Metaheurystyki i Obliczenia Inspirowane Biologicznie

Implementacja local search dla QAP

Tomasz Kasperek 116393, Jacek Kubiak 116307

14 listopada 2018

1 Wstęp

Problemy z dziedziny optymalizacji kombinatorycznej często charakteryzują się wysoką złożnością obliczeniową. Dla większych instancji problemów, znalezienie optymalnego rozwiązania w racjonalnym czasie jest niemożliwe, tym bardziej, gdy poszukiwania są wykonywane na sprzęcie o skromnej mocy obliczeniowej.

W rzeczywistości rozwiązania nieoptymalne, a nadal bardzo dobre, są wystarczające dla wielu zastosowań. Uzyskujemy je przy pomocy heurystyk takich jak np. przeszukiwanie lokalne. W sprawozdaniu przedstawiono porównanie algorytmu przeszukiwania lokalnego w wersji greedy i steepest, random search oraz naszej autorskiej heurystyki na problemie kwadratowego przydziału (QAP).

1.1 Instancje

Testowanie algorytmów odbyło się na zbiorze 8. instancji: **chr18a**, **chr20a**, **tai12a**, **tai12b**, **tai15a**, **tai35b**, **esc32g**, oraz **lipa50a**. Rozmiary tych instancji zostały dobrane w taki sposób, aby obliczenia nie zajmowały zbyt dużo czasu, a jednocześnie można było wyciągnąć z nich ciekawe wnioski.

1.2 Algorytm heurystyczny

Ideą naszego algorytmu heurystycznego jest znalezienie elementu w permutacji który w połączeniu z pozostałymi elementami odpowiada za największą część kosztu całkowitego, a później na znalezieniu najlepszego zastępstwa dla niego. Algorytm ten ograniczony jest odcięciem, które następuje po 1000. iteracji. Ten krok jest potrzebny, ponieważ w przeciwnym przypadku, algorytm może działać w nieskończoność.

2 Sąsiedztwo

Operator sąsiedztwa zastosowany w algorytmie local search (w wersji „greedy" oraz „steepest") to wybranie 2. elementów w rozwiązaniu oraz zamiana ich kolejności. W przykładzie poniżej pokazano zamianę elementów w permutacji, a konkretnie elementu 3 oraz 7.

