

Górka Bartosz      Zimniak Kajetan  
127228                127229

# Algorytmy ewolucyjne i metaheurystyki

Sprawozdanie 4

## 1. Opis etapu projektu

Celem kolejnego etapu projektu była implementacja dwóch algorytmów przeszukiwania lokalnego. Pierwszym z nich był **Multiple Start Local Search**. Został on uruchomiony 10-cio krotnie, każdorazowo realizując 100 operacji przeszukiwania lokalnego ze zrandomizowanym przydziałem punktów do grup.

Drugi algorytm to **Iterated Local Search**, który został przygotowany w trzech wersjach. Pierwsza realizuje małą perturbację polegającą na przeniesieniu 2% punktów pomiędzy grupami w sposób zrandomizowany. Dwie pozostałe realizują dużą perturbację z przeniesieniem 30% punktów. W fazie **Destroy** zostają wytypowane i usunięte punkty. Jedna z wersji realizuje to w sposób losowy, druga natomiast wybiera najbardziej oddalone od reszty grupy punkty. Kolejną fazą jest **Repair** z przydziałem punktu do grupy, dla której najmniej pogarsza wartość funkcji celu.

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. Ostatni rozdział dotyczy wizualizacji najlepszych uzyskanych rozwiązań.

## 2. Pseudokody

### 2.1. Multiple Start Local Search

W pętli wykonaj iterację aż osiągniesz limit iteracji lokalnego przeszukiwania {  
    Zainicjalizuj losowy przydział punktów do grup

    W zależności od parametru metody, wykonaj przeszukiwanie lokalne z wykorzystaniem algorytmu Greedy bądź Steepest

    Jeżeli osiągnięto lepsze rozwiązanie niż do tej pory najlepsze {

        Zapisz wartość funkcji celu

        Zapisz uzyskany przydział punktów do grup

    }

}

W naszym przypadku procedura opisana wyżej powtarzana jest 10-cio krotnie, za każdym razem zapamiętujemy uzyskany wynik i przydziął punktów do grup.

## 2.2. Iterated Local Search

Wyznacz liczbę punktów do przeniesienia w zależności od użytej perturbacji

Wykonuj dopóki nie przekroczone limitu czasu wykonania {

Dokonaj perturbacji rozwiązania startowego

Z wykorzystaniem algorytmu Steepest wyznacz optimum lokalne

Jeżeli wyznaczone rozwiązanie jest lepsze niż dotychczas najlepsze {

Zapamiętaj wartość rozwiązania jako nowe najlepsze

Za rozwiązanie startowe przypisz uzyskane rozwiązanie

}

}

Algorytm wykorzystuje metodę perturbacji. Jej dokładna realizacja została przedstawiona w kolejnych podrozdziałach, aby nie powielać opisu.

## 2.3. Mała perturbacja

Wykonuj aż do przeniesienia zakładanej liczby punktów {

Wyznacz losowo grupę z której przenieś punkt

Wyznacz grupę docelową inną niż startowa

Wybierz punkt z grupy

Przenieś go pomiędzy grupami

}

## 2.4. Duża perturbacja losowa

W fazie Destroy usuń zakładaną liczbę punktów z grup (wybierz punkty losowo)

Dla każdego usuniętego punktu {

Wyznacz grupę dla której najmniej pogarsza wartość funkcji celu

Dodaj punkt do wyznaczonej grupy

}

## 2.5. Duża perturbacja heurystyczna

Wyznacz sumę odległości każdego punktu od innych w ramach tej samej grupy  
Posortuj listę zgodnie z malejącą sumą odległości

W fazie Destroy usuń zakładaną liczbę punktów z grup iterując na przygotowanej liście punktów

```

Dla każdego usuniętego punktu {
    Wyznacz grupę dla której najmniej pogarsza wartość funkcji celu
    Dodaj punkt do wyznaczonej grupy
}

```

### 3. Wyniki eksperymentów obliczeniowych

W tabeli 1 zaprezentowano wyniki eksperymentów obliczeniowych. W przypadku **Multiple Start Local Search** podany czas dotyczy obliczeń dotyczących minimalnej/maksymalnej/średniej wartości z 10 uruchomień algorytmu. Dla algorytmu iteracyjnego, wartości czasu obliczeń dotyczą całego procesu (od uruchomienia do wyczerpania limitu czasowego). W algorytmach nie wykorzystano cache ani ruchów kandydackich. Cały proces testowania algorytmów został wykonany 10-cio krotnie, aby możliwe było wskazanie wartości średniej.

Tabela 1. Wyniki eksperymentów obliczeniowych

| Cecha                                   | MSLS  | ILS small perturbation | ILS big perturbation | ILS big perturbation heuristic |
|---|-------|------------------------|----------------------|--------------------------------|
| Wartość minimalna funkcji celu          | 26.37 | 26.74                  | 26.37                | 26.37                          |
| Wartość maksymalna funkcji celu         | 26.39 | 27.82                  | 26.49                | 27.33                          |
| Wartość średnia funkcji celu            | 26.37 | 27.18                  | 26.41                | 26.87                          |
| Wartość minimalna czasu obliczeń [sec]  | 38.59 | 392.69                 | 392.70               | 392.74                         |
| Wartość maksymalna czasu obliczeń [sec] | 52.18 | 446.75                 | 446.81               | 446.82                         |
| Wartość średnia czasu obliczeń [sec]    | 39.97 | 399.76                 | 399.77               | 399.77                         |

Algorytm MSLS osiągał średnio najlepsze wyniki. Porównywalnie sprawdził się ILS z dużymi perturbacjami. Zgodnie z oczekiwaniemi, wprowadzanie niewielkich zmian w przydiale do grup nie wystarczało, aby znaczaco oddalić się od minimów lokalnych i ostatecznie algorytm z małymi perturbacjami nie osiągnął dobrych wyników. W porównaniu z poprzednimi testowanymi algorytmami, stworzone w ramach czwartego etapu projektu działają o wiele dłużej. Spowodowane jest to oczywiście ich charakterystyką, tj. wykorzystują wielokrotne uruchomienie lokalnego przeszukiwania.

