Kamil Jaworski
Piotr Jastrzębski

Dokumentacja końcowa projektu ALHE
Łamigłówka: **Mosty**
Metoda: **Algorytm ewolucyjny**

1. **Opis problemu**

Mosty to łamigłówka logiczna rozgrywana na prostokątnej planszy (tablicy N na M pól). Na niektórych polach umieszczone są wyspy. Każda wyspa posiada liczbę ją charakteryzującą (od 1 do 8). Dwie wyspy mogą być połączone mostem jeśli mają tę samą współrzędną rzędną lub odciętą. Dana para wysp może być połączona jednym, dwoma lub nie być połączona żadnym mostem. Żadne dwa mosty nie mogą się przecinać. Z każdej wyspy powinna wychodzić liczba mostów zgodna z liczbą ją charakteryzującą wyspę. Celem gry jest znalezienie takiej struktury połączeń mostowych, która spełnia wszystkie powyższe założenia oraz stanowi graf spójny (tj. zakładając iż most jest krawędzią nieskierowaną, z każdej wyspy można dojść do każdej innej).

2. **Model**

Zgodnie z założeniem, do rozwiązania problemu mieliśmy użyć algorytmu ewolucyjnego. Po wstępnej analizie problemu wybraliśmy sposób reprezentacji (zamodelowania) problemu, który nadaje się do użycia w połączeniu z metaheurystykami ewolucyjnymi. Wybraną formą reprezentacji został **wektor mostów**.

Wektor mostów to prosty wektor liczb całkowitych, z których każda należy do zbioru **{0,1,2}**.I-ta liczba reprezentuje krotność i-tego połączenia mostowego („krotność mostu”) pomiędzy pewnymi dwoma wyspami.

Dla każdej pozycji z wektora mostów musimy dysponować informacjami nt. wysp, które on łączy. Stąd docelowa reprezentacja składa się z dwóch macierzy:

- macierz wysp
- macierz mostów

Macierz wysp to macierz o **N rzędach i 3 kolumnach**, gdzie **N** jest liczbą wysp w zestawie wejściowym. Kolejne kolumny reprezentują: współrzędne **x,y** oraz liczba charakteryzująca wyspę, dalej zwana „**wartością wyspy**”.

Macierz mostów to macierz o **M rzędach i 3 kolumnach**, gdzie **M** jest liczbą wszystkich możliwych do zbudowania mostów w zadanym zestawie wejściowym. Jest budowana na podstawie macierzy wysp. Kolejne kolumny reprezentują: indeks wyspy początkowej oraz końcowej (w macierzy wysp; wysp połączonych danym mostem), a następnie krotność danego połączenia mostowego.

Zauważyć można, iż dzięki takiemu podejściu, opisywany na początku **wektor mostów** to trzecia kolumna **macierzy mostów**. Może on pełnić rolę osobnika optymizowanego za pomocą tworzonej heurystyki.

Po przetestowaniu wielu kandydatów, najlepszą funkcją oceny okazała się prosta suma trzech składników:

- współczynnika braku spójności (+1 dla każdej pary wysp, które są z siebie wzajemnie nieosiągalne)
- liczby mostów przecinających się
- sumy różnić pomiędzy faktyczną liczbą mostów odchodzących od danej wyspy, a wartością wyspy (po wszystkich wyspach)

Jak widać, koszt równy zero determinuje poprawne rozwiązanie problemu.

3. Dane wejściowe

Nasze programy obsługują wejście w formacie tekstowym. Informacje o wyspach powinny być dostarczane w następującym formacie:

```
x y val
1 1 3
2 1 2
1 2 2
2 2 1
```

4. Pierwszy etap ewolucji ewolucji, czyli pierwsze rozwiązanie

Pierwszym algorytmem, który został zaimplementowany, był, (zgodnie z założeniem z dokumentacji wstępnej) algorytm ewolucyjny **1+1**.

Algorytm **1+1** polega na zachowywaniu dotychczas najlepszego znalezionego osobnika, mutowaniu go, a w przypadku gdy osobnik zmutowany okaże się lepszy od oryginału, podmiana dotychczas najlepszego osobnika. Oprócz tego występuje adaptacja stopnia mutacji, która zostanie opisana danej.

Algorytm ewolucyjny **1+1** charakteryzuje się brakiem krzyżowania. Może się to wydawać zaprzeczeniem idei algorytmów ewolucyjnych oraz metafory, która stoi za tą rodziną metod heurystycznych, niemniej jest w pełni uzasadnioną decyzją projektową w określonych przypadkach.

Operacja krzyżowania z założenia ma umożliwiać takie połączenie dwóch (lub więcej) osobników, by osobnik wynikowy nosił znamiona jak największej liczby pozytywnych cech swych rodziców. Czasem jednak (jak w przypadku problemu „Mosty") operacja krzyżowania jest bardzo trudna, jeśli nie niemożliwa do wykonania w zgodności ze swą naturą – skrzyżowanie dwóch wartościowych osobników (rozwiązań problemu „Mosty") niemal zawsze poskutkuje stworzeniem osobnika (dużo) mniej wartościowego.

