Joanna Nużka
Kacper Szmajdel
Marcin Pilarski

Problem układania instalacji wodociągowej

1. Wstęp

Celem projektu jest poznanie często stosowanych w obszarze Badań Operacyjnych algorytmów: symulowanego wyżarzania oraz genetycznego.

2. Opis zagadnienia

Problem polega na ułożeniu sieci wodociągowej w taki sposób, by dostarczyć odpowiednią ilość wody do określonych punktów poboru. Zależy nam na jak najtańszym i najszybszym sposobie ułożenia tej instalacji (jest to problem optymalizacji wielokryterialnej). Wodociąg jest układany na krawędziach „szachownicy".

Dane:

- koszt i czas położenia instalacji na każdej krawędzi szachownicy, w obu możliwych kierunkach
- współrzędne punktów poboru
- wartość zapotrzebowania na wodę w każdym punkcie
- ilość wody pobieraną ze źródła

Ograniczenia:

- dx, dy – minimalna odległość między węzłami/rozgałęzieniami w osi x oraz y. Przykładowo jeśli jeden z węzłów jest w punkcie (x, y) = (xi, yj) to następne węzły mogą się pojawić jedynie poza obszarem (xi-dx÷ xi +dx, yj -dy÷ yj +dy)
- Nie mogą być dwie rury obok siebie
- Rura kończy się na odbiorcy
- Mamy różne typy węzłów – dzielące wodę w różny sposób
- Kara za niedostarczenie wody

Funkcja celu:

$$F_{cel} = \sum_i \left(\frac{k_i}{k_{max}} \cdot w_k + \frac{t_i}{t_{max}} \cdot w_t \right) + \frac{\sum_j p_j}{\sum_l p_l} n \rightarrow min$$

w_k – waga kosztów kładzenia instalacji
w_t – waga czasów kładzenia instalacji
k_i – koszt kładzenia i-tej krawędzi wchodzącej w skład danego rozwiązania
t_i – czas kładzenia i-tej krawędzi wchodzącej w skład danego rozwiązania
k_{max} – maksymalna wartość wśród kosztów kładzenia instalacji
t_{max}– maksymalna wartość wśród czasów kładzenia instalacji
p_j - wartość brakującego poboru w punktach poboru
p_l - wartość poboru wszystkich punktów poboru
n – dotkliwość kary za nie dotarcie do punktu poboru

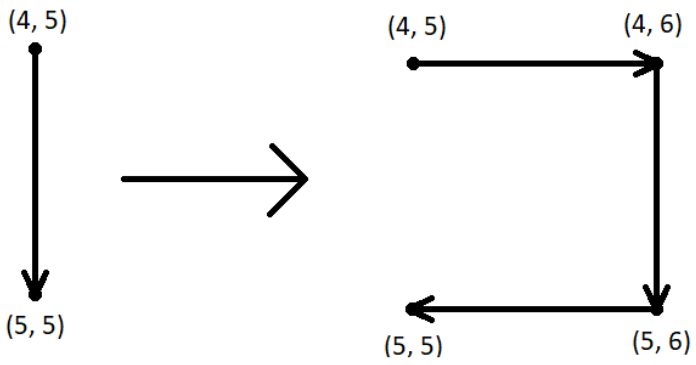
3. Opis algorytmów

Dla obu algorytmów rozwiązanie początkowe/populacja rozwiązań początkowych jest generowana losowo.

3.1. Algorytm SA – algorytm ten polega na losowym wyznaczaniu rozwiązań sąsiadujących z obecnym i sprawdzaniu wartości funkcji celu dla obu rozwiązań. Jeśli „sąsiad" jest lepszy od obecnego, zostaje on wybrany na rozwiązanie w kolejnej iteracji. W przeciwnym wypadku losowana jest wartość, która decyduje, bazując na rozkładzie Boltzmanna, o tym, czy rozwiązanie zostanie zamienione na gorsze.