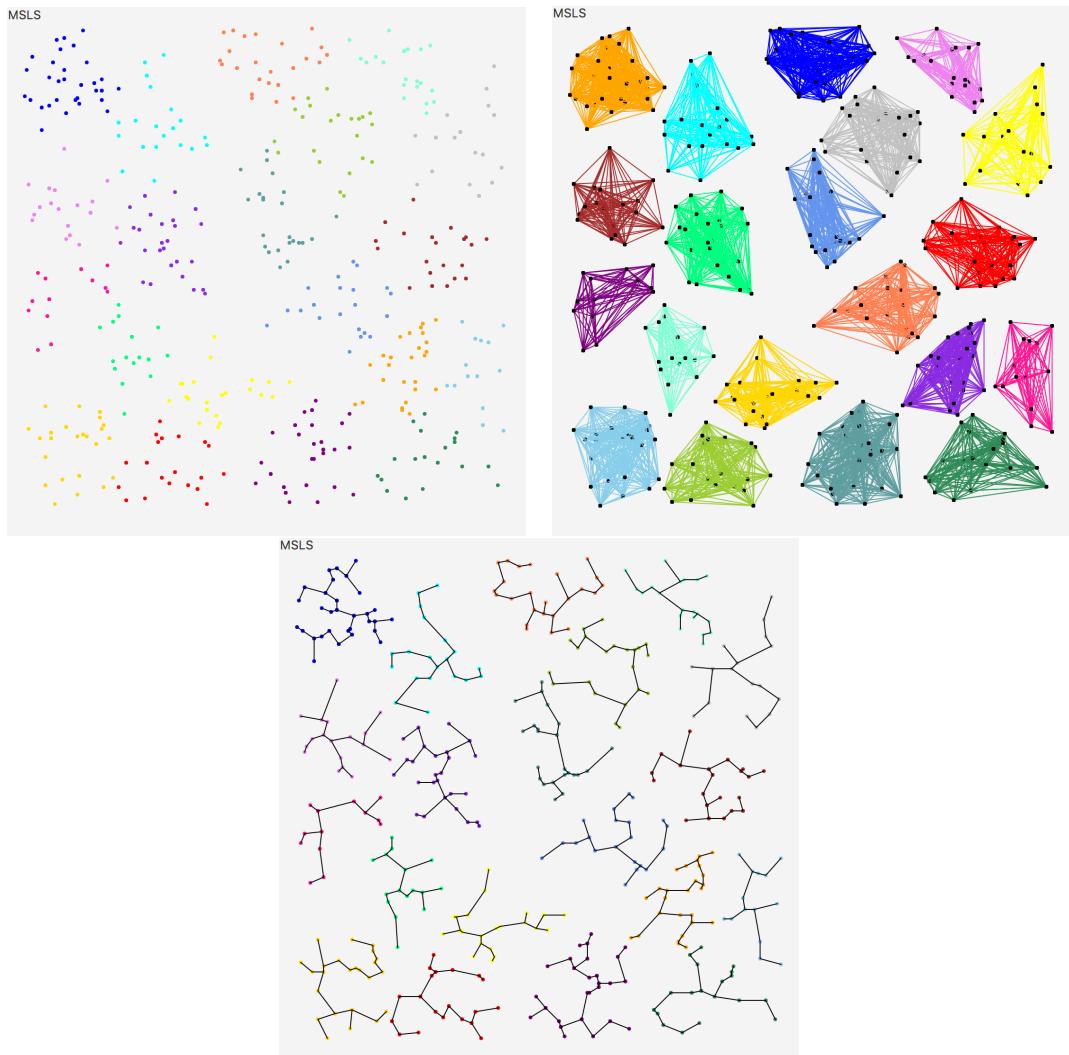
Na wskazanie zasługuje testowany dodatkowo przypadek. Każdy z algorytmów otrzymał ograniczony czas wykonania (20 sekund), w którym powinien poprawić rozwiązanie. Najlepszy wynik uzyskał **algorytm iteracyjnego przeszukiwania lokalnego z heurystyczną funkcją destroy**.

Zdeklasował on zarówno algorytm ILS w pozostałych wersjach, jak i MSLS uzyskując wynik **26.41**, podczas gdy pozostałe algorytmy ponad **26.57** (potrzebowaliby one dodatkowy czas, aby móc poprawić rozwiązanie).

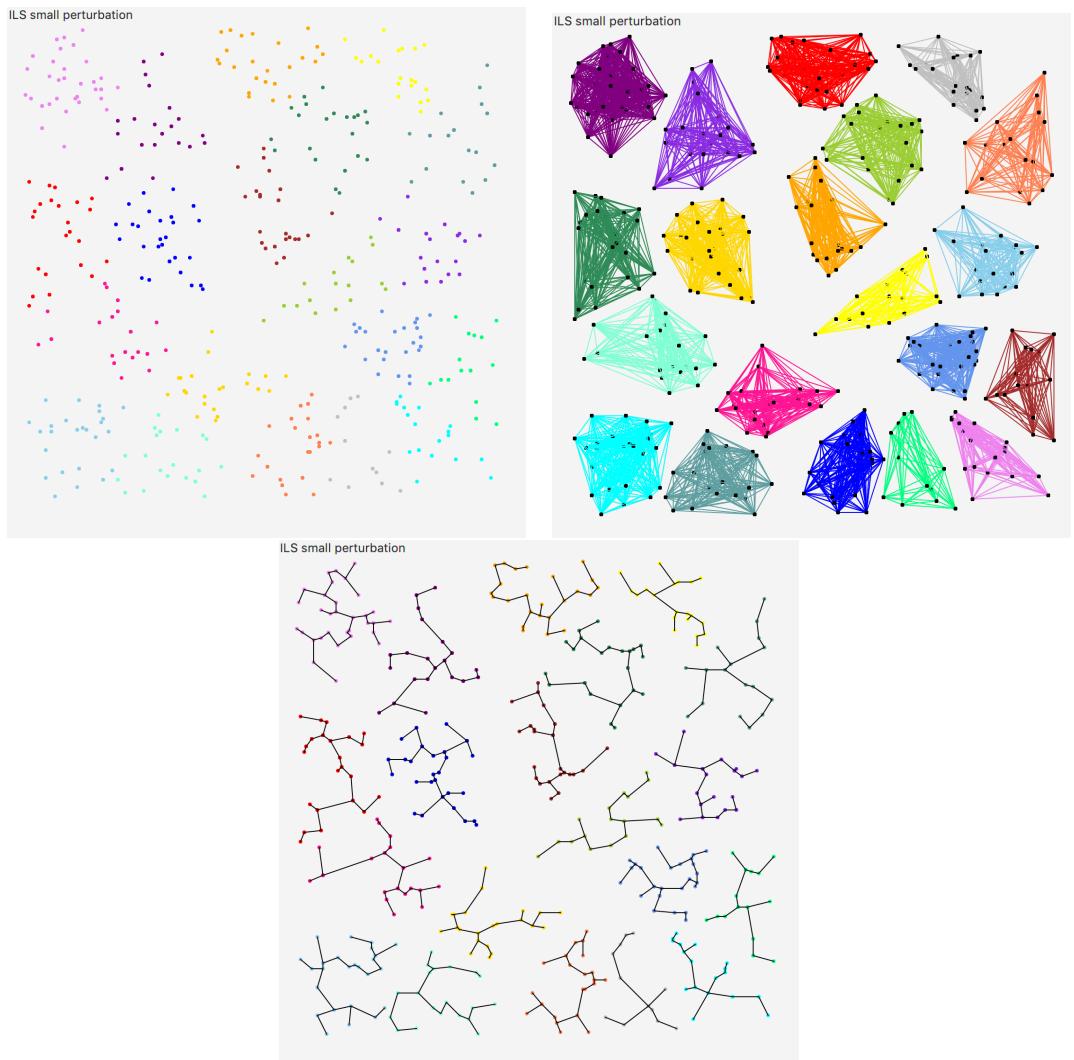
### 4. Wizualizacja najlepszych rozwiązań

Podobnie jak w poprzednich etapach 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 zaprezen-