$$1 - 2 - \mathbf{3} - 4 - 5 - 6 - \mathbf{7} - 8 - 9$$

$$1 - 2 - \mathbf{7} - 4 - 5 - 6 - \mathbf{3} - 8 - 9$$

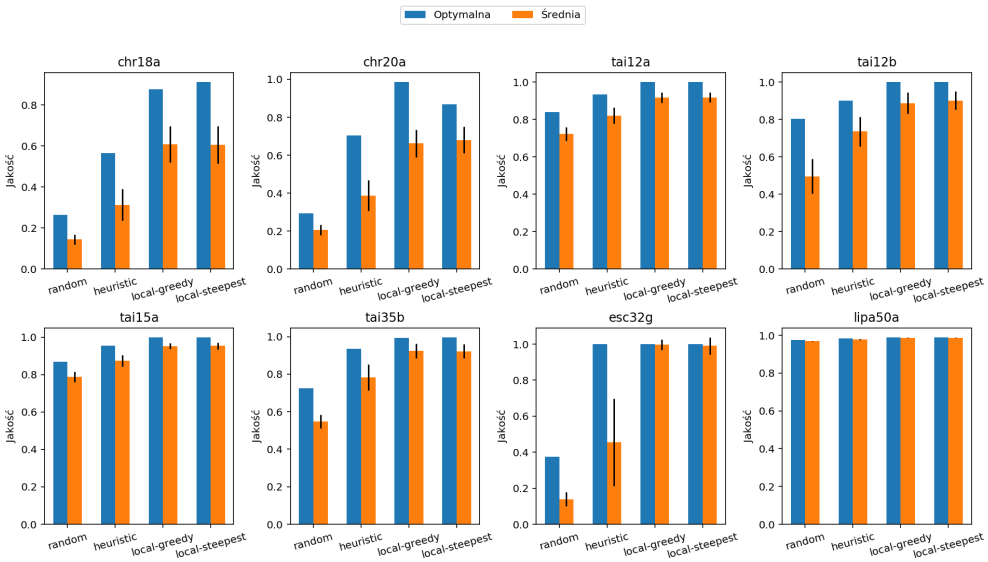
3 Porównanie 4 algorytmów

3.1 Jakość

Jakość zdefiniowana jest jako stosunek kosztu optymalnego rozwiązania do kosztu rozwiązania znalezionego przez algorytm. Tę zależność można zdefinować za pomocą wzoru:

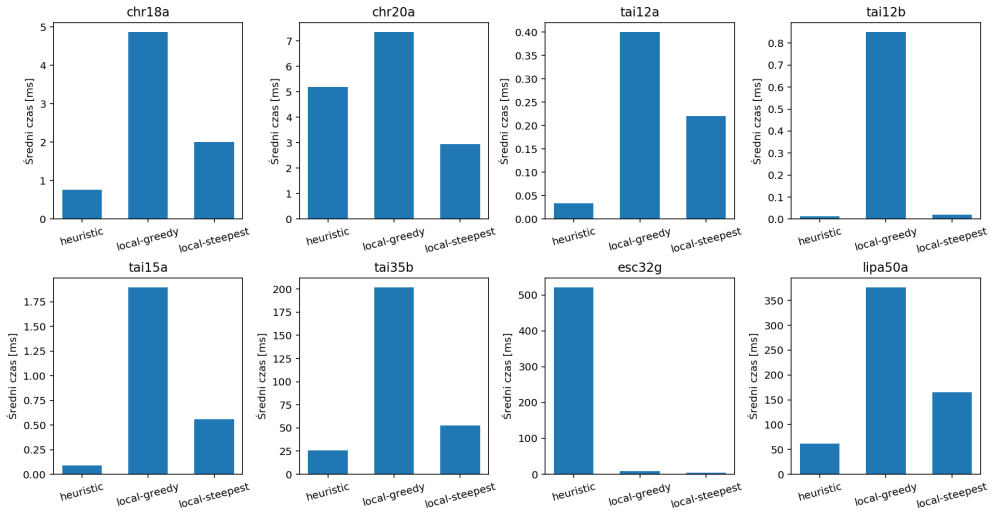
$$\text{quality} = \frac{\text{foundSolution}}{\text{optimalSolution}} \tag{1}$$

Wyniki eksperymentów przedstawiają się następująco:



Najgorszym algorytmem jest algorytm losowy, którego optymalna jakość nie przekraczała często nawet 0,5. Algorytm heurystyczny działa nieco lepiej, natomiast algorytmy local search (greedy oraz steepest) wyróżniają się jeszcze lepszą jakością. Algorytmy te, często docierały do rozwiązania optymalnego.

