Steepest Local Search

Górka Bartosz Zimniak Kajetan
127228 127229

Algorytmy ewolucyjne i metaheurystyki

Sprawozdanie 1

1. Opis problemu

Celem projektu było przygotowanie dwóch wersji algorytmów rozwiązywających problem grupowania. Liczba grup została ustalona na 10. Funkcja celu została zdefiniowana jako *minimalizacja średniej odległości wszystkich par obiektów umieszczonych w ramach tej samej grupy*.

Jako kluczowe było wykorzystanie macierzy odległości (dystansu pomiędzy punktami) zamiast użycia przestrzeni kartezjańskiej jako punktu wyjścia. Takie założenie pozwala wykorzystać algorytmy również w przypadku zmiany funkcji odległości (macierz odległości jest wystarczająca do dokonania przydziału).

W rozdziale 2 zaprezentowano pseudokody przygotowanych algorytmów, natomiast w rozdziale 3 wyniki działania algorytmów dla 100 iteracji. Ostatni rozdział dotyczy wizualizacji najlepszych uzyskanych rozwiązań.

2. Pseudokody przygotowanych algorytmów

2.1. Algorytm zachłanny

```
Dla każdego punktu p z listy punktów {  
    Dla każdej grupy g z listy grup {  
        Sprawdź wartość odległości między punktem p, a elementem startowym grupy g  
  
        Jeżeli odległość mniejsza od aktualnie najmniejszej {  
            Zapisz odległość jako najmniejszą  
            Jako wybraną grupę zapisz g  
        }  
    }  
    Do wybranej grupy z najmniejszym dystansem dodaj punkt p  
}
```

2.2. Algorytm z wykorzystaniem żalu

Algorytm z wykorzystaniem żalu dokonuje przydziału obiektu zgodnie z pseudokodem zaprezentowanym poniżej. W każdej iteracji przydziela jeden punkt do jednej grupy. Dla każdego punktu

sprawdza jak bardzo jego dolożenie do danej grupy pogorszy wartość funkcji celu. Następnie wybiera punkt, którego dolożenie do którejś z grup będzie najmniej korzystne i dołącza go do grupy, która znajduje się najbliżej niego (najkorzystniejszy wybór wśród wszystkich grup). Inicjalizacja punktów startowych odbywa się poprzez losowy wybór elementu startowego w każdej z grup.

```

Dopóki nie przydzielono wszystkich punktów {
    Dla każdego punktu p z listy wszystkich punktów {
        Dla każdej grupy g z listy grup {
            Oblicz średni dystans pomiędzy wszystkimi punktami w grupie g
            Zapisz jak bardzo dodanie punktu zmienia wartość funkcji celu
        }
    }
}

Wybierz grupę i punkt które mają największą wartość żalu
(minimalizacja kosztów dodania punktu w kolejnych krokach)
Dodaj wybrany punkt do wybranej grupy
}

```

3. Wyniki eksperymentów obliczeniowych

W tabeli 1 zaprezentowano wyniki eksperymentów obliczeniowych. Dokonano 100 powtórzeń obliczeń, za każdym razem z losowym wyborem elementu startowego w każdej z 10 grup. Za losowy element startowy uznaje się przydział 10 różnych punktów do 10 różnych grup (każda z powstałych grup miała jeden punkt). Przydzielony punkt określany jest punktem startowym grupy.