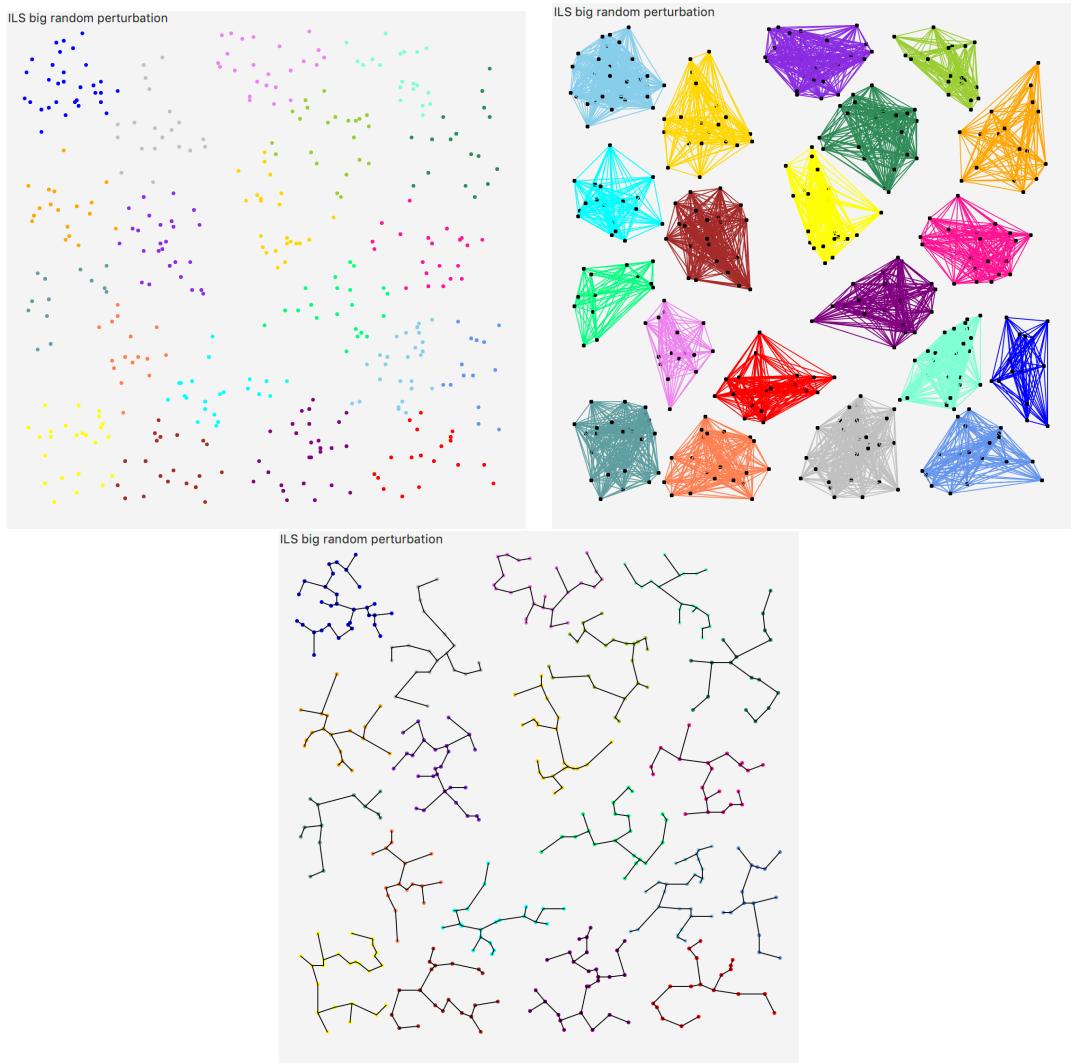
towanie powiązań pomiędzy punktami w ramach grupy. Ostatni sposób wykorzystuje minimalne drzewo rozpinające, które w przejrzysty sposób prezentuje przydział punktów do grup.



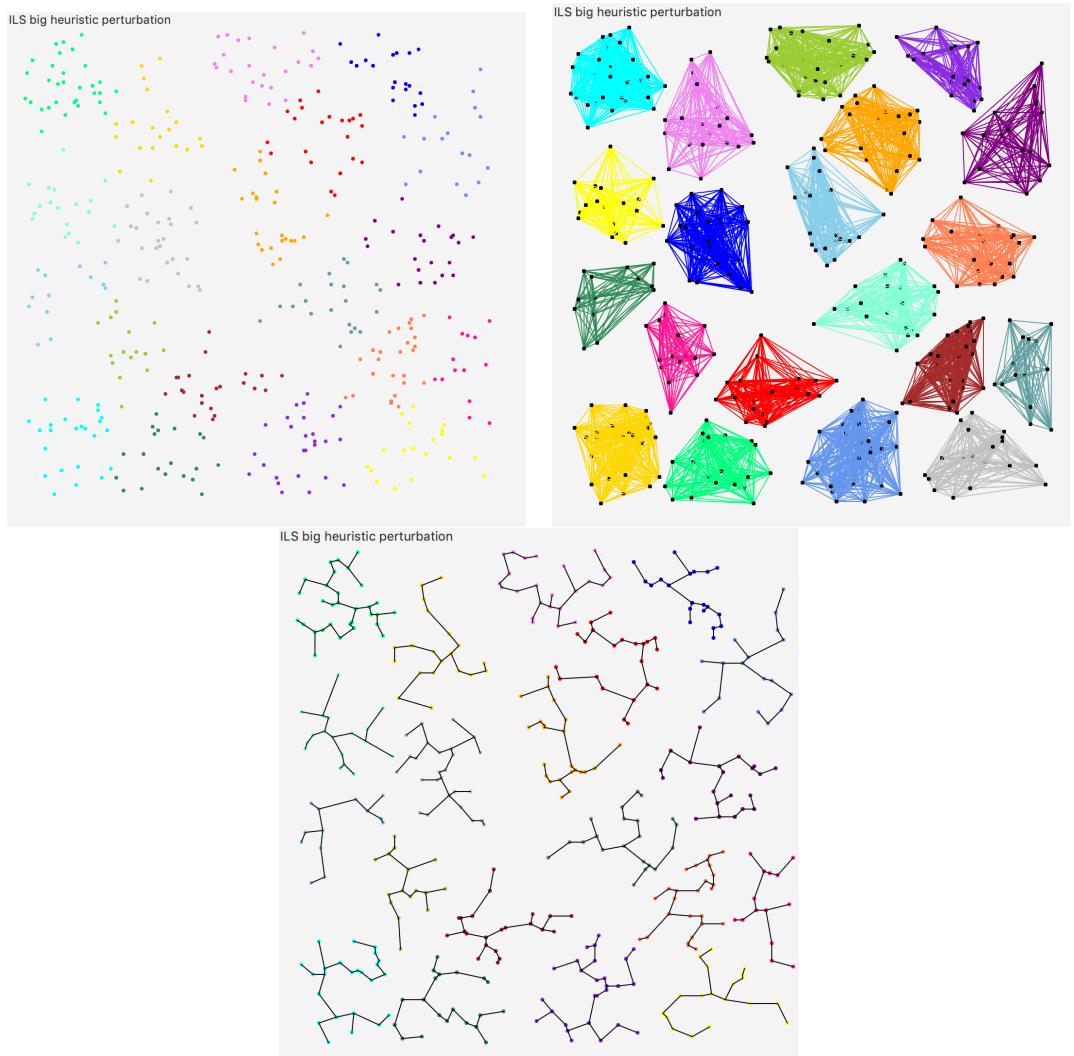
Rysunek 1. MSLS



Rysunek 2. ILS small perturbation



Rysunek 3. ILS big perturbation



Rysunek 4. ILS big perturbation heuristic

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 10 najbliższych punktów p2-p11 różnych od p1 należących do grup g2-g11  
            innych niż g1  
            Do listy możliwych ruchów dodaj przesunięcie punktu p1 z grupy g1 do grup g2-g11  
        }  
    }  
}
```

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
}