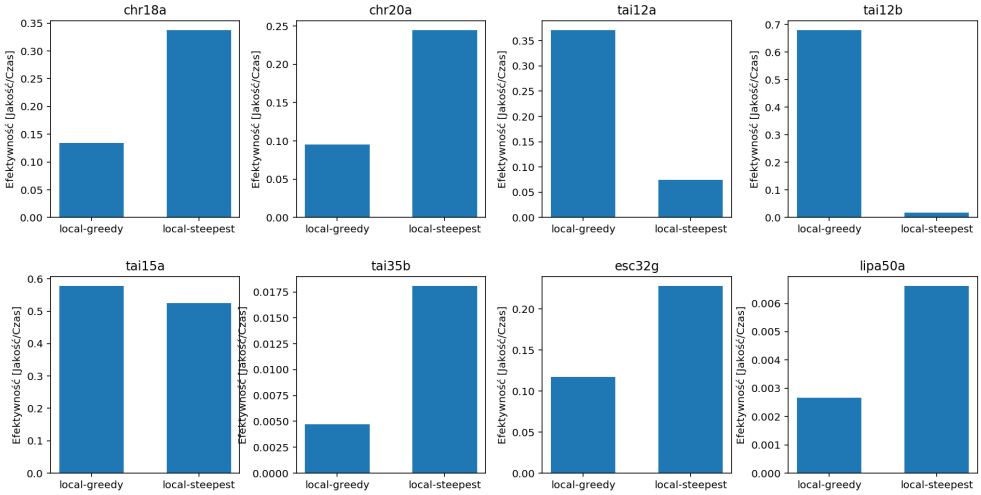
3.2 Czas działania



Porównanie czasu działania algorytmów pokazuje, że algorytm **greedy** przeważnie potrzebuje najwięcej czasu, aby dotrzeć do minimum lokalnego. Wydaje się to nieintuicyjne, ponieważ algorytm **steepest** przeszukuje najwięcej rozwiązań. Algorytm heurystyczny działa dość losowo i czasem potrzebuję dużo czasu na to, aby znaleźć rozwiązanie, a czasem wystarczy mu niewiele czasu. Wersja losowa została celowo pominięta, ponieważ nie ma znaczenia dla tych rozważań porównanie algorytmu z pozostałymi, jest on zawsze najszybszy.

Czasy przedstawione na wykresie są średnią z 300. uruchomień algorytmów.

3.3 Efektywność algorytmów



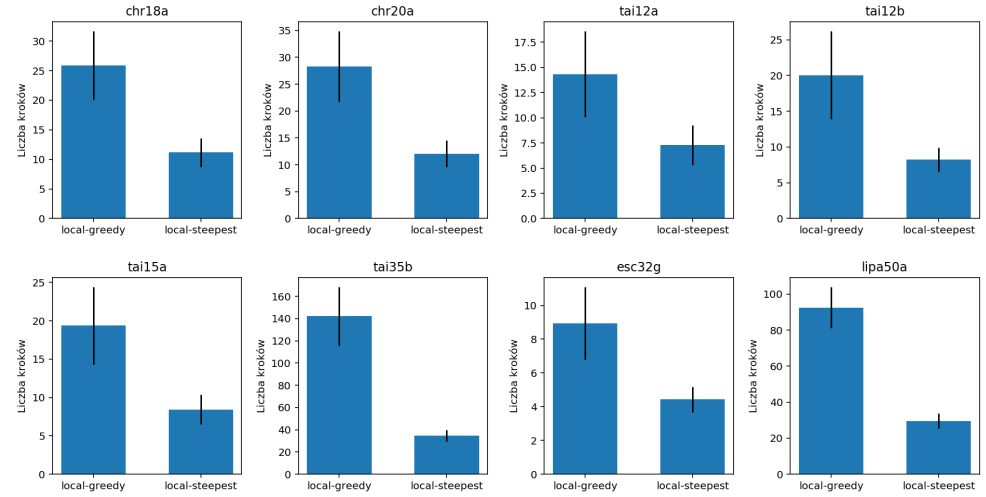
Efektywność algorytmu opisana jest jako stosunek jakości do czasu dotarcia do minimum lokalnego (lub globalnego). Tę zależność można przedstawić za pomocą wzoru:

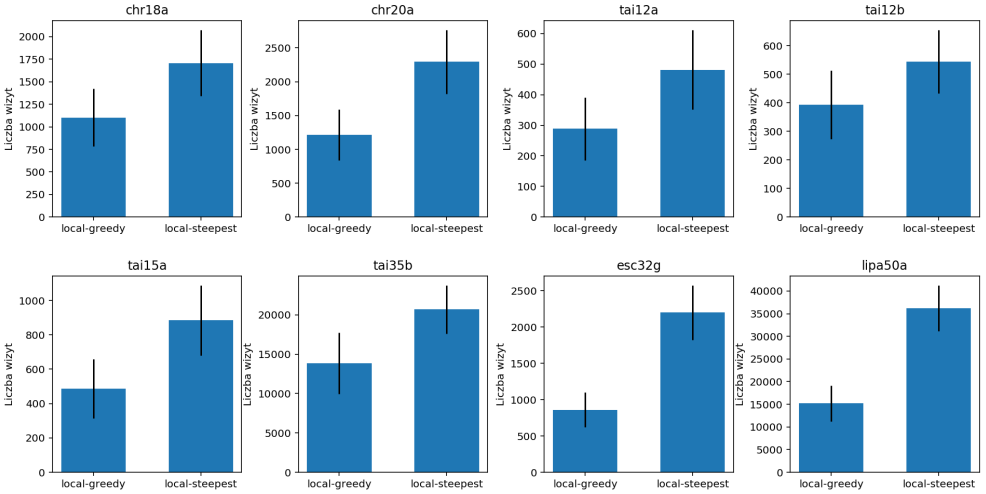
$$\text{effectiveness} = \frac{\text{quality}}{\text{time}} \tag{2}$$

