Sprawozdanie z laboratorium:
Metaheurystyki i Obliczenia Inspirowane Biologicznie

Część I: Algorytmy optymalizacji lokalnej, problem STSP

15 listopada 2017

Prowadzący: dr hab. inż. Maciej Komosiński

Autorzy:	**Patryk Gliszczyński**	inf117228	ISWD	patryk.gliszczynski@student.put.poznan.pl
	Mateusz Ledzianowski	inf117226	ISWD	mateusz.ledzianowski@student.put.poznan.pl

Zajęcia w środy, 15:10.

Oświadczam/y, że niniejsze sprawozdanie zostało przygotowane wyłącznie przez powyższych autora/ów, a wszystkie elementy pochodzące z innych źródeł zostały odpowiednio zaznaczone i są cytowane w bibliografii.

Udział autorów

Sprawozdanie zostało wykonane z wykorzystaniem szablonu dr hab. inż. Macieja Komosińskiego. [2]

Patryk Gliszczyński

PG zaimplementował algorytmy i przygotował środowisko badawcze, napisał skrypt do generowania wykresów.

Mateusz Ledzianowski

ML zaimplementowała funkcję mierzącą czas, generującą permutację, przeprowadził eksperymenty i opisał je w sprawozdaniu.

1 Symetryczny problem komiwojażera (STSP)

1.1 Opis problemu

Symetryczny problem komiwojażera modeluje sytuację znaną z rzeczywistego świata, w której osoba ma odwiedzić konkretne miejsca w dowolnej kolejności i wrócić do miejsca początkowego tak, aby pokonać jak najkrótszą drogę. Z tego typu zadaniem mierzą się przede wszystkim wszyscy dostawcy, listonosze, akwizytorzy. W symetrycznym problemie komiwojażera odległości pomiędzy dwoma miejscami są takie same w obie strony. Problem nie daje możliwości tworzenia dróg jednokierunkowych, a także budowana sieć dróg jest grafem pełnym.

1.2 Złożoność

W tak postawionym problemie, istnieje różnych $\theta(n!)$ rozwiązań, gdzie n oznacza liczbę miejsc do odwiedzenia. Miejsca możemy odwiedzać w dowolnej kolejności, więc jeśli zostaną one ponumerowane od 1 do n, każda permutacja n-elementowa może reprezentować pełne rozwiązanie. Rozwiązanie w postaci permutacji możemy odczytywać w taki sposób, że z miejsca na pozycji i, przemieszczamy się do miejsca na pozycji $i+1$, pamiętając o tym, żeby z miejsca na pozycji n wrócić do startowego o indeksie 1. Przestrzeń rozwiązań jest więc bardzo duża i trudno jest przejrzeć je wszystkie. Jeśli bylibyśmy w stanie sprawdzać $1'000'000'000$ rozwiązań w czasie 1 sekundy, rozwiązania dokładnego dla $n = 16$, szukalibyśmy przez ok. $6h$, a znalezienie go dla $n = 20$ zajęłoby 77 lat.

1.3 Rozwiązanie losowe

Ponieważ rozwiązania można reprezentować w postaci permutacji, da się w łatwy i szybki sposób wygenerować losowe początkowe rozwiązanie dla wielu innych algorytmów poprzez wygenerowanie losowej permutacji. Złożoność generowania permutacji to $\theta(n)$, gdzie n jest jej długością. Aby wygenerować losową permutację należy zastosować poniższą procedurę:

1. Wypełnij tablicę liczbami od 1 do n.
2. $i := n$.
3. Zamień element z pozycji $i - 1$ z elementem na losowej wcześniejszej lub tej samej pozycji (od 0 do $i - 1$).
4. $i := i - 1$.
5. Jeżeli $i > 1$, wróć do kroku 2.

1.4 Heurystyka

Dla problemu komiwojażera istnieje prosta heurystyka o złożoności $\theta(n^2)$, dająca zadowalające wyniki — przeciętnie odległe od rozwiązania optymalnego o $10 - 15\%$. [3] Polega ona na wykonaniu poniższych kroków:

1. Wybierz losowe miasto początkowe.
2. Znajdź najbliższe nieodwiedzone miasto i udaj się tam.
3. Jeśli pozostały jeszcze jakieś nieodwiedzone miasta, idź do kroku 2.
4. Wróć do początkowego miasta.