Skupienie się na operacji mutacji niesie za sobą zagrożenia jak i korzyści. Oczywistym zagrożeniem jest możliwość ograniczenia przestrzeni możliwych do osiągnięcia rozwiązań już na etapie losowania pierwszego osobnika. Z korzyści – możliwa jest dokładna analiza skuteczności operacji mutacji i optymalizacja szybkości uzyskiwania lepszych wyników. Jest to możliwe za sprawą adaptacji stopnia mutacji. Okazuje się, iż **odsetek sukcesów** (prób mutacji zakończonych znalezieniem lepszego osobnika od oryginału) powinien w idealnym przypadku wynosić **20% [1].** Możemy więc zliczać sukcesy i co określoną liczbę prób dostosowywać stopień mutacji w ten sposób, by dążyć do zadanej skuteczności. Dzięki temu szybkość uzyskiwania poprawy wyniku będzie w ogólnym przypadku optymalna.

Rozwiązanie oparte na powyższej idei zostało zaimplementowane jako metaheurystyka w pliku **evol.r**. Stworzono również dwie instancje tej metaheurystyki – rozwiązujące problem sortowania oraz „mosty" – odpowiednio w plikach **evol.sort.r** oraz **evol.bridges.r**

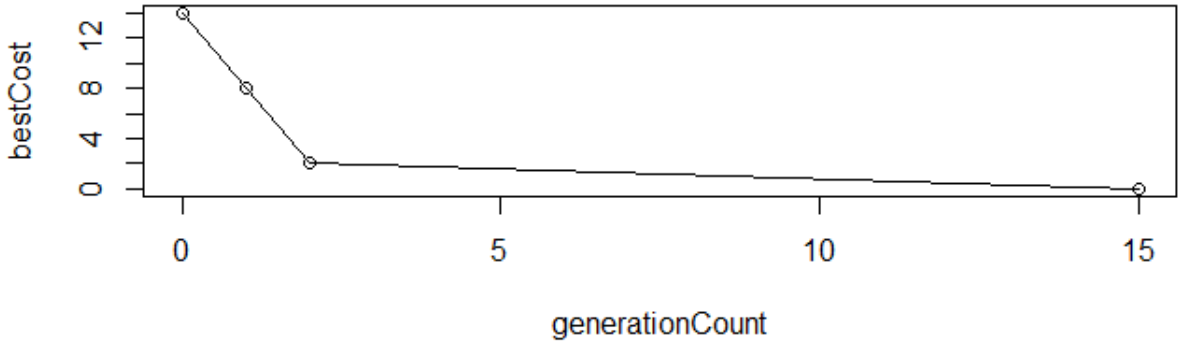
Poprzez podanie parametrów wejściowych (nadpisanie funkcji **evol.mutationLevelTuneSettings**), możliwe jest dostosowywanie działania algorytmu. Metody, które należy nadpisać w celu instancjacji tej metaheurystyki, znajdują się na początku pliku.

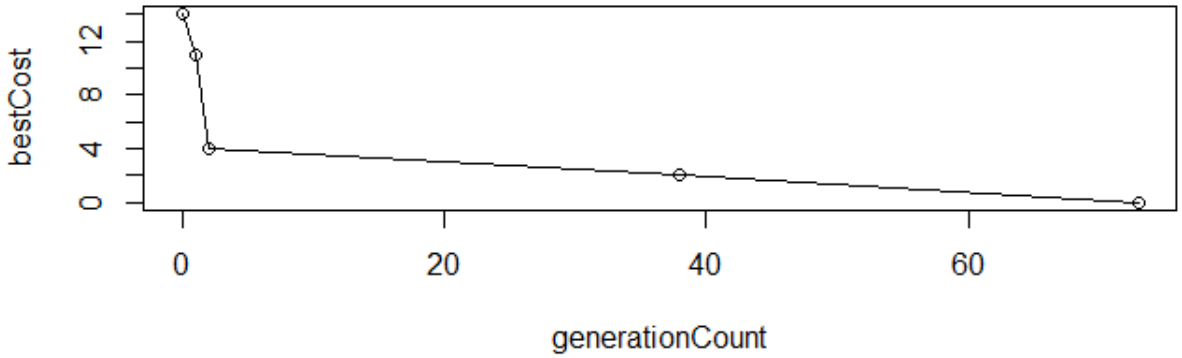
5. Testy metody 1+1 (mosty)