Wzór ten wykorzystuje wcześniej zdefiniowaną przez nas miarę jakośći oraz czas działania algorytmu. Rosnąca jakość oraz malejący czas będzie oznaczał większą efektywność. Powyższy wzór ma pewną wadę, ponieważ nie bierze pod uwagi rozmiaru instancji problemu, przez co efektywność algorytmu dla przykładu **lipa50a** będzie na pewno gorsza od efektywności dla przykładu **chr18a** Na wykresie powyżej efektywność obliczona jest na podstawie jakości oraz średniego czasu dla 300. rozwiązań algorytmów.

3.4 Średnia liczba kroków oraz liczba ocenionych rozwiązań

Intuicyjnie, algorytm **greedy** powinien ocenić mniej rozwiązań, ponieważ akceptuje pierwsze lepsze rozwiązanie dla danych przykładów sąsiadujących. Ciekawą obserwacją jest fakt, iż algorytm **steepest** faktycznie wykonuje więcej porównań, jednak dla danych instancji wykonuje znacznie mniej kroków algorytmu. Wykresy poniżej potwierdzają ten fakt. Dla każdej instancji algorytm **steepest** jednocześnie wykonuje średnio więcej porównań, ale też wykonuje średnio mniej kroków.

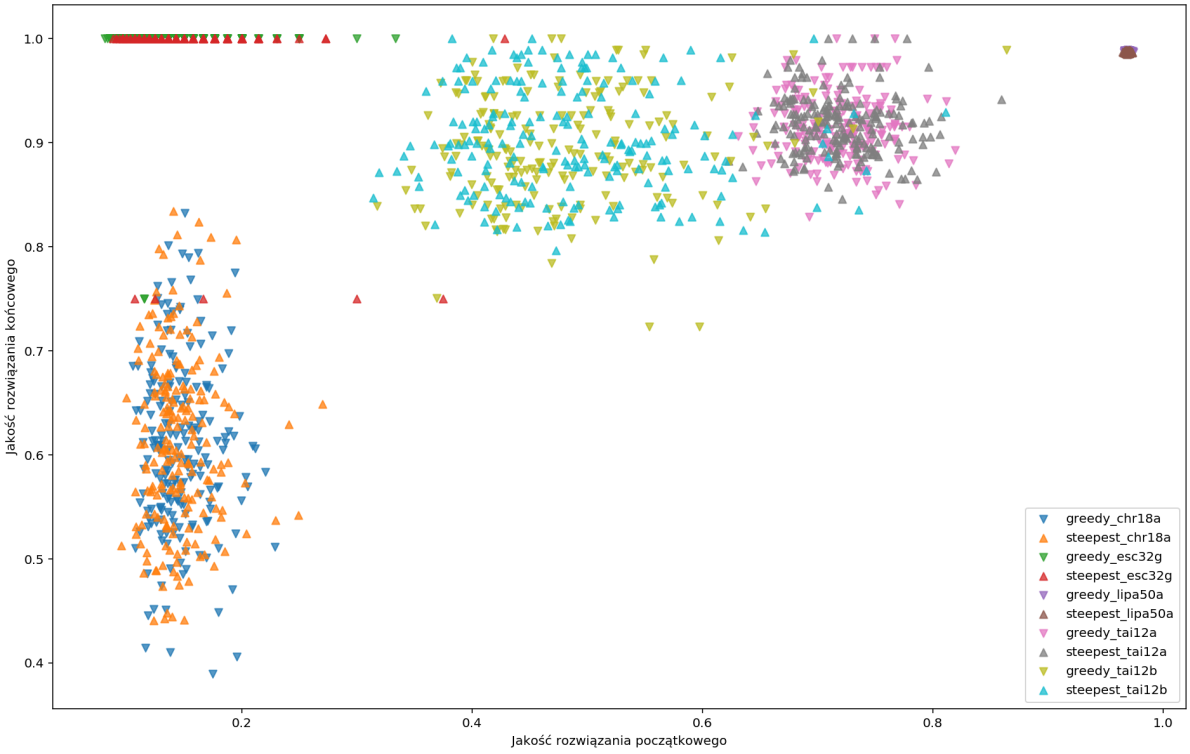

chr18a
chr20a
tai12a
tai12b
tai15a
tai35b
esc32g
lipa50a
Liczba kroków
local-greedy
local-steepest


chr18a
chr20a
tai12a
tai12b
tai15a
tai35b
esc32g
lipa50a
Liczba wizyt
local-greedy
local-steepest

4 Przeszukiwanie lokalne: greedy oraz steepest - rozwiązania początkowe i końcowe

W celu określenia jakości znalezionego rozwiązania skorzystano z miary pokazanej w **równaniu 1**. Miara ta określa, o ile procent znalezione rozwiązanie przewyższa optymalne. Im większa wartość jakości tym lepsze jest dane rozwiązanie.

Poniżej przedstawiono uzyskane wyniki na instancjach: **chr18a**, **esc32g**, **lipa50a**, **tai12a**, **tai12b**.



Dla testowanych instacji, wyniki uzyskane przez algorytmy: steepest oraz greedy, są względem siebie podobne, skupiska punktów obejmują podobne obszary.

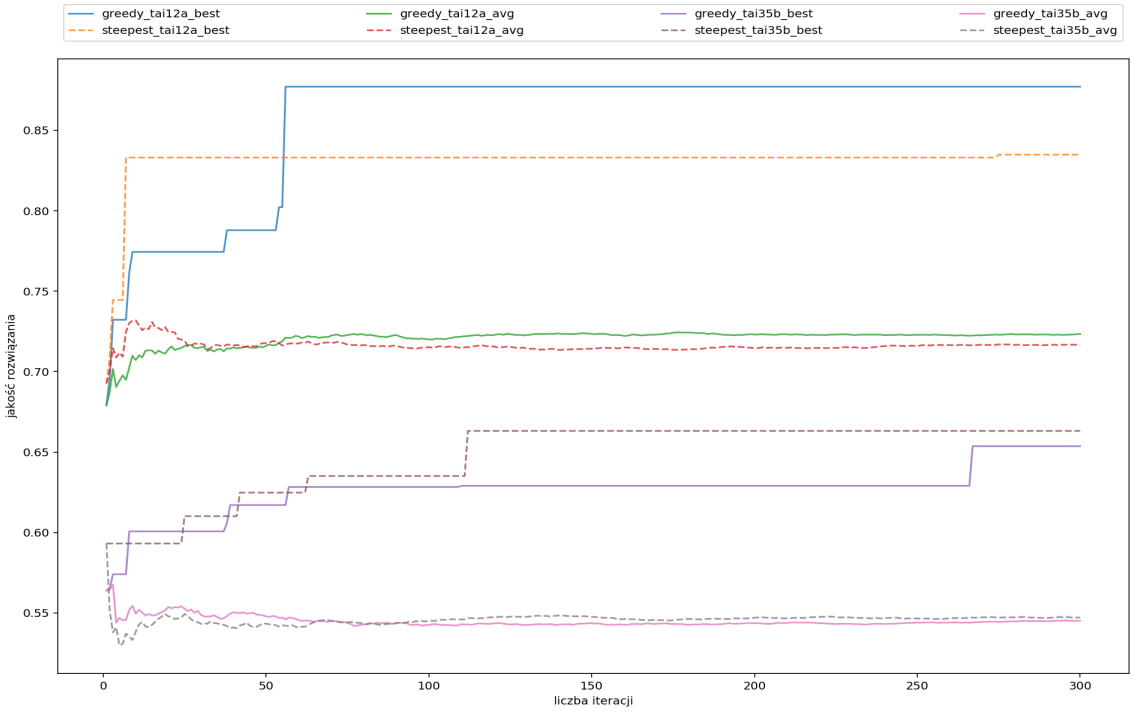
chr18a rzadko osiągał lepszą jakość początkową niż 0.2. Dodatkowo końcowe rozwiązania nie było względem początkowego zależne, na wykresie widzimy, że zakres tych rozwiązał oscylował pomiędzy ok. 0,45 a 0,85.