```

### 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.

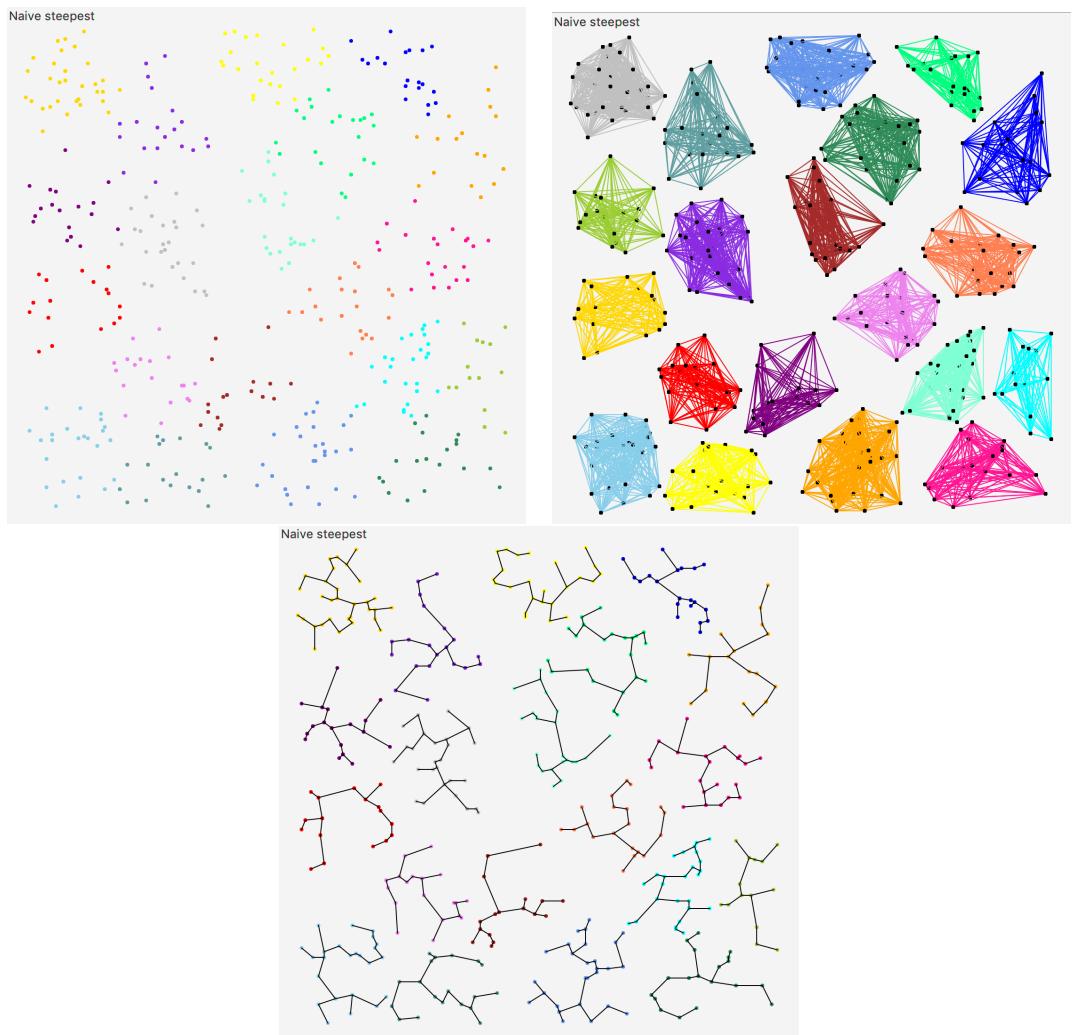
Tabela 1. Wyniki eksperymentów obliczeniowych dla 100 iteracji

| Cecha                                   | Naive Steepest | Naive Steepest + Cache | Naive Steepest + Candidates | Naive Steepest with Cache + Candidates |
|---|----------------|------------------------|-----------------------------|--|
| Wartość minimalna funkcji celu          | 26.39          | 26.39                  | 26.65                       | 27.15                                  |
| Wartość maksymalna funkcji celu         | 28.67          | 28.67                  | 31.25                       | 31.39                                  |
| Wartość średnia funkcji celu            | 27.27          | 27.27                  | 27.82                       | 27.96                                  |
| Wartość minimalna czasu obliczeń [sec]  | 0.81           | 0.57                   | 0.55                        | 0.13                                   |
| Wartość maksymalna czasu obliczeń [sec] | 2.32           | 3.16                   | 2.05                        | 2.52                                   |
| Wartość średnia czasu obliczeń [sec]    | 1.40           | 1.19                   | 0.96                        | 0.68                                   |

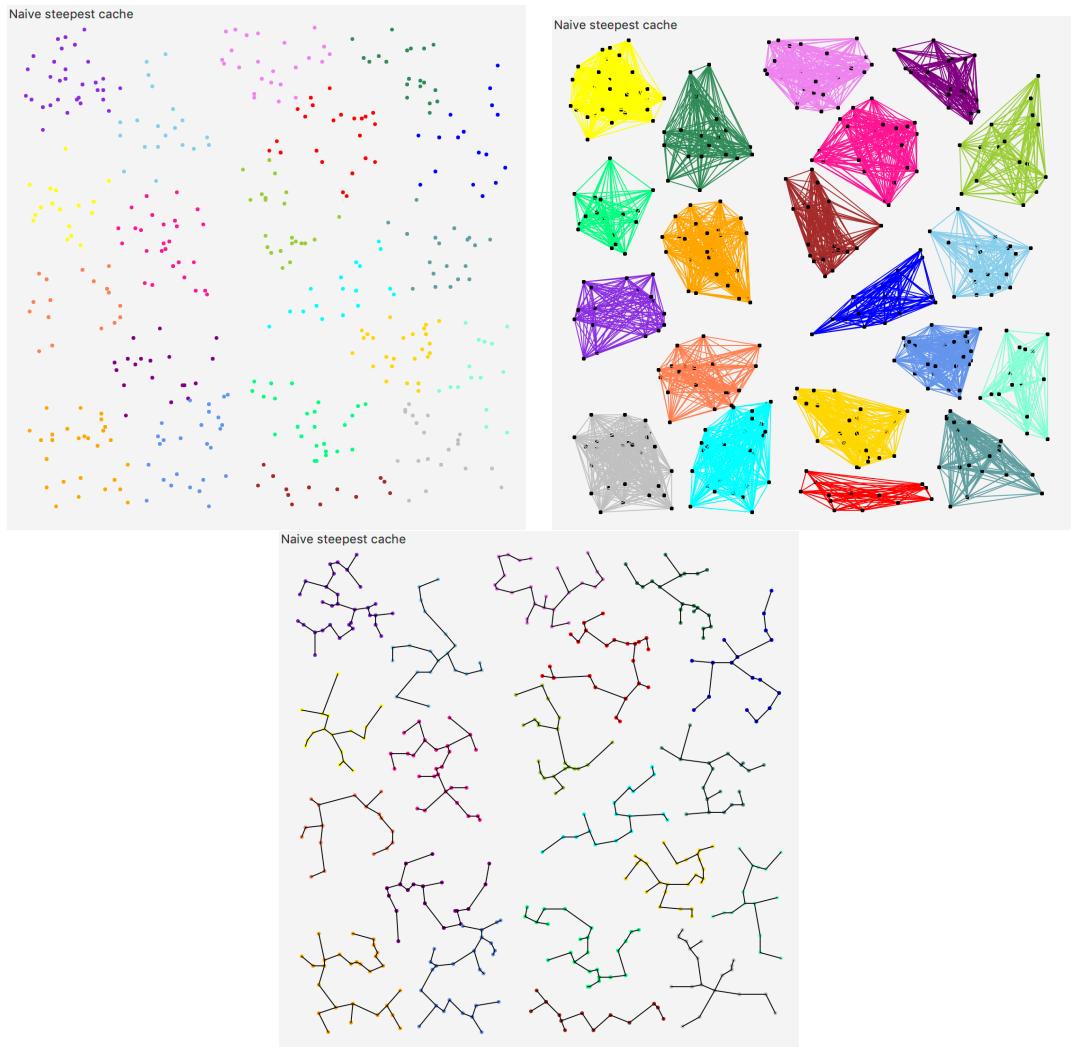
Zgodnie z oczekiwaniemi, zastosowanie mechanizmu łuków kandydackich pozwoliło przyspieszyć działanie algorytmu. Za zmianę czasu wykonania algorytmu musimy jednakże liczyć się z nieznacznym pogorszeniem wartości funkcji celu.