W przypadku naszego problemu mieliśmy kilka pomysłów na zdefiniowane sąsiedztwa rozwiązania (pogrubione zostały wprowadzone):

- Usunięcie węzła wychodzącego – losujemy krawędzie z węzła i usuwamy je oraz następne aż do uzyskania rozwiązania dopuszczalnego
- Dodanie węzła – losujemy punkt, w którym dodajemy węzeł wychodzący, krawędzie prowadzimy aż do uzyskania rozwiązania dopuszczalnego
- Usunięcie krawędzi krańcowych – usuwamy krawędzie krańcowe (zaczynając od wylosowanej) aż otrzymamy rozwiązanie dopuszczalne
- Dodanie krawędzi do węzła o niezadowalającym przepływie
- **Zmiana typu węzła – zmiana stosunków przepływów w już istniejącym węźle**
- **Zamiana krawędzi – zamiana jednej krawędzi na maksymalnie 3, omijające tę krawędź:**



Rysunek 1. Wizualizacja sąsiedztwa: zamiana krawędzi

Kryteria stopu w algorytmie SA to kryterium maksymalnej liczby iteracji i kryterium osiągnięcia przez temperaturę wartości w przybliżeniu równej 0 (z dokładnością maszynową).

Algorytm SA jest realizowany zgodnie z poniższym schematem:

```
best_solution = initial_solution  (początkowo najlepsze rozwiązanie to rozwiązanie
                                   wejściowe)
actual_solution = initial_solution  (aktualne rozwiązanie to rozwiązanie początkowe)
t = t0      (temperatura aktualna to temperatura wejściowa)
while iteration_number < max_it_number and t > 0: (kryteria stopu)
        for i in range(iteration_number_in_one_temp):
                solution_prim = neighborhood(actual_solution) (znalezienie rozwiązania
                                                               w sąsiedztwie aktualnego)
                if objective_function(solution_prim) < obejctive_function(best_solution):
                        best_solution=solution_prim (aktualizacja najlepszego rozwiązania)
                delta=objective_function(solution_prim) -objective_function(actual_solution)
                        (różnica wartości funkcji celu dla rozwiązania aktualnego i sąsiedniego)
                If delta < 0:  (jeśli nastąpiła poprawa rozwiązania)
                        actual_solution = solution_prim  (aktualizacja aktualnego rozwiązania)
                else:
                        x = random(0,1)   (wylosowanie liczby od 0 do 1)
                        if x < exp(-delta/t):
                                actual_solution = solution_prim
        t = alfa * t (aktualizacja wartości temperatury)
```

3.2. Algorytm genetyczny – algorytm polegający na modyfikacji populacji rozwiązań aż do spełnienia warunku stopu. Składa się z trzech etapów: selekcji, krzyżowania oraz mutacji. Jest on realizowany zgodnie z poniższym schematem:

1. Generacja populacji losowych rozwiązań początkowych

2. Zapisanie najlepszego rozwiązania

W pętli dopóki nie zostanie osiągnięte kryterium stopu:

3. Krzyżowanie rozwiązań

4. Mutacja otrzymanych rozwiązań

(5. Ewentualne ulepszenia – opisane poniżej)

6. Ewentualna zmiana najlepszego rozwiązania

8. Dodanie potomków do zbioru rozwiązań

7. Selekcja – wybór nowej populacji rozwiązań (ze zbioru rodziców i potomków dołączonych w tym pokoleniu)

Selekcja jest w naszym przypadku realizowana przez turniej – porównywanie par rozwiązań i wybieranie lepszego z pary, pod względem wartości funkcji celu, do zbioru „rodziców".

Następnie losowo sprawdzamy czy kolejne, wybierane losowo pary rodziców mogą zostać skrzyżowane i krzyżujemy je w przypadku sukcesu. Sam proces krzyżowania polega na wybraniu punktu krzyżowania, znajdującego się w obu rozwiązaniach, a następnie na utworzeniu dwóch nowych rozwiązań: jedno zawiera krawędzie pierwszego rodzica, do których można dojść od źródła, nie przechodząc przez punkt krzyżowania, a także krawędzie drugiego rodzica, wychodząc od punktu krzyżowania i idąc do końca; drugie przeciwnie. Jeżeli podczas wykonywania operacji okazuje się, że nie możemy dodać jakiejś krawędzi, bo istnieje już taka dodana z pierwszego z rodziców, to nie jest ona dodawana tak, by cały czas otrzymywać rozwiązanie dopuszczalne. Jeżeli nie możemy dodać krawędzi z innego powodu, krzyżowanie nie jest wykonywane i jako wynik zwracany jest jeden z rodziców. W ogólnym przypadku w wyniku krzyżowania możemy otrzymać rozwiązanie, w którym nie wszystkie punkty poboru są odwiedzone.