1.5 Instancje problemu

Problem jest bardzo powszechny i istnieje wiele gotowych instancji, których można użyć w badaniach. Wybraliśmy 8 instancji z udostępnionego przez uniwersytet „Heidelberg" zbioru. [1] Są to:

1. berlin52,
2. ch130,
3. eil51,
4. kroA100,
5. lin105,
6. pr76,
7. rd100,
8. tsp225.

Przy doborze konkretnych instancji zależało nam na tym, aby nie były one zbyt duże (przeważnie do 200 miast), a także miały znalezione rozwiązania optymalne, były różnorodne oraz podane w formacie EUC_2D, czyli za pomocą współrzędnych na dwuwymiarowej płaszczyźnie Euklidesowej.

2 Optymalizacja lokalna

2.1 Wstęp

Algorytmy optymalizacji lokalnej pozwalają na poszukiwanie lepszych rozwiązań od aktualnie znalezionego, tak długo, aż w zadanym sąsiedztwie nie można go już bardziej poprawić. Dzięki takiej optymalizacji, nie trzeba przeszukiwać całej przestrzeni rozwiązań, a osiągać dobre wyniki.

2.2 Operatory sąsiedztwa

Aby stosować przeszukiwanie lokalne, należy określić sąsiedztwo dla każdego rozwiązania. Jest to przestrzeń rozwiązań, które są podobne do posiadanego. Jednym z możliwych sąsiedztw jest OPT-2.

2.2.1 OPT-2

Sąsiedztwo OPT-2 definiuje się poprzez zamianę kolejności wierzchołków na dwóch pozycjach lub odwrócenie kolejności odwiedzania miast na łuku pomiędzy dwoma pozycjami. W zależności od tego, czy rozważany problem komiwojażera jest symetryczny, czy nie — różne sąsiedztwa sprawdzają się z odmienną skutecznością. Rozważając, które z sąsiedztw wybrać, należy zastanowić się, które z nich w najmniejszym stopniu wpływa na funkcję celu.

Zamiana miast Sąsiedztwo OPT-2 polegające na zamianie miast można zaimplementować poprzez prostą zamianę elementów na pozycji i-tej i j-tej. Jest ono skuteczniejsze dla problemu asymetrycznego komiwojażera, ponieważ zamiana łuków spowodowałaby większą różnicę w funkcji celu — inne byłyby nie tylko cztery drogi, prowadzące do dwóch zamienianych miast, lecz także wszystkie drogi na łuku.