### 4. Wizualizacja najlepszych rozwiązań

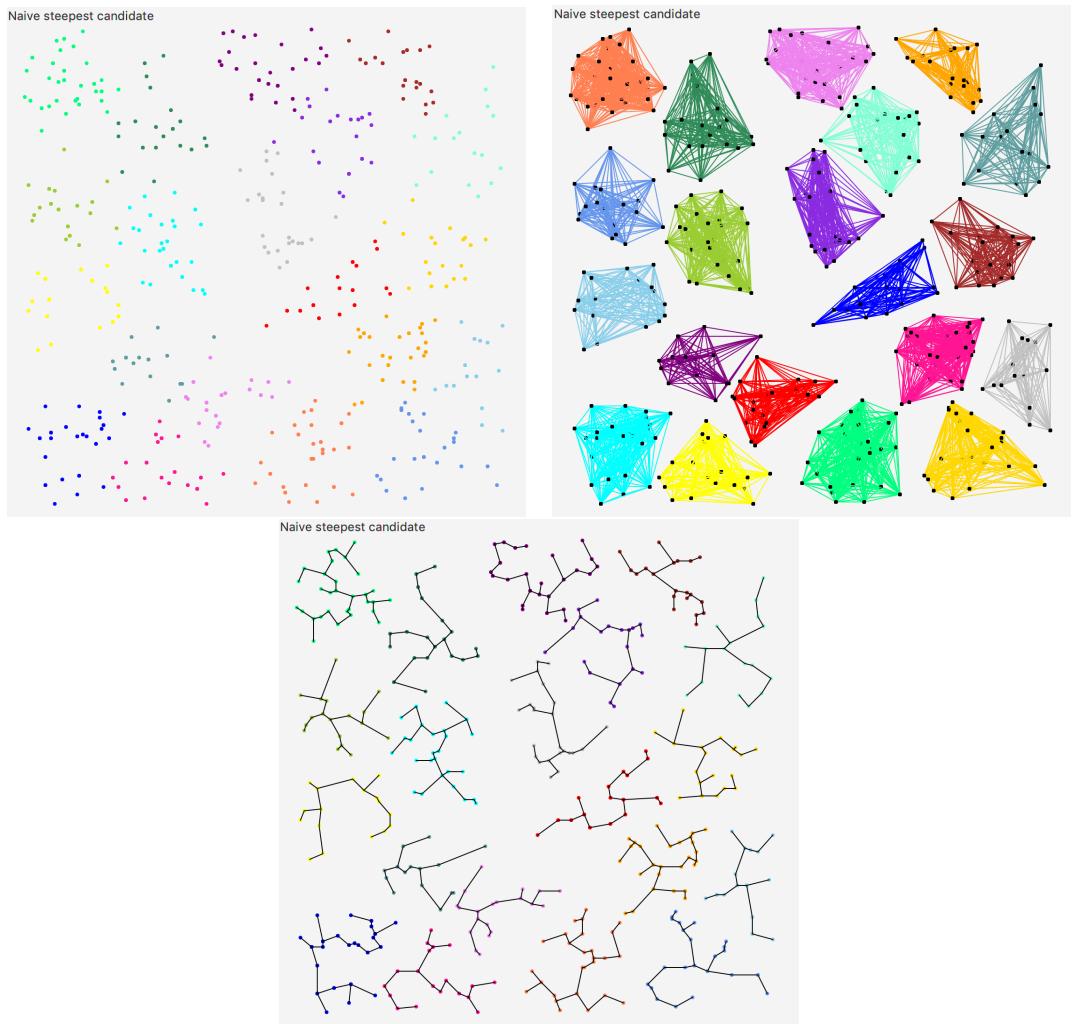
Podobnie jak w poprzednim etapie projektu, również w tym zastosowano wizualizację najlepszych rozwiązań na trzy sposoby. Pierwszy z nich to zaprezentowanie samych grup punktów, bez jakichkolwiek powiązań. Drugim sposobem jest prezentacja zgodna z funkcją celu, czyli zaprezentowanie powiązań pomiędzy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