Dla **tai12a** oraz **tai12b**, widzimy podobny trend, chociaż algorytmy osią-

gały początkowe wyniki o wyższej, lepszej jakości, a kończyły ze zbliżonymi wartościami do optmimum globalnego.

W przypadku **esc32g** i **lipa50a** rozwiązanie początkowe nie miało żadnego wpływu na końcowe, algorytmy osiągały optimum niezależnie od początkowych rozwiązań, które dla **esc32g** charakteryzowały się niską jakością, a dla **lipa50a** bardzo wysoką.

5 Przeszukiwanie lokalne: greedy oraz steepest - liczba restartów

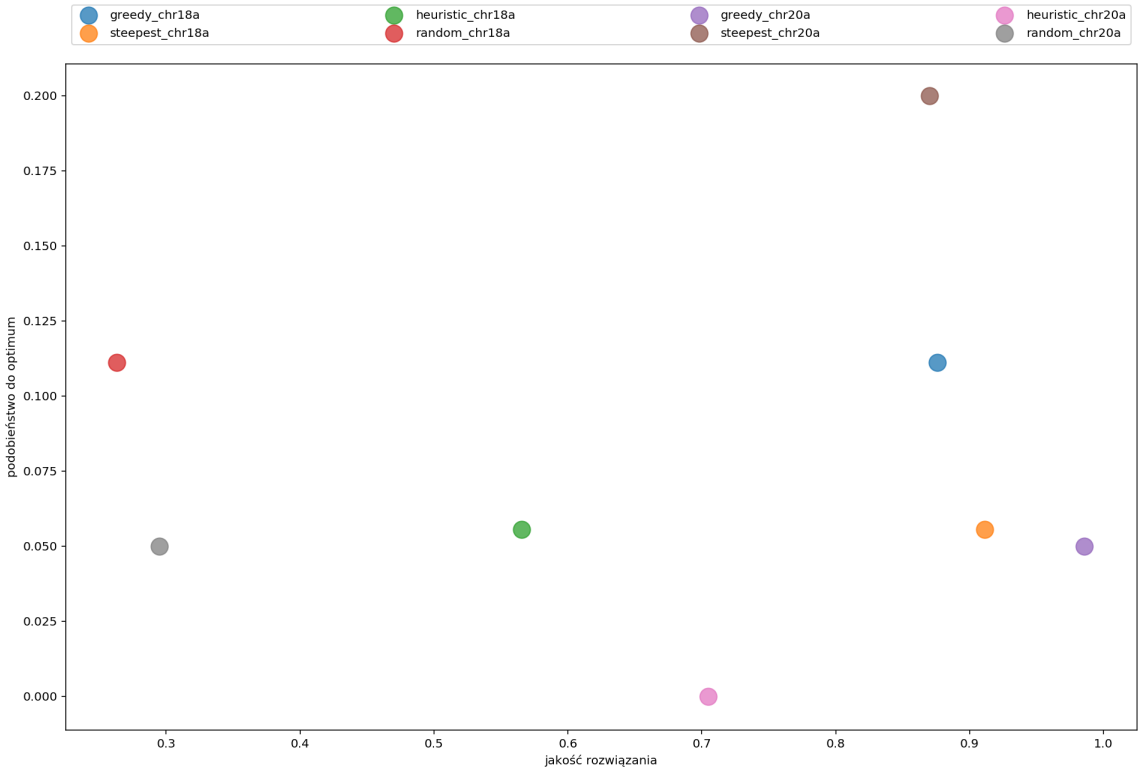


Porównano **tai12a** z **tai12b**. W przypadku tego pierwszego, algorytmy szybko znajdują wynik, który będzie ich rozwiązaniem końcowym. Dla greedy dzieje sie to ok. 50. iteracji, a steepest znajduje takie rozwiązanie już po kilku iteracjach. W **tai12b** algorytmy znajdują finalne rozwiązania po ok. 110. i 260. iteracji.

W przypadku średniej wartości rozwiązań, występują początkowa fluktuacja, tak aby po ok. 50 iteracji ustabilizować trend.

Na podstawie obserwacji ciężko jednoznacznie odpowiedzieć jaka powinna być optymalna liczba restartów. Na pewno restarty pomagają znaleźć lepsze rozwiązanie małym kosztem czasowym, lecz jak to okazało się dla *greedy* przy **tai12b**, takie rozwiązanie może zostać odnalezione znacznie później.

6 Porównanie jakości do uzyskanego podobieństwa



Miarą podobieństwa jest porównanie 2. permutacji odpowiadających rozwiązaniu, oraz sprawdzenie wartości na tych samych pozycjach. Można to zilustro-

wać takim przykładem. Mając permutację $1-2-3-4$ oraz $1-2-4-3$, widać że wartości na 1. oraz 2. pozycji są takie same, natomiast na 3. oraz 4. są inne, dlatego wartość podobieństwa tych 2. rozwiązań wyniesie 0,5.