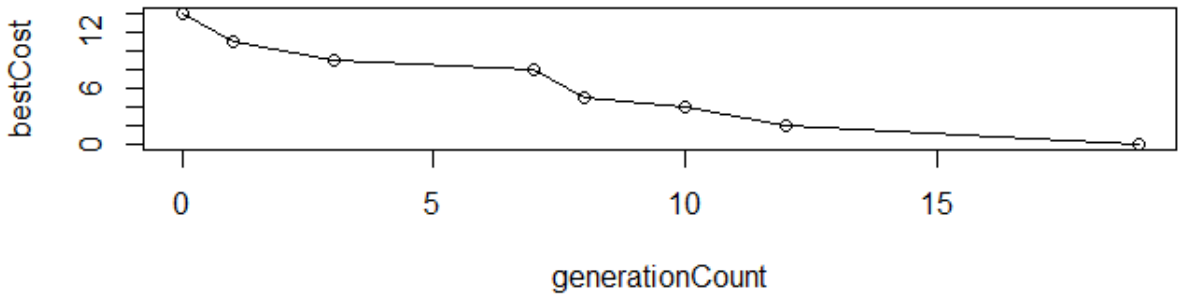
Na kolejnych stronach zaprezentujemy wykresy z kilkukrotnego wykonania programu dla różnych plików wejściowych. Informacje o ustawieniach są podawane w formacie:

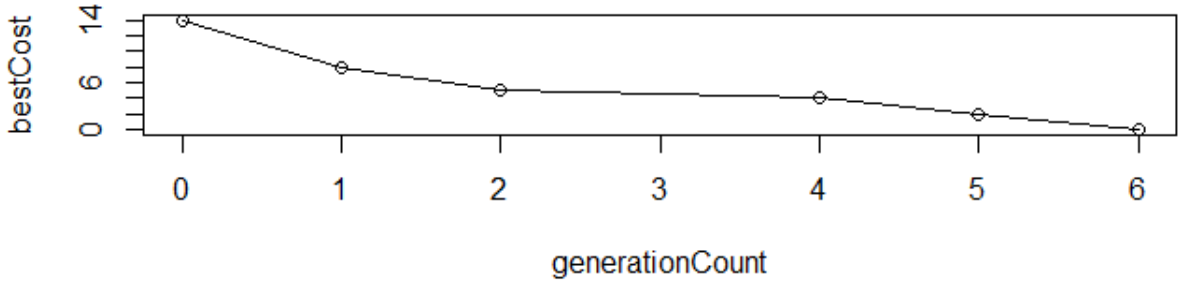
(okres adaptacji mutacji, minimalna mutacja, maksymalna mutacja)

1.txt (10,2,5)

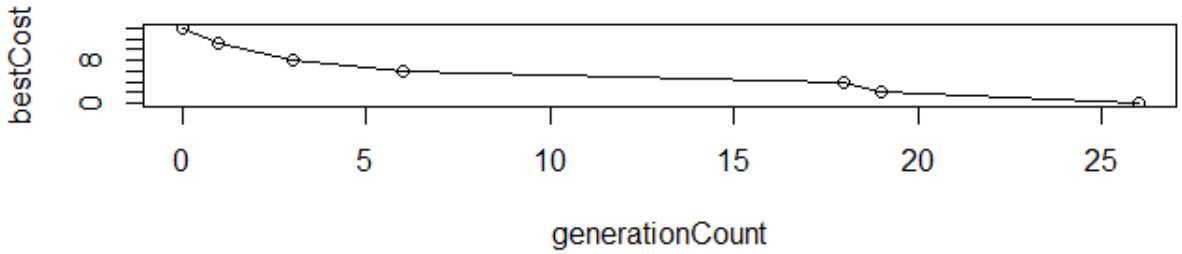

bestCost
0
4
8
12
0
5
10
15
generationCount


bestCost
0
4
8
12
0
20
40
60
generationCount

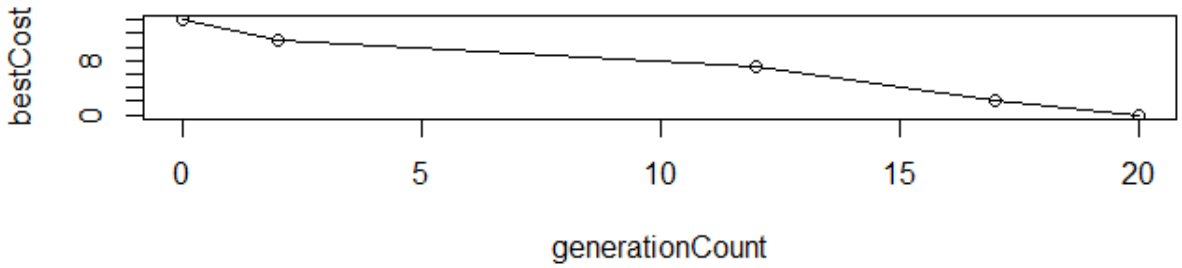

bestCost
0
6
12
0
5
10
15
generationCount


bestCost
0
6
14
0
1
2
3
4
5
6
generationCount

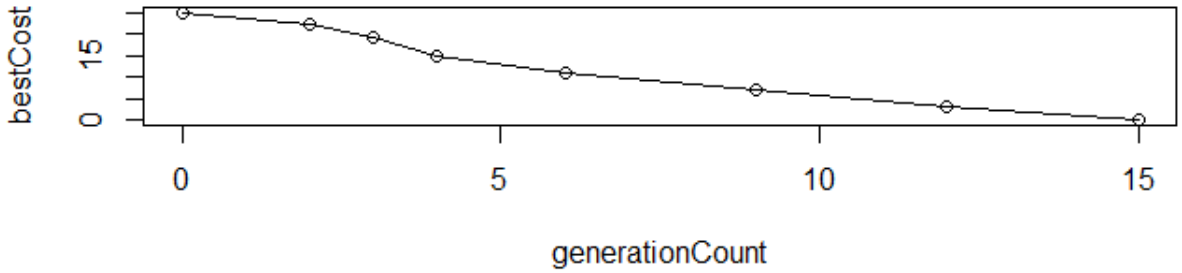
1.txt (10,2,2)