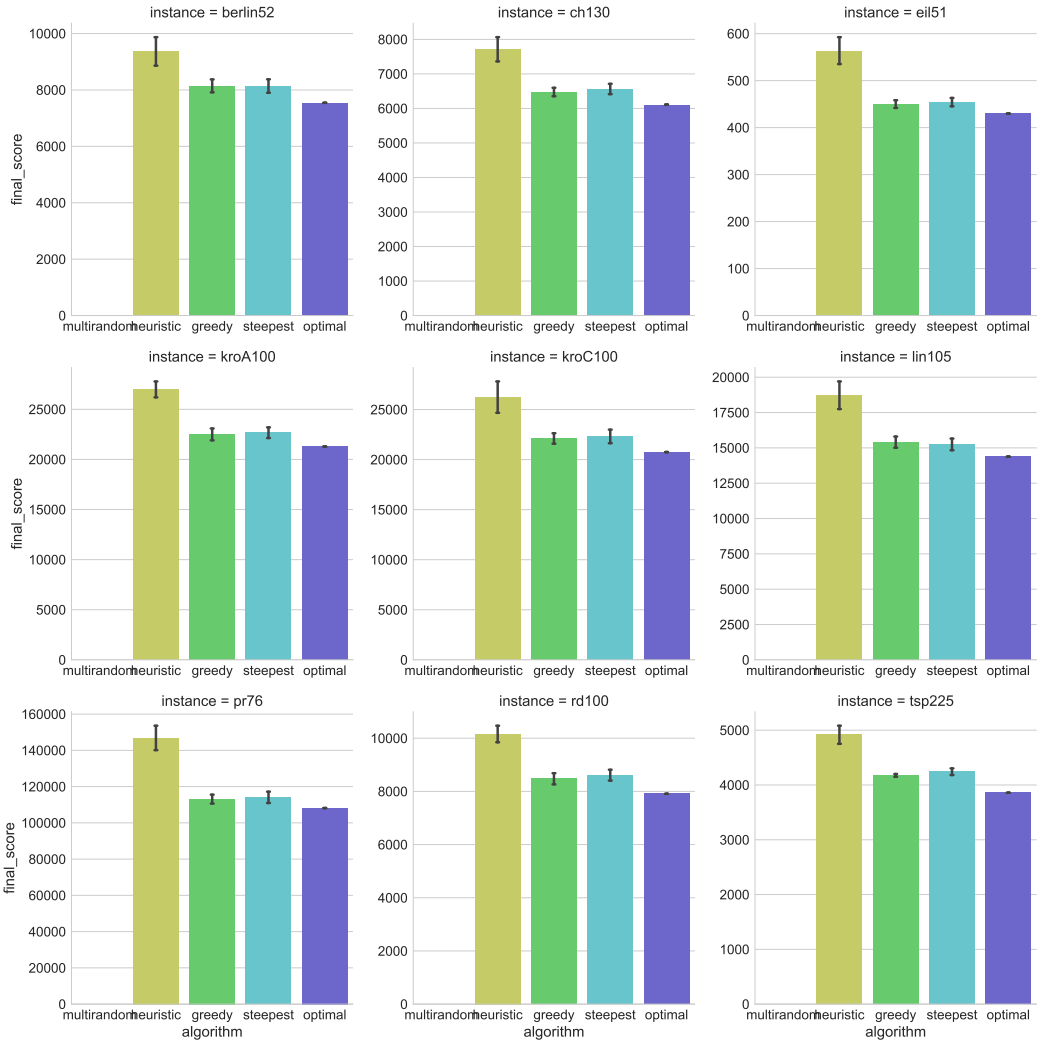
Odwrócenie łuku W przypadku odwrócenia łuku pomiędzy miastami na pozycjach i-tej i j-tej, funkcja celu ulegnie zmianie spowodowanej tylko modyfikacją dwóch dróg prowadzących do miast i-tego i j-tego z jednej strony, a z drugiej odwrócenie łuku, gdy drogi w obie strony mają taką samą długość, niczego nie zmieni.

2.3 Greedy

Jednym z algorytmów przeszukiwania lokalnego jest algorytm Greedy. Polega on na tym, że jeśli w momencie przeszukiwania sąsiedztwa aktualnego rozwiązania, znajdzie rozwiązanie lepsze, natychmiast je wybiera jako aktualne i szuka następnego w jego sąsiedztwie, dopóki istnieją rozwiązania poprawiające aktualne.

2.4 Steepest

Algorytm Steepest różni się od poprzednika tym, że gdy przegląda swoje sąsiedztwo, nie wybiera jako aktualne rozwiązanie, pierwszego znalezionego, lepszego rozwiązania, ale zawsze przegląda wszystkich sąsiadów i wybiera najlepszego spośród nich, dzięki czemu porusza się „po najbardziej stromym zboczu" na wykresie funkcji celu.