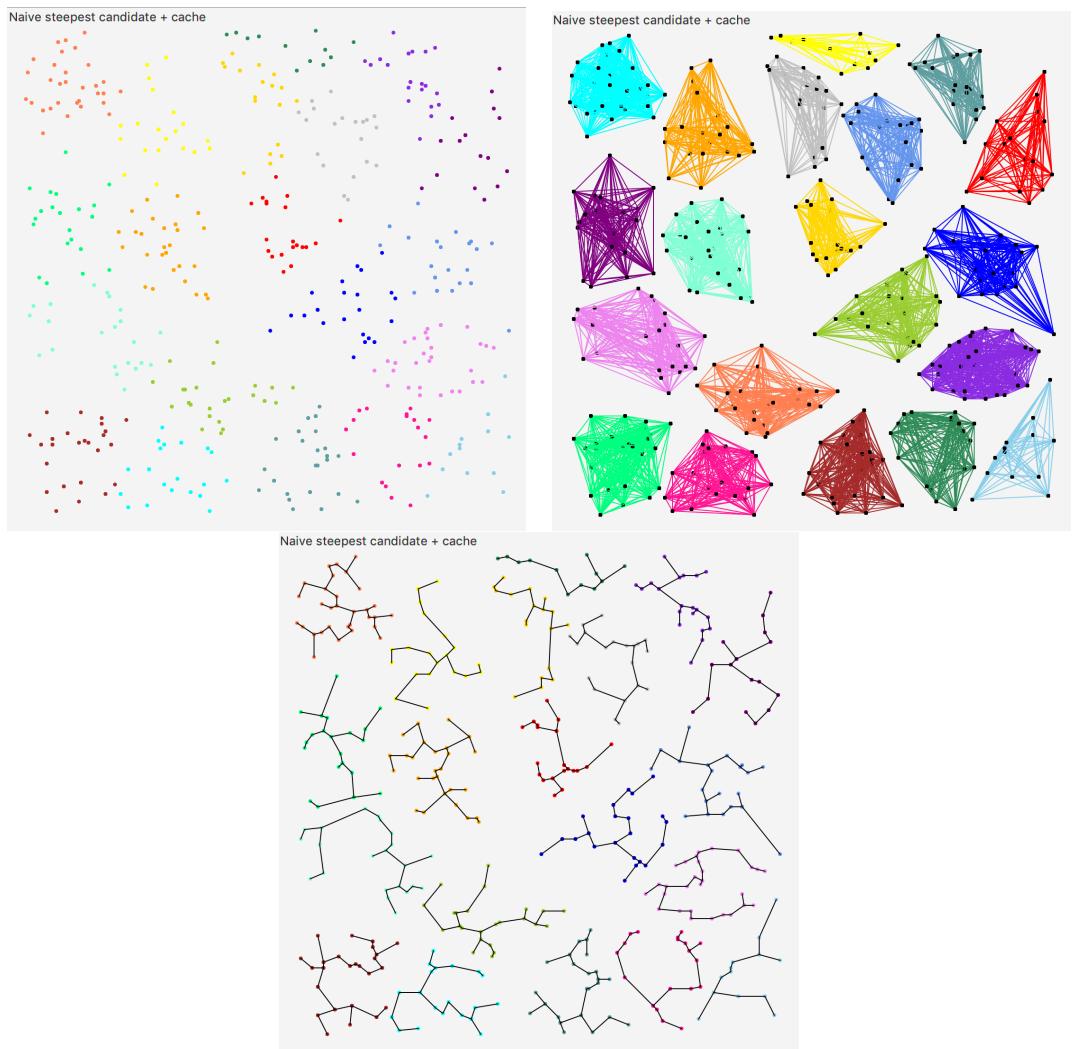
Rysunek 1. Naive Steepest Local Search



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

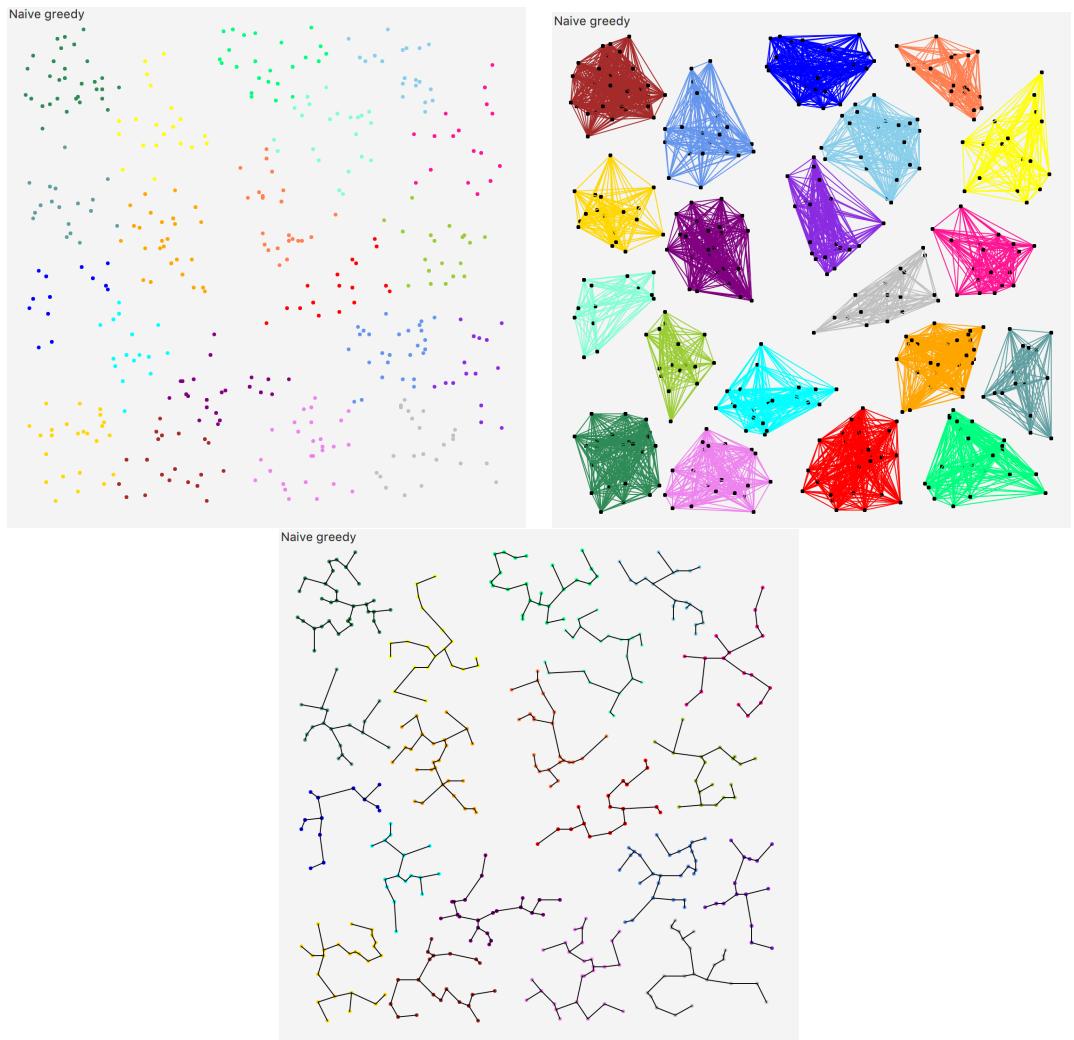
W tabeli 1 zaprezentowano wyniki eksperymentów obliczeniowych. Dokonano 100 powtórzeń obliczeń. Za każdym razem algorytmy zostały uruchomione dla dwóch rozwiązań startowych. Pierwszy z nich to wynik działania algorytmu naïwnego przygotowanego w poprzednim ćwiczeniu, a drugi to przydział przygotowany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

Tabela 1. Wyniki eksperymentów obliczeniowych dla 100 iteracji

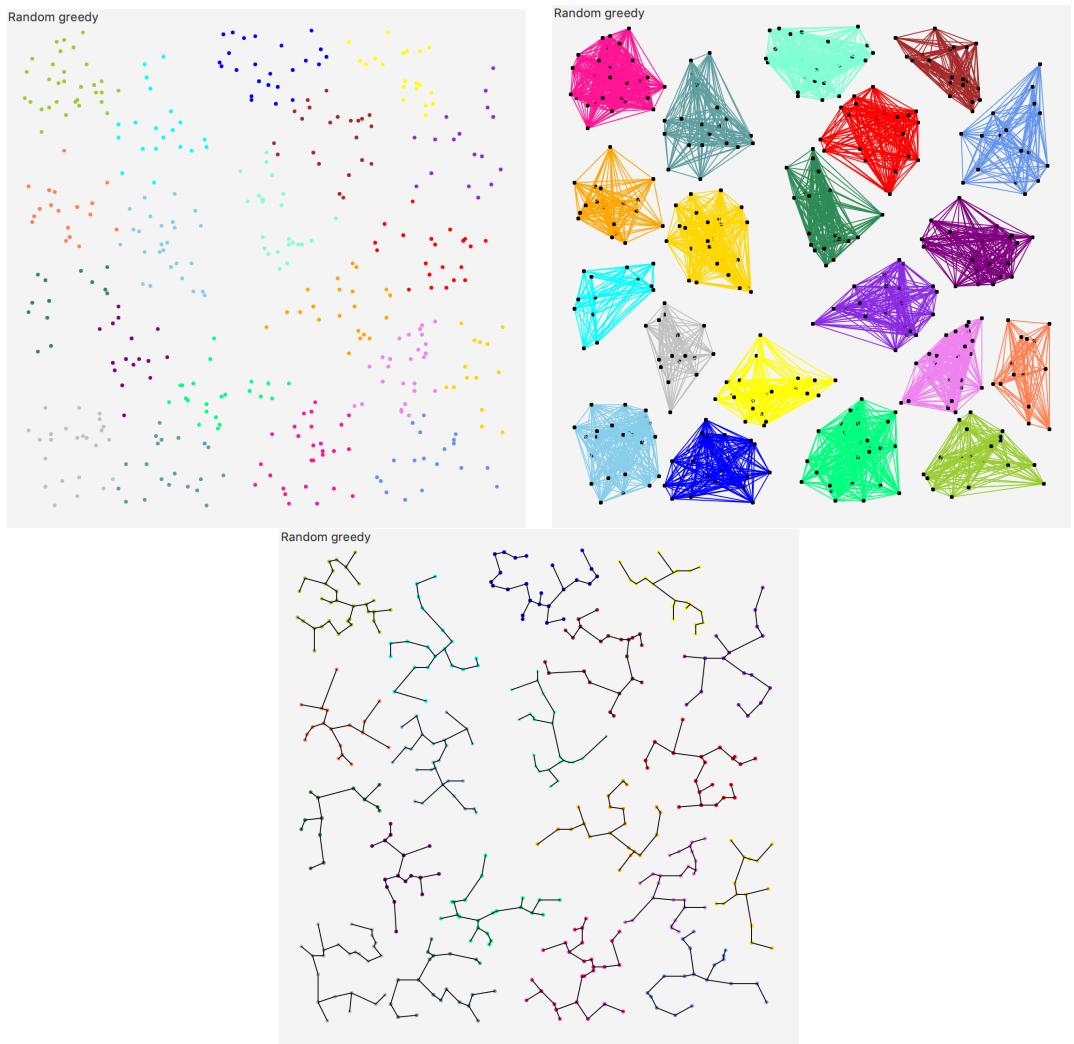
| Cecha                                   | Naive Greedy LS | Random Greedy LS | Naive Steepest LS | Random Steepest LS |
|---|-----------------|------------------|-------------------|--------------------|
| Wartość minimalna funkcji celu          | 26.39           | 26.37            | 26.39             | 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