Przykładowo, dla poniższych rozwiązań początkowych i zaznaczonego na żółto punktu krzyżowania w skład pierwszego rozwiązania wejdą krawędzie z obu rozwiązań zaznaczone na niebiesko.

Rysunek 2. Wizualizacja krzyżowania

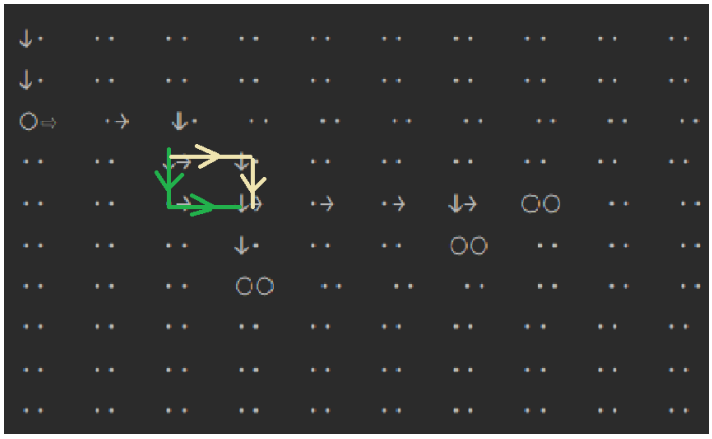
Sposoby przedstawienia sąsiedztwa rozwiązań z algorytmu SA zostały wykorzystane ponownie, jako sposoby mutacji rozwiązań.

Kryterium stopu dla naszego algorytmu to maksymalna liczba iteracji.

Dodatkowo mieliśmy różne pomysły dotyczące ulepszeń uzyskiwanych rozwiązań. Ulepszenia te mogłyby być wywoływane dla wszystkich rozwiązań populacji w każdej lub co którejś iteracji algorytmu. Byłyby one wykonywane po operacji mutacji. Niestety, z braku czasu nie udało nam się ich zaimplementować.

Ulepszenia:

- Usuwanie zduplikowanej drogi – jeśli w rozwiązaniu między jakimiś punktami poprowadzone są dwie „równoległe” drogi to usuwana jest ta z większymi kosztami i czasami. Drogi są traktowane jako równoległe tylko jeśli zaczynają i kończą się w tym samym punkcie oraz w żadnej z nich nie występuje punkt poboru lub węzeł. Przykład równoległych dróg jest pokazany na poniższym rysunku, gdzie zostały one zaznaczone kolorem żółtym i zielonym.



Rysunek 3. Wizualizacja dróg równoległych w instalacji

- Jeżeli nie został osiągnięty punkt poboru, a rozwiązanie przebiega w jego sąsiedztwie, możemy dodać jedną krawędź tak, by dołączyć ten punkt poboru do rozwiązania. Następnie sprawdzamy, czy nastąpiła poprawa wartości funkcji celu i jeśli tak, zamieniamy te rozwiązania. Opisaną sytuację przedstawiono na poniższym rysunku. W zaznaczonym miejscu wystarczy dołączyć jedną krawędź w dół, aby przyłączyć punkt poboru do rozwiązania.