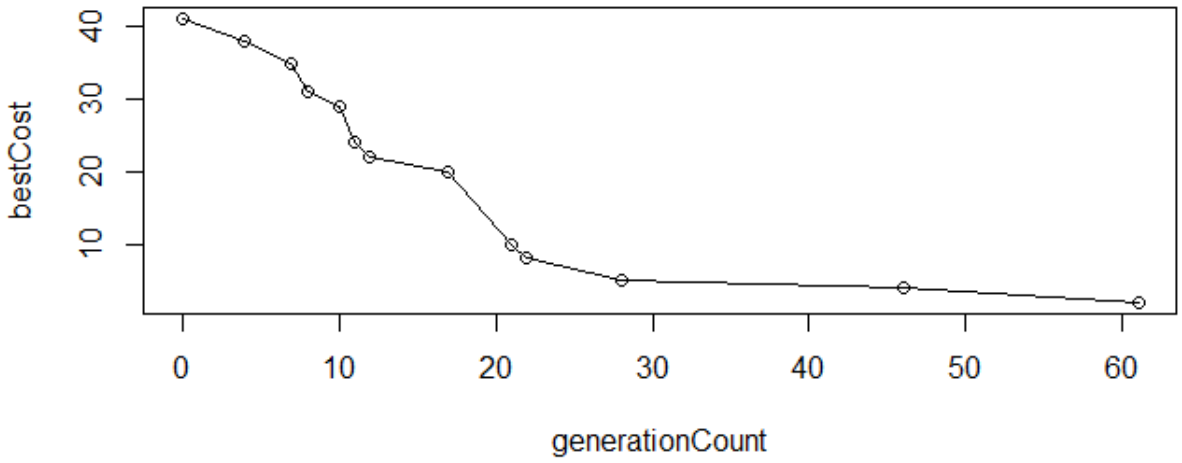
2.txt (10,2,2)



3.txt (10,2,2)

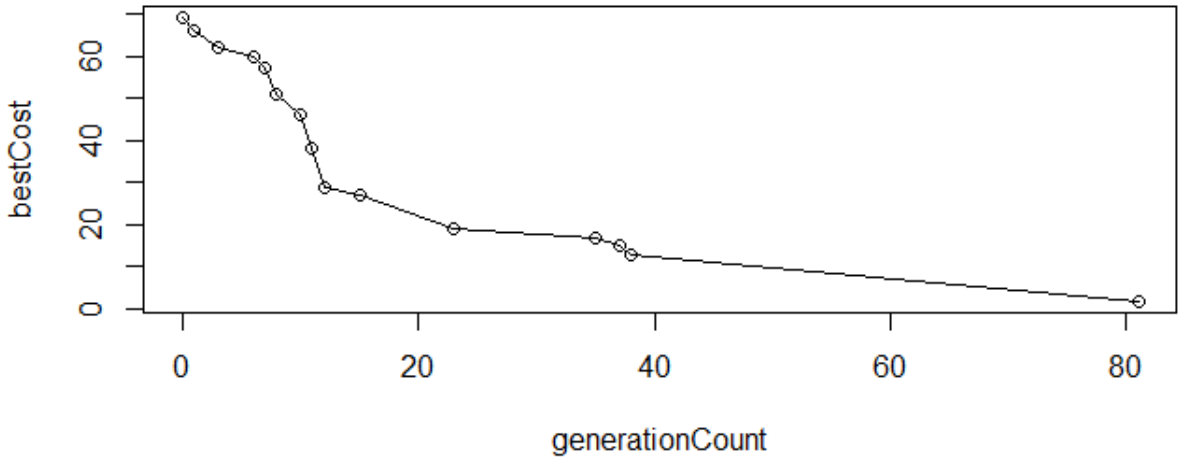


4.txt (10,2,2)