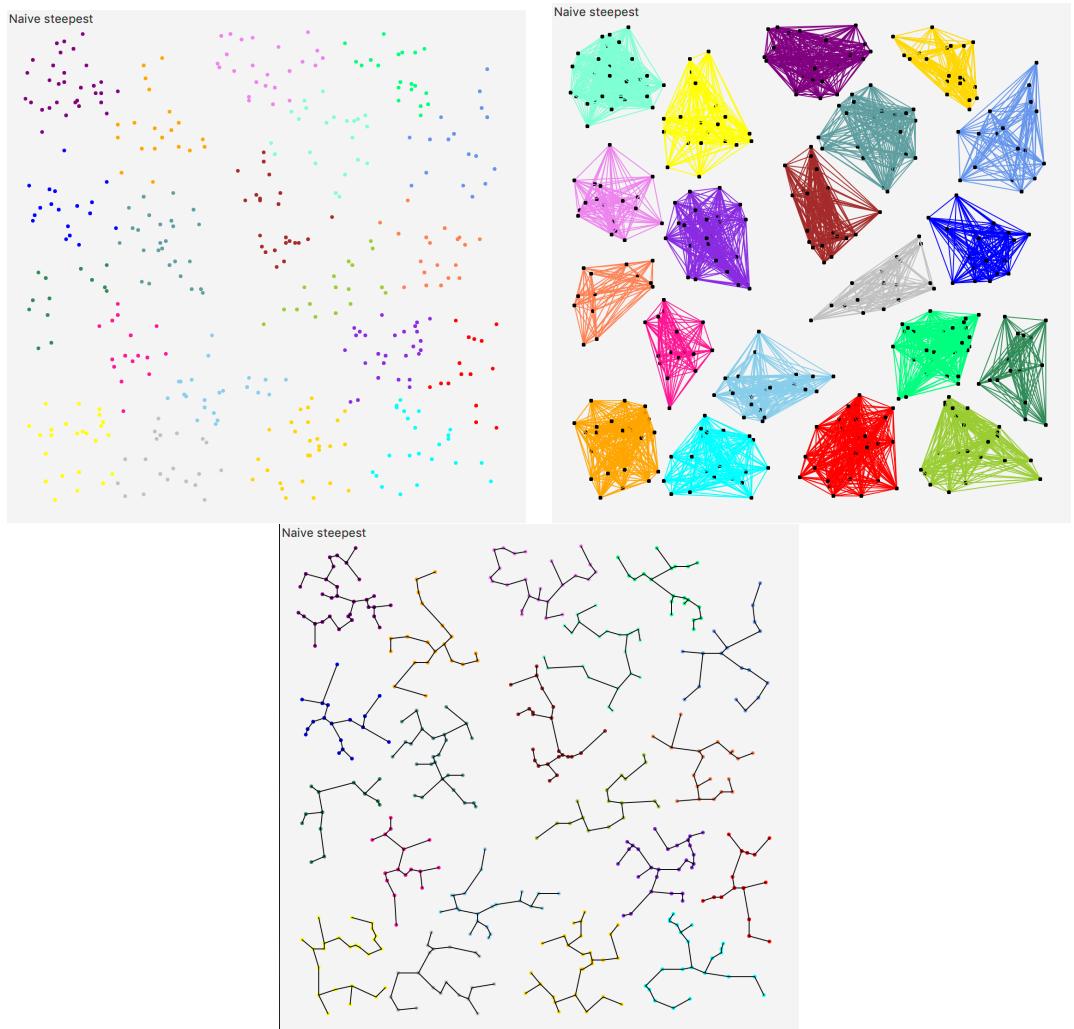
W wizualizacji najlepszych rozwiązań wykorzystano trzy sposoby prezentacji rozwiązań. Pierwszy z nich to zaprezentowanie samych grup punktów, bez jakichkolwiek powiązań. Drugim sposobem jest prezentacja zgodna z funkcją celu, czyli zaprezentowanie powiązań pomiędzy punktami w ramach grupy. Ostatni sposób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



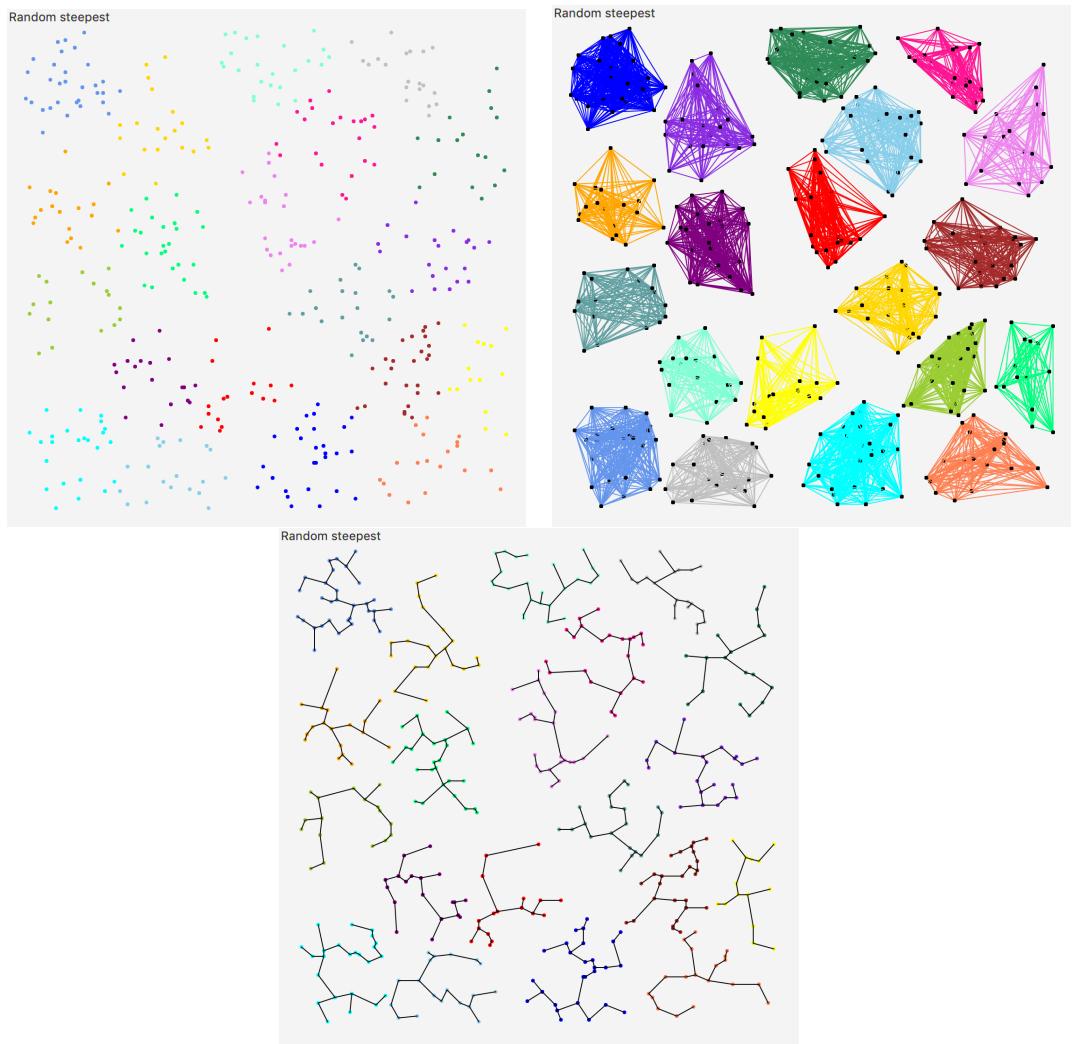
Rysunek 1. Naive Greedy Local Search



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.