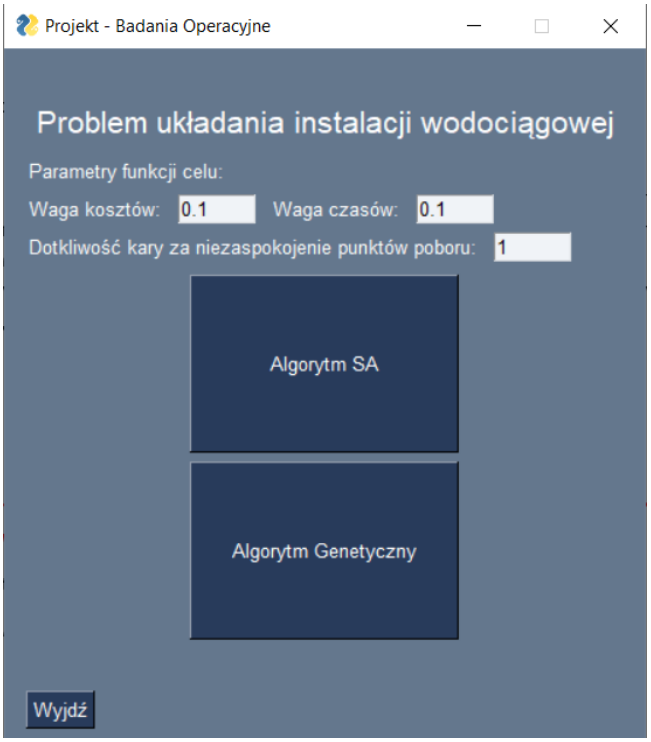
Rysunek 4. Wizualizacja nieosiągnięcia punktu poboru

- Usuwanie krawędzi o zerowym przepływie – jeśli w wyniku działania algorytmu, otrzymamy krawędzie o zerowym przepływie, mogą zostać one usunięte (tak, by otrzymać cały czas rozwiązanie dopuszczalne). Realizować to miała funkcja delete_edges w pliku SA. Niestety występują w niej błędy, których z braku czasu nie daliśmy rady naprawić.

4. Aplikacja

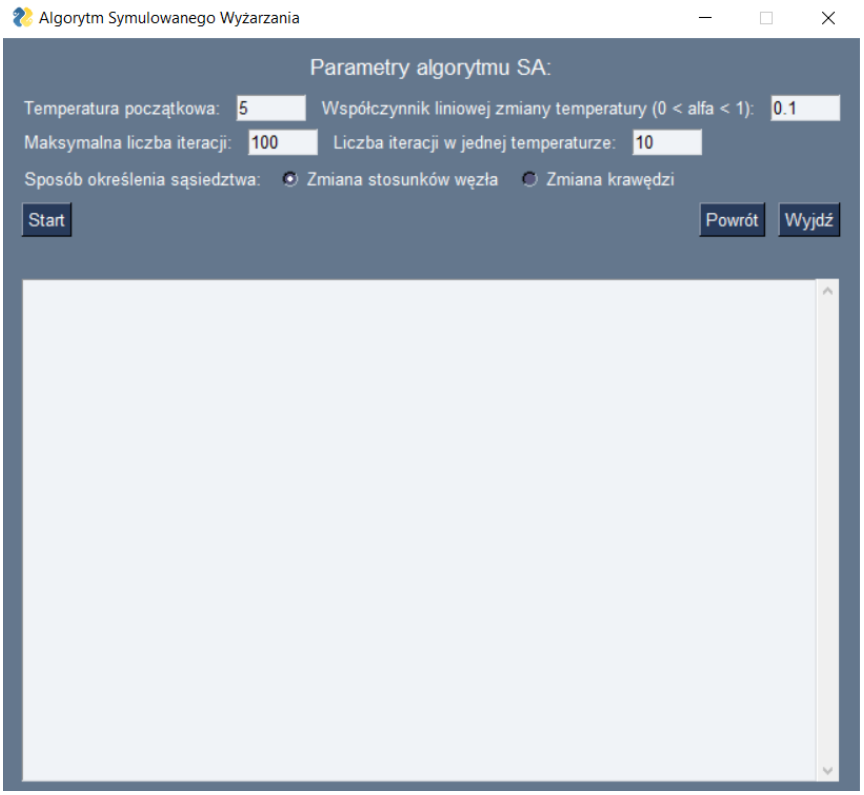
Pakiety potrzebne do uruchomienia kodu to PySimpleGUI, numpy i pandas.