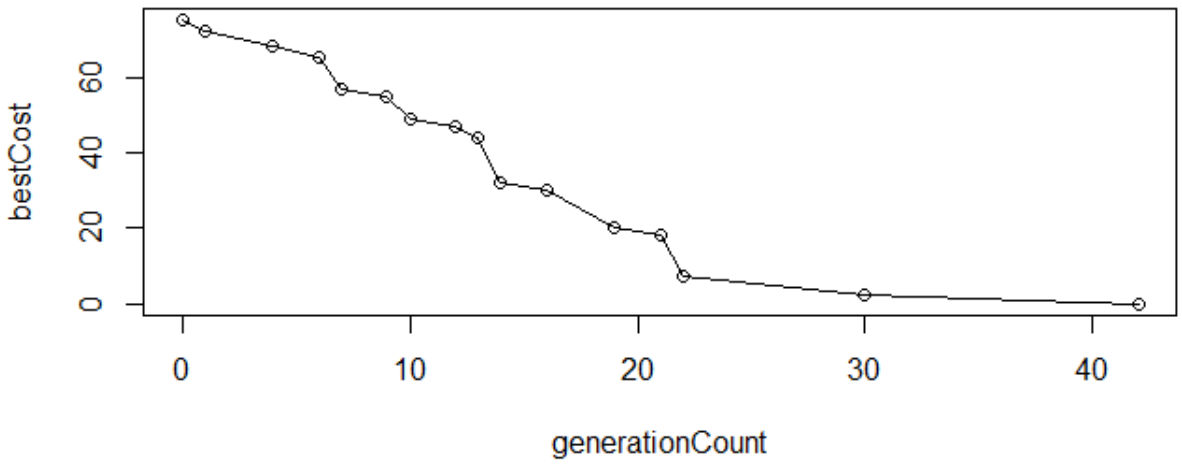
Powyżej widać problemy algorytmu **1+1** z przejściem przez maksimum lokalne. Jak widać, szybko dochodzimy do rozwiązania taniego, niemniej po jego znalezieniu proces poszukiwań zachodzi w ślepą uliczkę i nie znajduje optymalnego rozwiązania. Problemy tego typu skłoniły nas do pracy nad lepszą wersją algorytmu przeszukiwania przestrzeni rozwiązań.

5.txt(10,2,2)



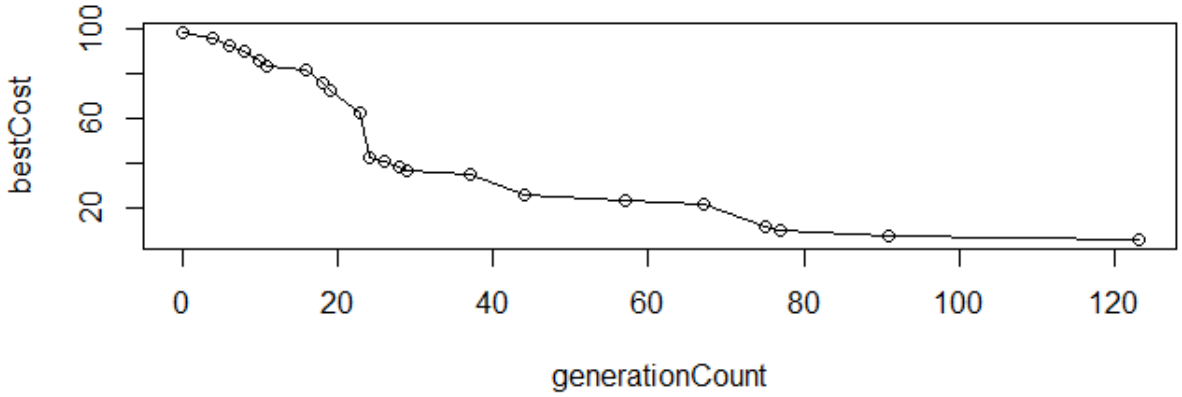
Sytuacja analogiczna do poprzedniej.

6.txt (10,2,2)



Nie wszystkie większe testy są skazane na porażkę. Jak widać, funkcja celu jest stosunkowo dobrze dobrana, gdyż pomimo dużej skłonności algorytmu 1+1 do wpadania w maksimum lokalne, maksimum globalne jest odnajdowane w stosunkowo wielu przypadkach. Między innymi test **6.txt** jest rozwiązywany przez za pomocą metody 1+1 bez większych problemów.

7.txt (10,2,5)



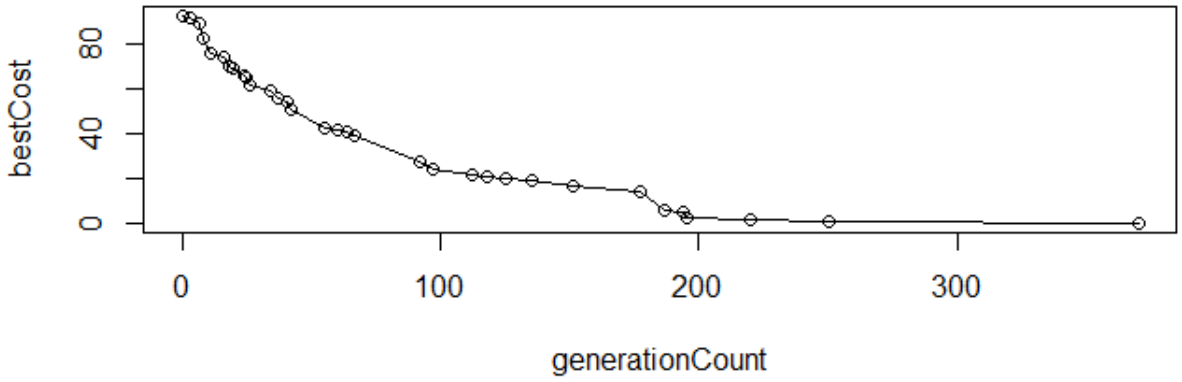
Wnioski:

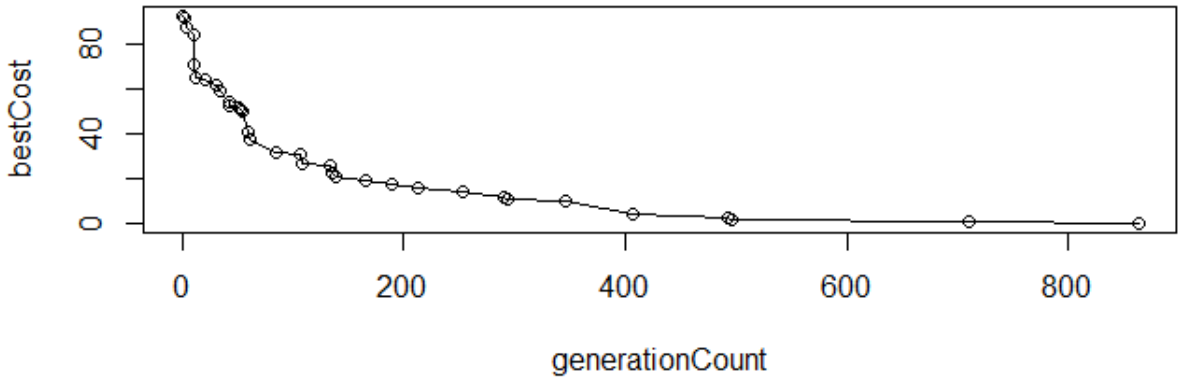
Metoda 1+1 ma dużą skłonność do zatrzymywania się w maksimum lokalnym funkcji celu. Z tego powodu jej przydatność jest ograniczona w sytuacjach, gdy funkcja celu posiada wiele maksimów lokalnych.

6. **Testy metody 1+1 (sortowanie)**

Pomimo porażki, którą metoda 1+1 ponosi podczas rozwiązywania problemu mosty, udowodnimy iż może być ona przydatna podczas rozwiązywania problemów innej klasy. Problem sortowania ma tę pożądaną własność, iż da się dla niego łatwo skonstruować wypukłą funkcję celu.

Test sortowania dla 26 liczb





Ten sam test dla ustawień (1,1,1) – u góry, oraz (10,1,40) – na dole.

Jak widać – dzięki adaptacji stopnia mutacji szybkość poprawy wyniku jest wyraźnie lepsza (ustawienia (1,1,1) sprowadzają algorytm to postaci bez adaptacji)

7. **Drugi etap ewolucji ewolucji, czyli drugie rozwiązanie**

Napotkawszy problemy z zatrzymywaniem się procesu wyszukiwania na maksimum lokalnym funkcji celu, zaimplementowaliśmy algorytm ewolucyjny nieco bardziej przypominający typowe podejście do problemu ewolucji w dziedzinie metod heurystycznych.

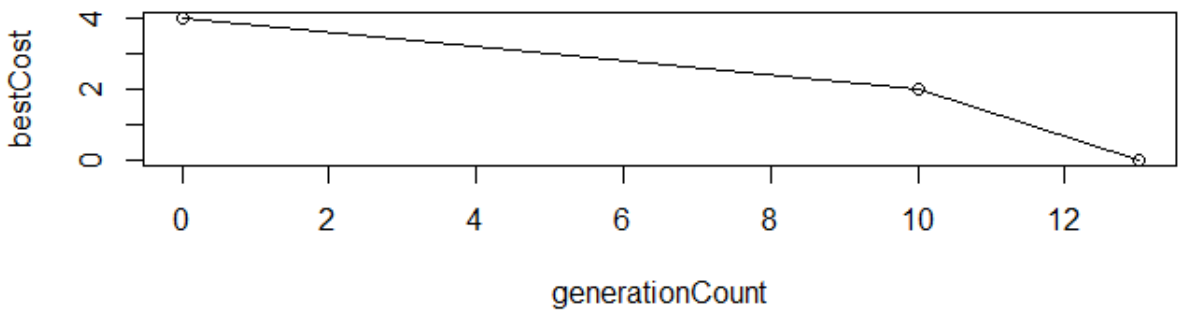
Plik **evol2.r** zawiera implementację metaheurystyki ewolucyjnego wyszukiwania. Podobnie jak w przypadku rozwiązania pierwszego, metody niezbędne do przykrycia w celu użycia tej metaheurystyki znajdują się na początku pliku.