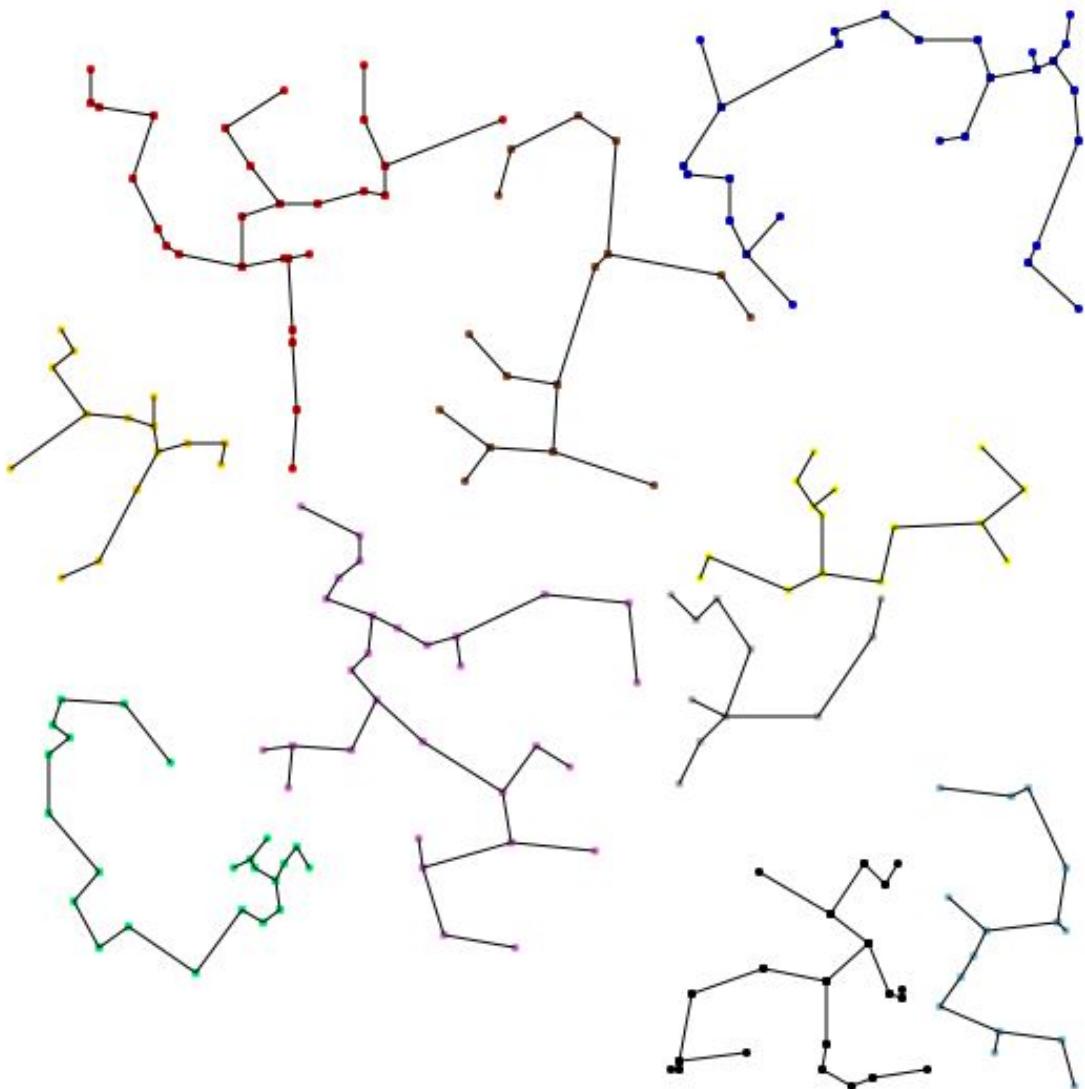
Obydwia algorytmy wykorzystywały ten sam przydział początkowy grup w ramach pojedynczej iteracji, aby możliwe było ich porównanie.

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                 |

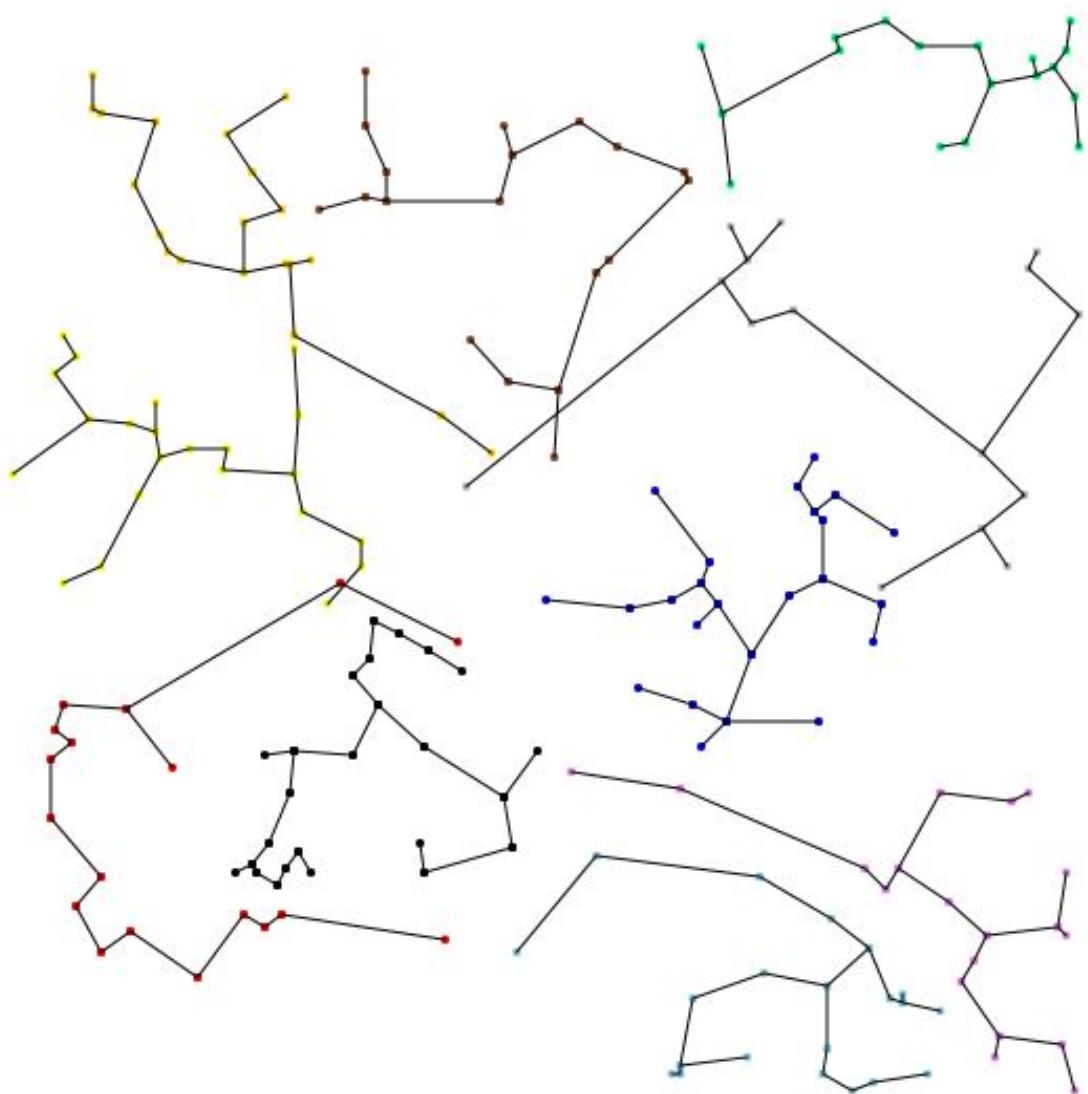
## 4. Wizualizacja najlepszych rozwiązań

### 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