Aby uruchomić program zawierający algorytmy, należy otworzyć plik main.py i uruchomić kod. Po uruchomieniu zostanie wyświetlone główne menu programu, w którym można zmienić parametry funkcji celu oraz wybrać algorytm. W przypadku algorytmu genetycznego dotkliwość kary za niezaspokojenie punktów poboru powinna być 100 razy większa niż pozostałe wagi, by algorytm działał prawidłowo.

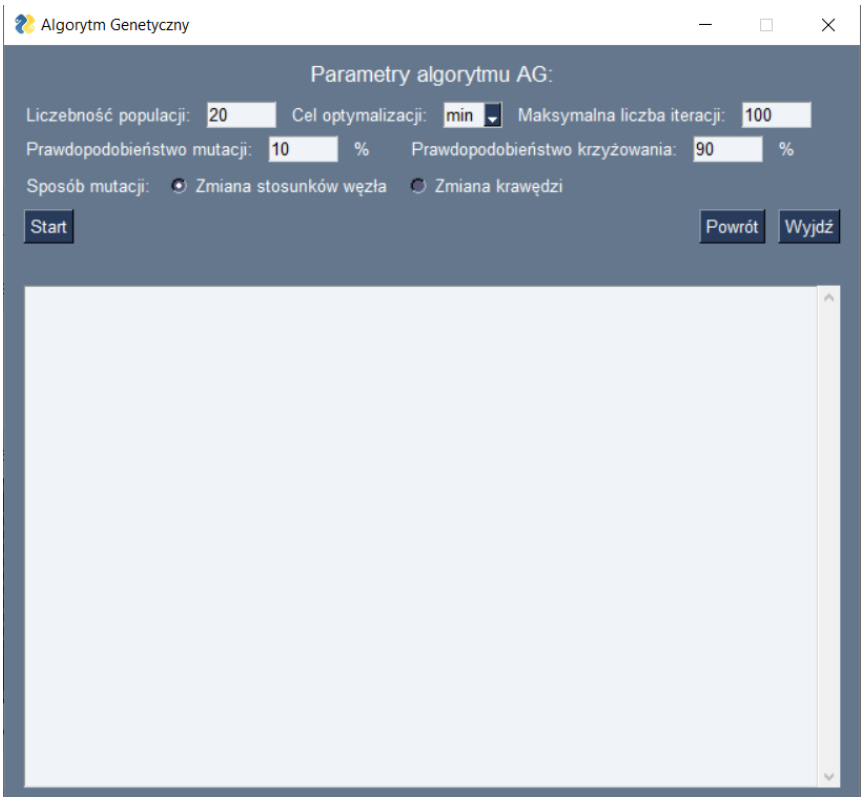


Rysunek 5. Główne okno GUI

Po wybraniu algorytmu, otwarte zostanie okno, w którym można wybrać parametry odpowiedniego algorytmu. Po wciśnięciu przycisku Start algorytm zacznie pracę, po czym wyświetli rezultaty w polu tekstowym poniżej. Istnieje też możliwość powrotu do głównego menu, dzięki czemu można sprawnie testować oba algorytmy bez konieczności uruchamiania od nowa kodu programu.



Rysunek 6. Okno algorytmu SA



Rysunek 7. Okno algorytmu genetycznego

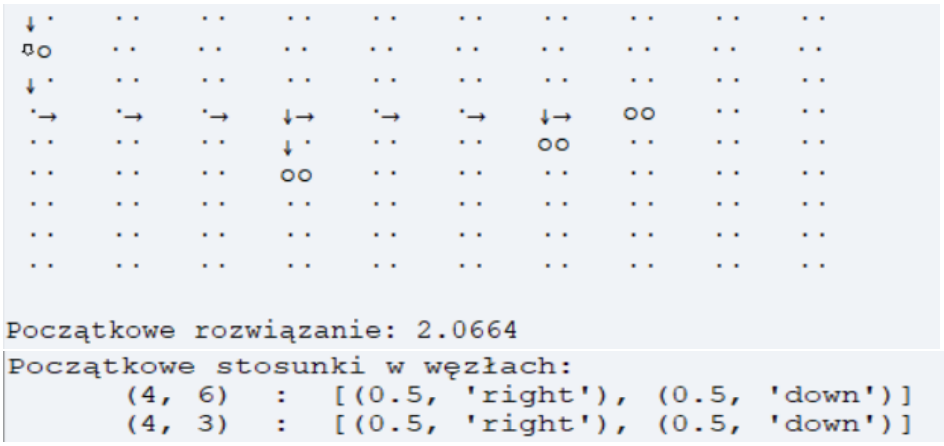
5. Eksperymenty

5.1. Eksperyment SA:

Test algorytmu SA wykonano dla parametrów:

- T_0 = [5, 10, 20]
- α = [0.1, 0.5, 0.9]
- N_{max} = [10, 100, 1000] – maksymalna ilość iteracji
- $n_{i,T}$ = [5, 10, 20] – ilość iteracji dla jednej wartości temperatury

I następującego rozwiązania początkowego:



Rysunek 8. Rozwiązanie początkowe dla SA

Sąsiedztwo w algorytmie było określone na 2 sposoby: jako zmiana stosunków w węźle lub jako zmiana krawędzi na 3 inne.

Wyniki dla pierwszego sposobu:

Wartość funkcji celu dla początkowego rozwiązania: 2.0664

Najlepsze rozwiązanie:
T_0 = 5, α = 0.1, N_{max} = 100, $n_{i,T}$ = 10 : f.celu = 1.9983, iteracja 2

Najgorsze rozwiązanie:
T_0 = 20, α = 0.5, N_{max} = 1000, $n_{i,T}$ = 5 : f.celu = 2.0664, iteracja 0

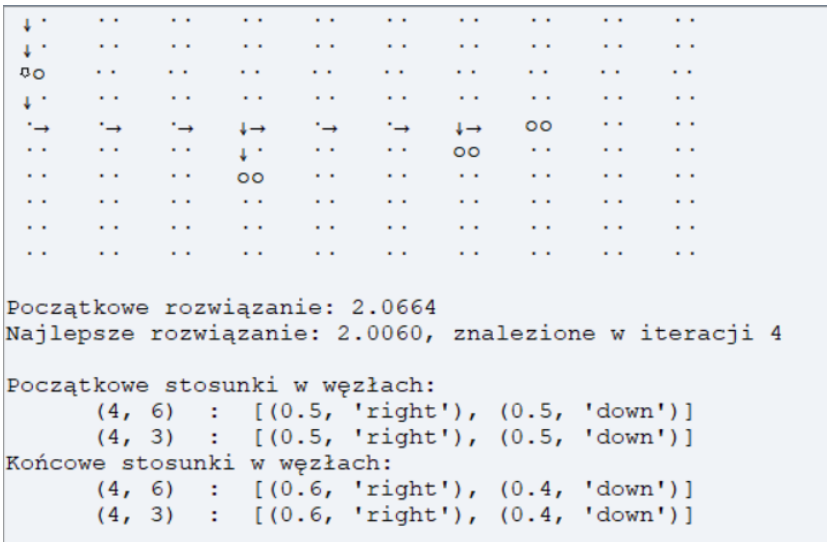
Zdecydowana większość parametrów dała rozwiązanie o wartości funkcji celu 2.0060. Maksymalna ilość iteracji nie miała zbyt dużego znaczenia, gdyż algorytm wykonywał się przez maksymalnie 48 iteracji – należałoby wobec tego przyjąć N_{max} = 100. Nie da się jednoznacznie określić optymalnych wartości innych parametrów, niemniej jednak większość rozwiązań startujących z T_0 = 5 charakteryzowała się niższą od innych wartością funkcji celu.

Średnia wartość funkcji celu wyniosła 2.0132 – jest to obniżenie wartości funkcji celu rozwiązania początkowego o 2.6%. Średnio algorytm kończył pracę po 5 iteracjach.