Ze względu na to, iż problem sortowania byłby zaimplementowany w sposób analogiczny do instancjacji rozwiązania pierwszego, drugie rozwiązanie ma tylko jedną instancję – rozwiązywanie problemu „Mosty” (plik **evol2.bridges.r**)

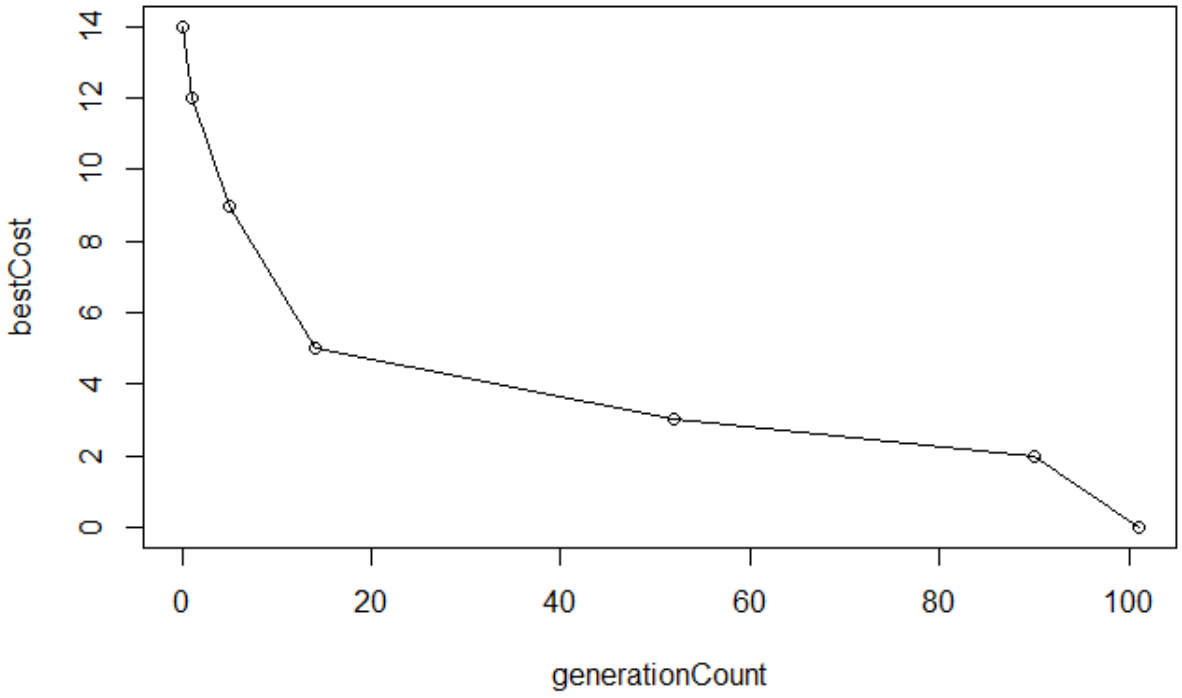
Możliwe ustawienia:

- liczność pokolenia (selekcja)
- liczba potomków zmutowanych (generacja)
- liczba potomków skrzyżowanych (generacja)

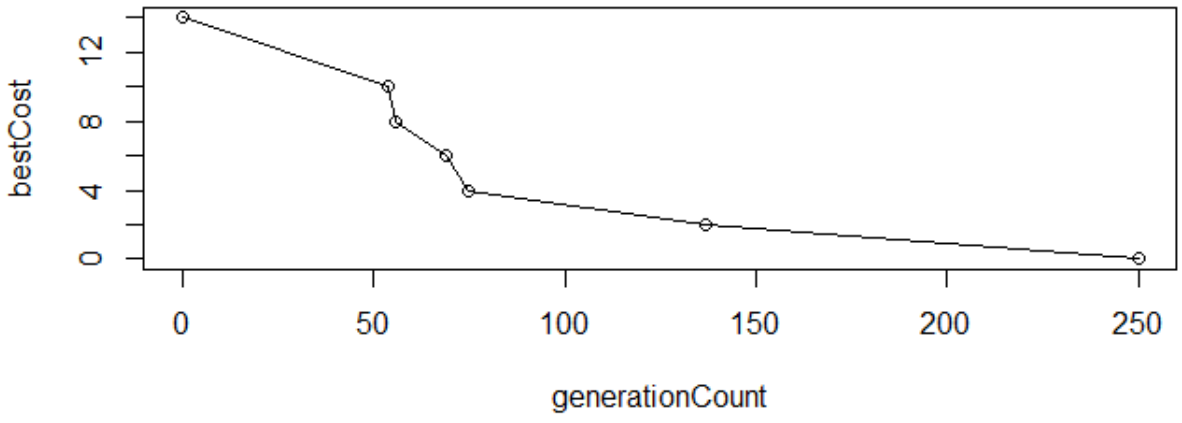
1.txt (2,2,10)