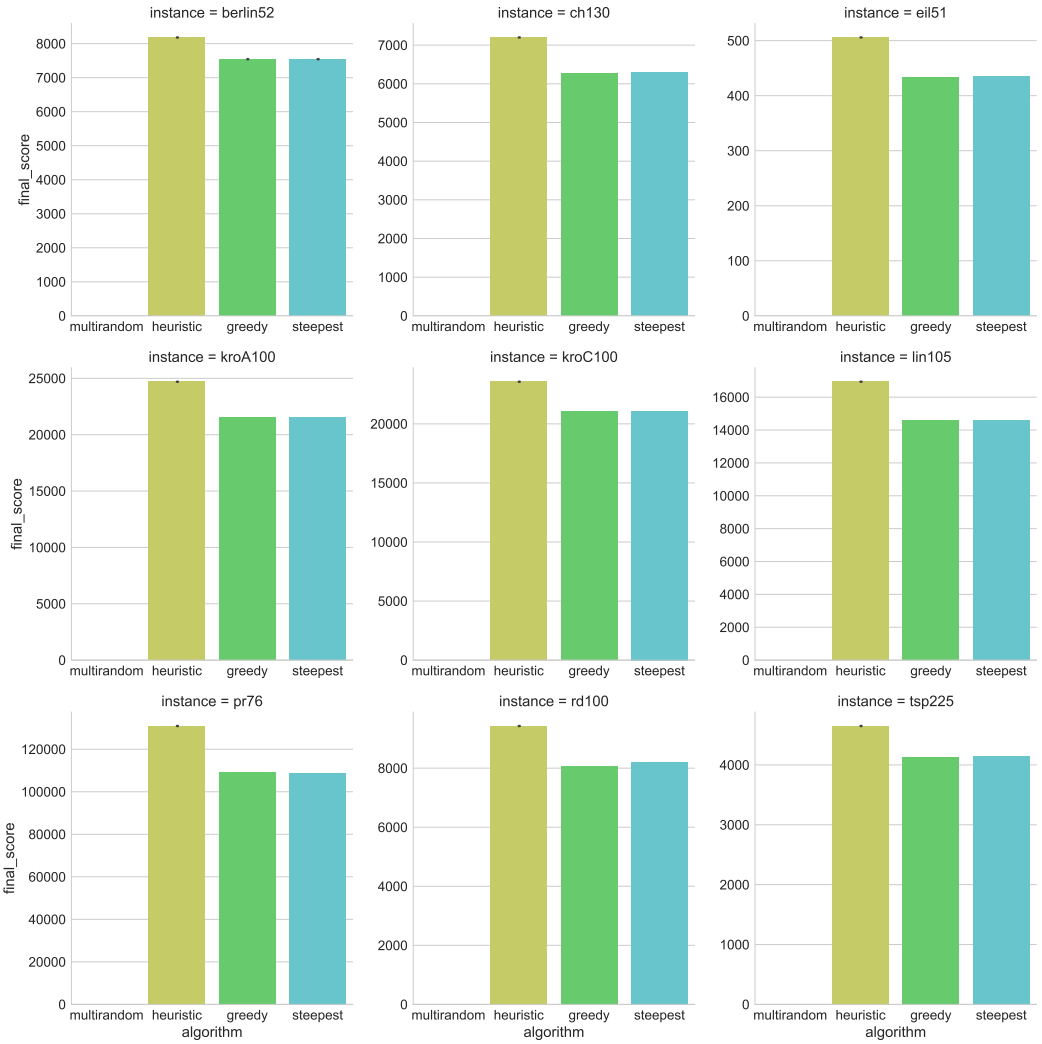


Rysunek 1: Porównanie średnich rozwiązań na różnych instancjach.