Edycja wyników dla pierwszego sposobu:

Spróbowaliśmy powtórzyć eksperyment SA, aby sprawdzić, dla jakich stosunków węzłów otrzymana została najlepsza wartość funkcji celu. Niestety, nie udawało się ponownie uzyskać tego wyniku. Zrobiliśmy więc przegląd zupełny rozwiązań i okazało się, że najlepsza wartość funkcji celu dla danego rozwiązania początkowego to 2.0060. Jest to też wartość najczęściej zwracana przez algorytm SA, dlatego wiemy (a przynajmniej mamy duże podstawy, by podejrzewać), że działa on w sposób poprawny. Wcześniejszy błędny wynik mógł wynikać z drobnych błędów implementacji, które występowały, gdy przeprowadzaliśmy poprzedni eksperyment. Byliśmy pewni, że błędy te nie wpływały na działanie algorytmu SA dla powyższego rozwiązania początkowego, ale jednak okazało się, że jest inaczej.

Najlepszy wynik występuje dla poniższych stosunków przepływów w węzłach:



```
Początkowe rozwiązanie: 2.0664
Najlepsze rozwiązanie: 2.0060, znalezione w iteracji 4

Początkowe stosunki w węzłach:
      (4, 6)   :  [(0.5, 'right'), (0.5, 'down')]
      (4, 3)   :  [(0.5, 'right'), (0.5, 'down')]
Końcowe stosunki w węzłach:
      (4, 6)   :  [(0.6, 'right'), (0.4, 'down')]
      (4, 3)   :  [(0.6, 'right'), (0.4, 'down')]
```

Rysunek 9. Stosunki w węzłach dla najlepszego rozwiązania

Wyniki dla drugiego sposobu:

Niestety, specyfika drugiego sposobu sąsiedztwa (dodawanie aż do 3 krawędzi i usunięcie jednej) powodowała, że funkcja celu, dla ustalonych danych, ulegała pogorszeniu za każdym razem. Wobec tego eksperyment nie został wykonany, a sposób został jednoznacznie uznany za niewłaściwy dla algorytmu SA. Sąsiedztwo to miałoby prawo działać dla bardziej zróżnicowanych kosztów i czasów kładzenia krawędzi.

5.2. Algorytm genetyczny

Przeprowadziliśmy testy algorytmu genetycznego dla różnych parametrów (dla każdego zestawu 100 prób):

- Wielkość populacji = [10, 20, 30]
- Prawdopodobieństwo mutacji = [10%, 70%]
- Prawdopodobieństwo krzyżowania = 90%
- Maksymalna ilość iteracji = 20. Nie przeprowadzaliśmy testów dla różnych wartości maksymalnej liczby iteracji, ponieważ zauważyliśmy, że algorytm znajduje najlepsze rozwiązanie zaraz na początku i zazwyczaj jest to przed 20 iteracją.
- Dwóch różnych sposobów mutacji: przez zmianę stosunków w węźle lub przez zamianę krawędzi na trzy inne.
- Dwóch różnych zestawów parametrów funkcji celu.

Szczegółowe wyniki testów zebrane zostały w folderze EA_tests.

Podczas testów zauważyliśmy, że aby uzyskiwać sensowne rozwiązania w funkcji celu należy ustawić dotkliwość kary za niedostarczenie wody ok. 100 razy większą niż waga czasów i kosztów położenia instalacji. W przeciwnym przypadku algorytm często uznawał, że najbardziej opłaca się po prostu nie kłaść instalacji. Zaobserwowaliśmy też, że w wyniku

krzyżowania często powstają rodzice (identyczne rozwiązania jak wejściowe). Jednak mimo to algorytm zwraca dosyć sensowne wyniki.

Wyniki dla bazowego zestawu parametrów funkcji celu (waga czasów = 0.1, waga kosztów = 0.1, kara za niedostarczenie wody = 10):

Wartości najlepszych otrzymanych wartości wahają się od 1.873 do 1.961. Nie są to bardzo duże różnice.

Najlepsza wartość funkcji celu została otrzymana dla wielkości populacji = 20, prawdopodobieństwa mutacji = 70% i mutacji przez zmianę stosunków w węzłach, zaś najgorsza dla wielkości populacji = 20, prawdopodobieństwa mutacji = 70% i mutacji przez zmianę krawędzi. Widzimy więc, że w naszym przypadku mutacja mogła mieć duży wpływ na jakość wyników.