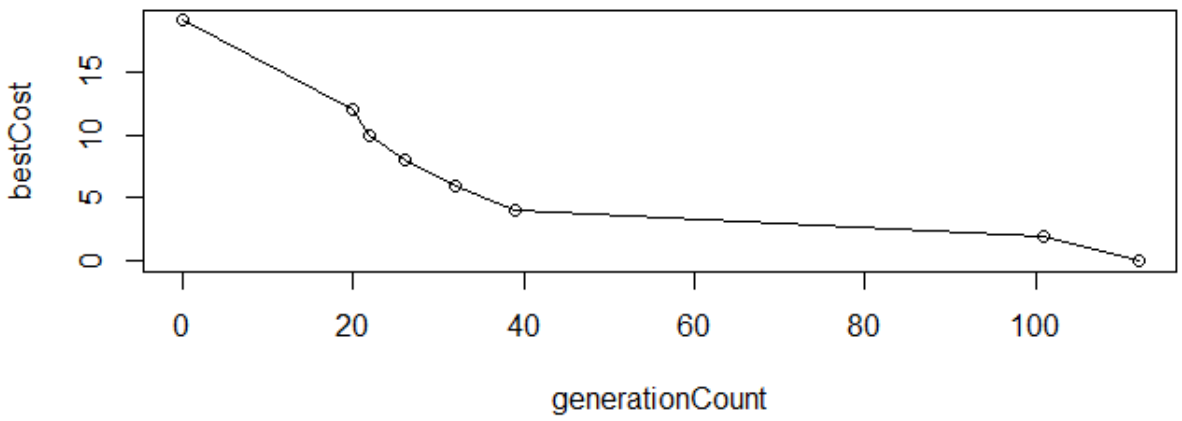
3.txt (2,2,10)



4.txt (2,2,10)

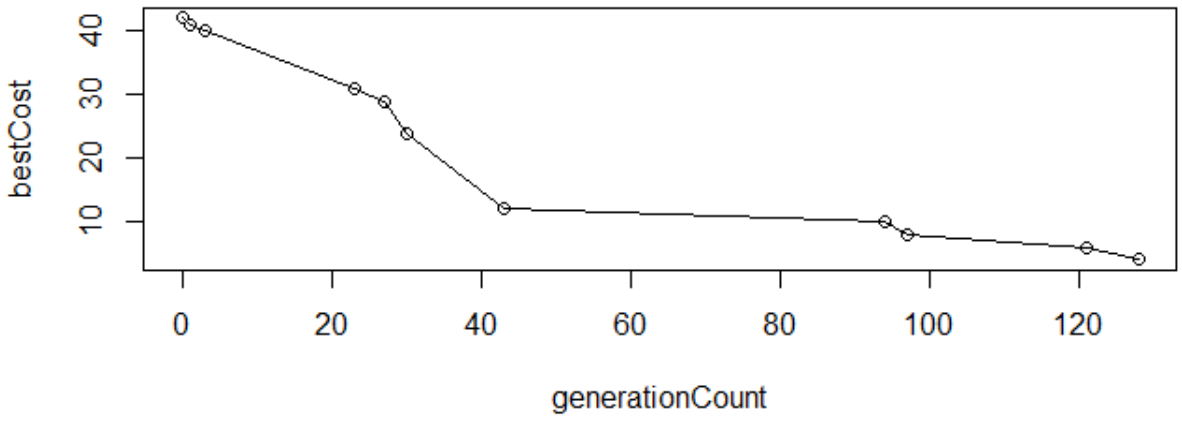


5.txt (2,2,10)



Jak widać, po wprowadzeniu krzyżowania algorytm znajduje rozwiązania dla zestawów, dla których wcześniej nie był w stanie udzielić poprawnej (nieprzybliżonej) odpowiedzi. Nie oznacza to jednak, że tak zmodyfikowane wyszukiwanie rozwiązuje wszystkie problemy. Dla większych danych wejściowych również zmodyfikowana wersja znajduje jedynie przybliżone rozwiązanie.

10.txt (2,2,10)



Problem wyszukiwania igły w stogu siana – nierozwiązywalny lub bardzo trudny do rozwiązania podejściem ewolucyjnym.

Powyższe ustawienia wyszukiwania – (2,2,10) – zostały dobrane eksperymentalnie. Zwiększenie udziału krzyżowania powoduje istotne spowolnienie lub uniemożliwienie znalezienia poprawnego wyniku (ze względu na wcześniej opisywane problemy z krzyżowaniem osobników – rozwiązań problemu „Mosty").

8. **Wnioski**

Algorytmy ewolucyjne nie są najlepszym wyborem, gdy zachodzi konieczność znalezienia jednego z niewielu poprawnych rozwiązań. Ich głównym zastosowaniem powinna być optymalizacja, tj. znajdowanie kosztu przybliżonego do globalnie najniższego.

Dobór metody do naszego zadania był więc nieco niefortunny. Metody ewolucyjne nie są zupełnie bezsilne wobec tego typu zadań, niemniej nie są one najlepszym wyborem.

Wyjątkiem jest tu metoda **1+1**, w przypadku której zachodzi jednak silne wskazanie posiadania wypukłej funkcji celu. Jeśli dla rozwiązywanego problemu da się znaleźć wypukłą funkcję celu, metoda **1+1** zapewni szybkie wyszukiwanie coraz lepszych rozwiązań.

[1] „*Podstawy Sztucznej Inteligencji, Preskrypt wykładu"*, Paweł Wawrzyński, str. 63