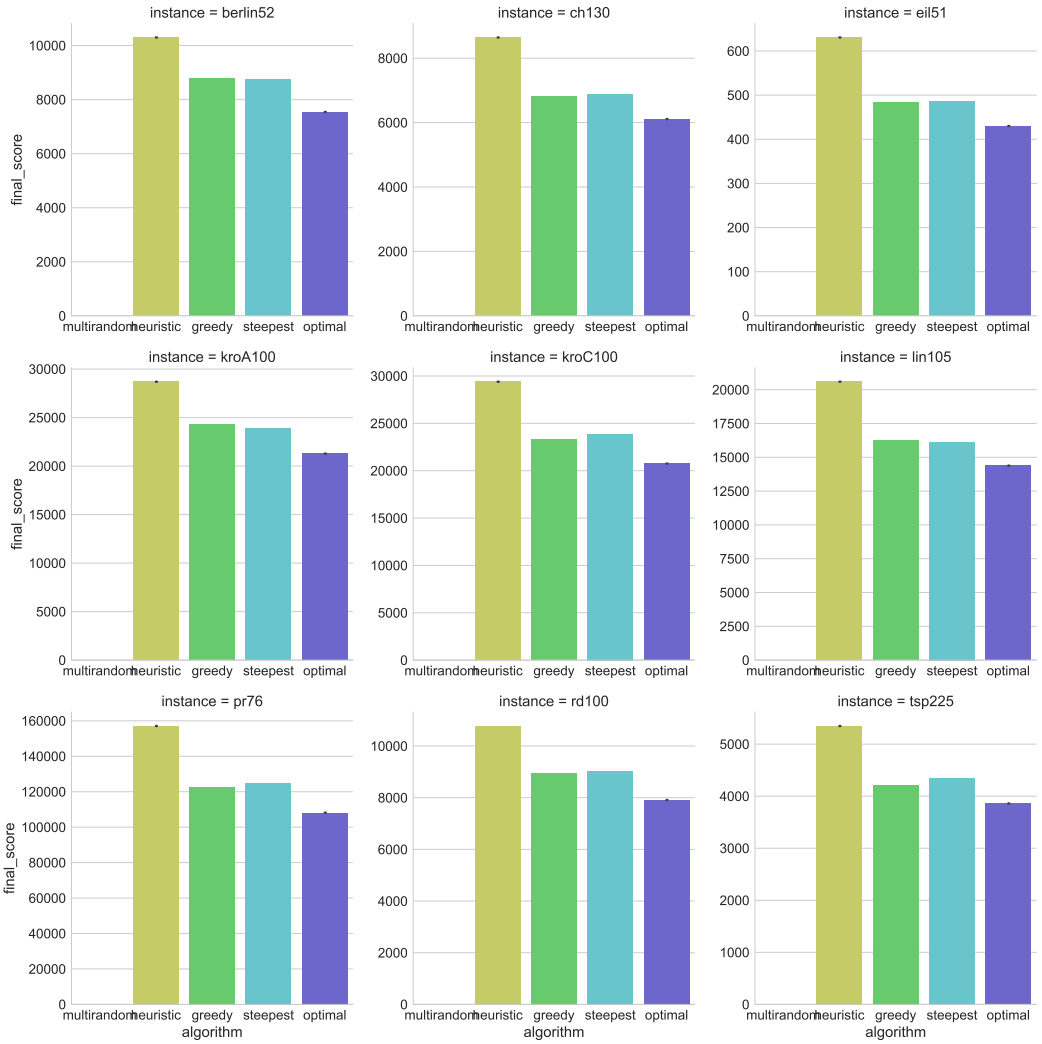
3 Eksperymenty

3.1 Odległość od optimum

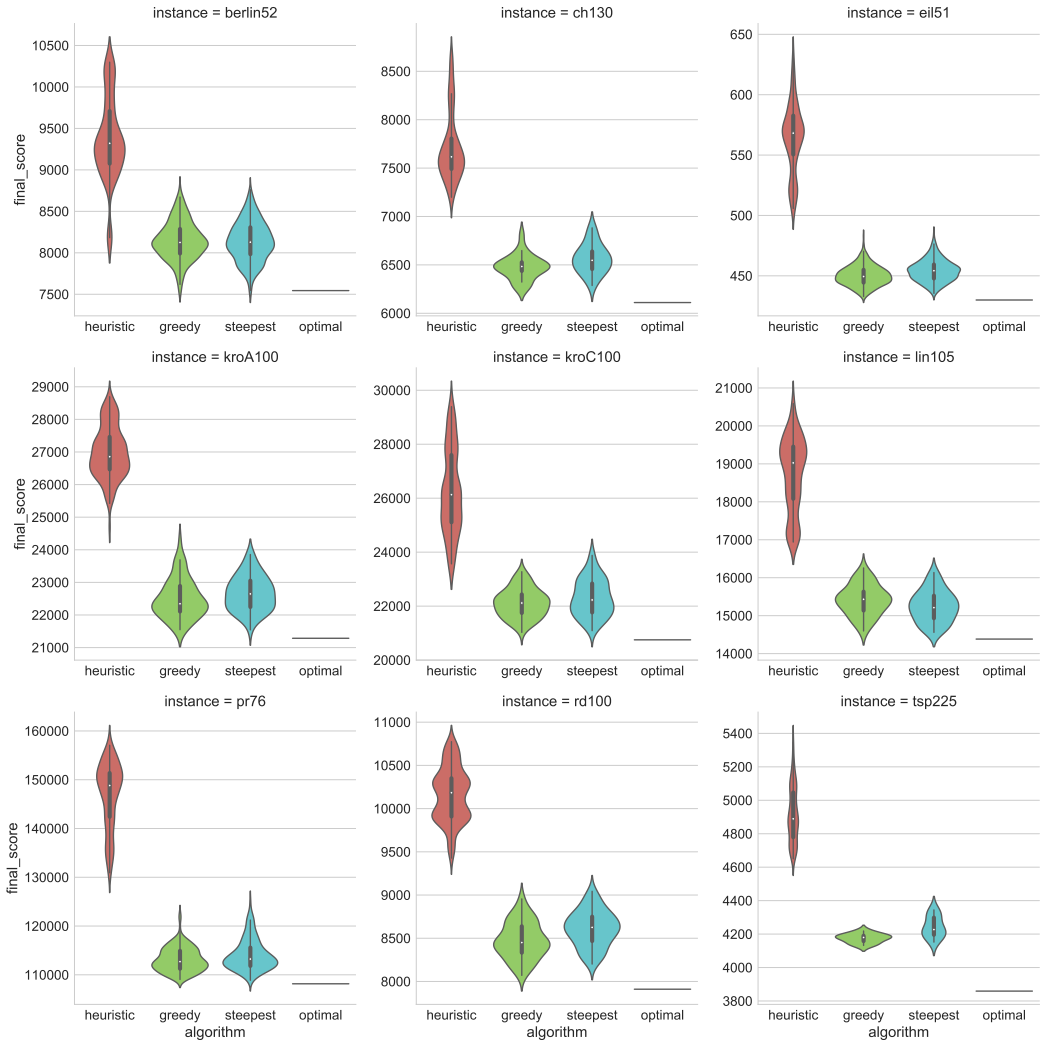
Z rys. 1 wynika, że dla każdej instancji problemu, heurystyka i algorytmy przeszukiwania lokalnego osiągają podobne wyniki, które są kilkukrotnie lepsze od rozwiązania losowego, z którego startują zarówno Greedy, jak i Steepest. Algorytmy Greedy i Steepest dają podobne rezultaty — różnią się nieznacznie dla różnych instancji i nie ma tendencji co do tego, który ogólnie działa lepiej. Można za to zaobserwować, że oba algorytmy przeszukiwania lokalnego są lepsze niż heurystyka.