Porównując wszystkie algorytmy na 2. przykładowych rozwiązaniach widać wyraźnie, że algorytm local search przewyższa jakością algorytm heurystyczny oraz losowy. Powyższy wykres pokazuje również, że podobieństwo oraz jakość rozwiązania nie są z sobą skorelowane. Widać to na przykładzie **greedy_chr20a** oraz **random_chr18a**, gdzie mimo lepszej jakości rozwiązania dla algorytmu **greedy**, podobieństwo rozwiązania jest mniejsze od podobieństwa dla algorytmu **random**.

7 Wnioski

Powyższe ekspereymenty pokazują, że algorytm local search w wersji **greedy** oraz **steepest** jest znacznie lepszy od algorytmu losowego oraz zaproponowanej heurystyki. Dotyczy to głównie jakości rozwiązania, ponieważ algorytm w wersji **random** oraz **heuristic** jest prawie zawsze szybszy od algorytmu local search, natomiast różnica nie jest na tyle duża, żeby warto było poświęcić jakość rozwiązania. Dodatkowo, czas rozwiązania zaproponowanych algorytmów local search można jeszcze poprawić, ponieważ nie została zaimplementowana najwydajniejsza wersja programu.

W dalszych rozważaniach warto byłoby sprawdzić inną miarę podobieństwa 2. rozwiązań, ponieważ aktualna nie mówi nam zbyt dużo. Również większa liczba eksperymentów na innych instancjach mogłaby dostarczyć nowych informacji do wysnucia dalszych wniosków.

7.1 Trudności

Jedną z napotkanych trudności było zaznajomienie się ze środowiskiem .NET Core na systemie macOS. Nie stanowiło to dużego problemu, ponieważ sam język C jest podobny do znanego nam już języka Java, natomiast była to pewnego rodzaju „bariera wejściowa".

7.2 Ulepszenia

W pierwszej fazie implementacji algorytmów local search, skorzystaliśmy z naiwnej wersji która dla każdego zaproponowanego rozwiązania liczyła koszt wprost z wartości przepływów i dystansów. Dużym usprawnieniem wydajności programu okazało się zastosowanie tablicy delt dla tego problemu. Pomogło to wykonać obliczenia w krótszym czasie.