Obserwacje dla wyników, gdy prawdopodobieństwo mutacji = 10% i bazowego zestawu parametrów:

Zazwyczaj lepsze wyniki dawało zastosowanie mutacji przez zmianę stosunków w węzłach. Średnio lepsze wartości uzyskujemy dla większej populacji, występują tam też mniejsze odchylenia standardowe wartości funkcji celu. Prawdopodobnie jest to spowodowane tym, że duży wpływ na wynik algorytmu mają wygenerowane rozwiązania początkowe, więc im jest ich więcej, tym większa szansa na znalezienie wśród nich dobrego.

6. Podsumowanie

Nasz projekt jest dosyć złożony implementacyjnie i właśnie implementacja rodziła największe problemy. Zajęła nam ona sporo czasu i dlatego nie zdążyliśmy między innymi zrealizować różnych ulepszeń w algorytmie genetycznym. Aby rozwinąć nasz projekt, można właśnie zaimplementować i przetestować te ulepszenia. Można też spróbować zastosować inne sposoby selekcji lub zmienić krzyżowanie tak, by rzadziej zwracało rodziców.

Wyniki działania obu algorytmów zależą w dużej części od wygenerowanego rozwiązania początkowego, jednak ciężko jest zmienić sposób jego tworzenia, jeśli chcemy (w przypadku algorytmu genetycznego) otrzymać ich populację. Sposobem na poradzenie sobie z tym jest generowanie dużej populacji w przypadku algorytmu genetycznego. Dla algorytmu symulowanego wyżarzania możemy natomiast wygenerować kilka rozwiązań początkowych i wybrać najlepsze do dalszej edycji.

7. Literatura

- Wykłady i materiały z zajęć

8. Podział pracy:

Etap	Joanna Nużka	Marcin Pilarski	Kacper Szmajdel
Zdefiniowanie problemu, opis ograniczeń	30% - Zdefiniowanie ograniczeń	30% - Zdefiniowanie ograniczeń	40%: - Wymyślenie problemu

			- Zdefiniowanie ograniczeń
Zdefiniowanie danych do problemu, wczytywanie danych	0%	50% - Zdefiniowanie danych	50% - Wczytywanie danych z pliku
Implementacja modelu problemu (spora część metod była modyfikowana i pisana przez kilka osób)	50% - Klasa Point: metoda copy - Klasa Solution: metody copy, update_flow, add_knot_out, add_knot_in_out, check_cycle, check_if_edge_correct, add_edge, debugging wielu z tych funkcji	35% - Klasy Point i ExtractionPoint - Klasa Solution: metody update_extraction_point, update_flow, add_knot_out, add_knot_in, check_if_edge_correct - funkcja objective_function	15% - Klasa Point - Klasa Solution: metody check_if_edge_correct, check_knots_distances, return_solution, return_knots_proportions
Konstrukcja rozwiązań początkowych	40% - rozwiązanie początkowe z węzłami (initial solution with knots) - losowe rozwiązanie początkowe - poprawki	0%	60% - rozwiązanie początkowe bez węzłów (initial solution) - losowe rozwiązanie początkowe
Algorytm SA - implementacja	70% - Sąsiedztwo: change_random_knot - Algorytm SA	30% - Sąsiedztwo: change_random_edge	0%
Algorytm genetyczny	45% - Operator krzyżowania, funkcje crossing() i mutation()	20% - Funkcja delete_edges	35% - Selekcja - Ogólny schemat algorytmu
GUI i wizualizacja rozwiązań	0%	65% - GUI	35% - funkcje do wizualizacji rozwiązań
Testy algorytmów	15% - Testy algorytmu SA - poprawka	30% - Testy algorytmu SA	55% - Testy algorytmu genetycznego
Dokumentacja	60% - Rozdziały 3 (pseudokod SA, szczegóły i pseudokod alg. genetycznego), 5.1 (edycja SA), 5.2, 6	40% - Rozdziały 1, 2, 3 (opis SA, ogólny opis alg. genetycznego), 4, 5.1	0%