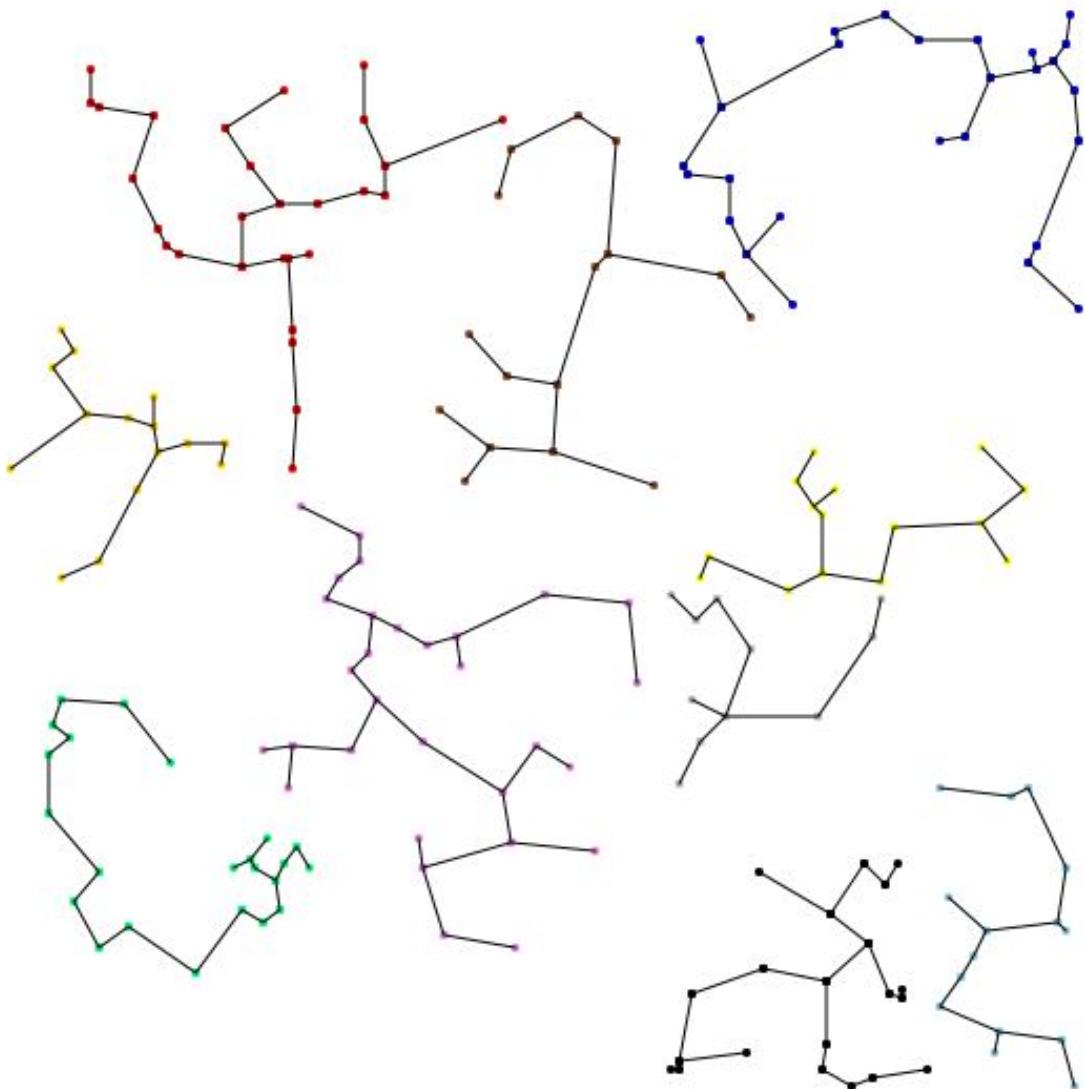
Obydwia algorytmy wykorzystywały ten sam przydział początkowy grup w ramach pojedynczej iteracji, aby możliwe było ich porównanie.

Tabela 1. Wyniki eksperymentów obliczeniowych dla 100 iteracji

Cecha	Algorytm zachłanny	Algorytm oparty o żal
Wartość minimalna funkcji celu	33.92	37.62
Wartość maksymalna funkcji celu	44.44	73.82
Wartość średnia funkcji celu	38.47	48.36
Wartość minimalna czasu obliczeń [sec]	0,000804	0,18
Wartość maksymalna czasu obliczeń [sec]	0,029	1,52
Wartość średnia czasu obliczeń [sec]	0,0017	0,258

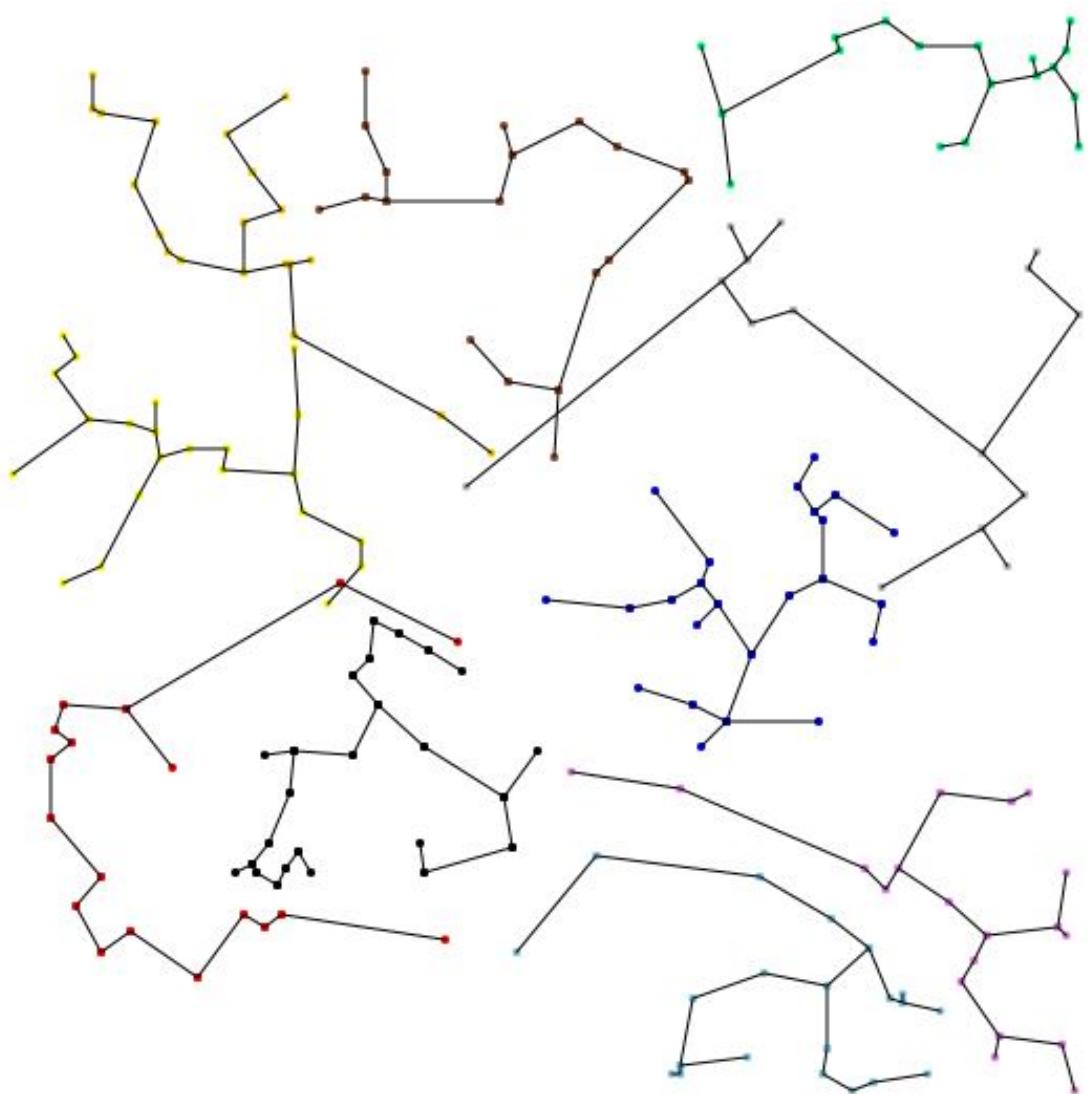
4. Wizualizacja najlepszych rozwiązań

4.1. Algorytm zachłanny



Rysunek 1. Algorytm zachłanny - wizualizacja najlepszego przydziału

4.2. Algorytm oparty o żal



Rysunek 2. Algorytm oparty o żal - wizualizacja najlepszego przydziału