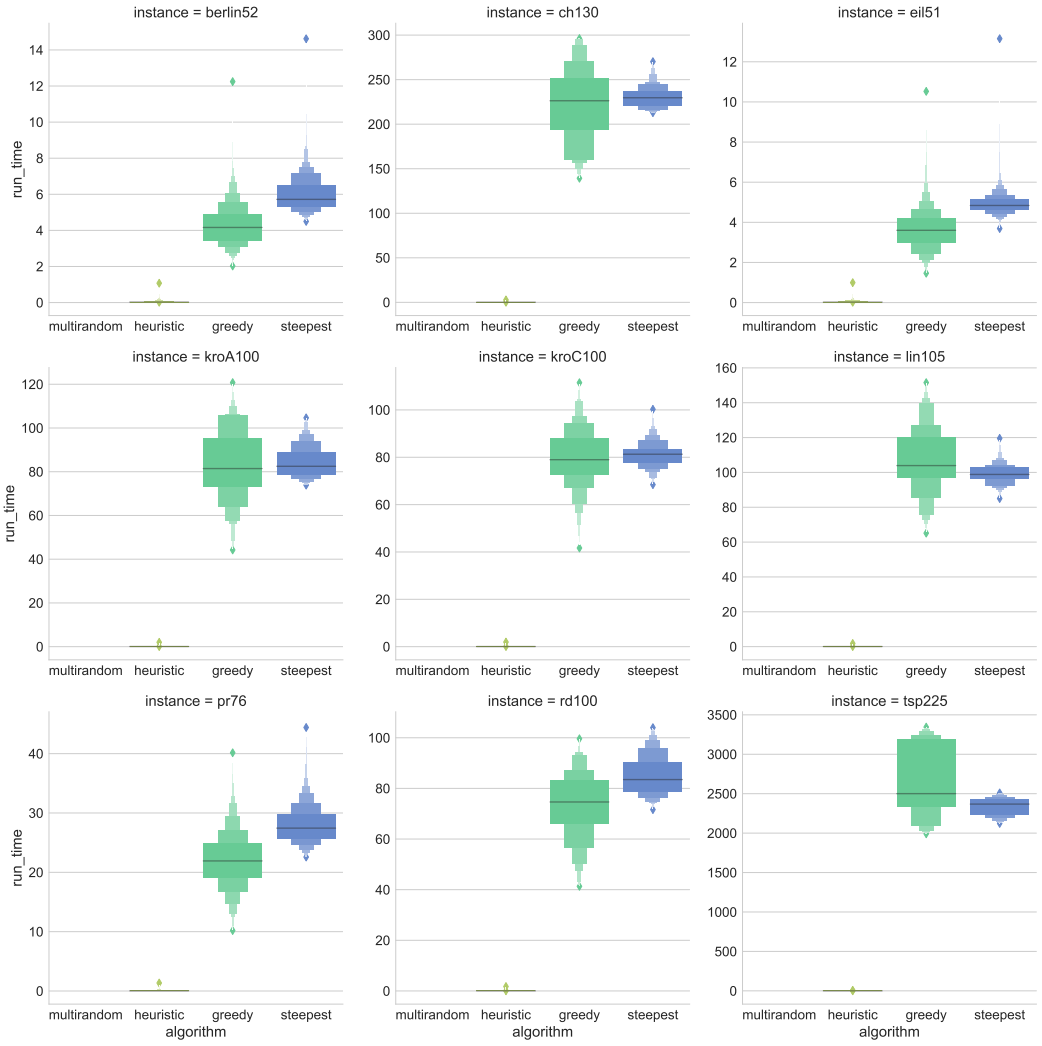
Rysunek 2: Porównanie najlepszych znalezionych rozwiązań przez algorytmy na różnych instancjach.



Rysunek 3: Porównanie najgorszych znalezionych rozwiązań przez algorytmy na różnych instancjach.



Rysunek 4: Porównanie rozkładów znalezionych rozwiązań przez algorytmy na różnych instancjach.



Rysunek 5: Porównanie czasu działania algorytmów na poszczególnych instancjach.

3.2 Czas działania

Algorytm losowy jest zdecydowanie najszybszy, ponieważ sprawdza tylko jedno rozwiązanie. Podobnie heurystyka, jednak wyznaczenie przez nią rozwiązania zajmuje trochę więcej czasu. Greedy i Steepest przeszukują przestrzeń rozwiązań, dopóki nie mogą już bardziej poprawić wyniku, przez co trwają zdecydowanie najdłużej. Przeważnie obliczenia Steepesta zajmują więcej czasu, niż wykonanie algorytmu Greedy. Na rys. 5 można też zauważyć, że pojedyncze wykonania trwają znacznie dłużej od pozostałych — szczególnie jest to widoczne przy instancji berlin52.

3.3 Efektywność algorytmów

3.3.1 Wybrana miara

Aby porównać algorytmy pod względem jakości, można to zrobić przez zdefiniowanie kosztu czasowego, jaki trzeba ponieść, aby uzyskać dane rozwiązanie. Czyli należy policzyć iloraz $cost = time/result$, co jest przedstawione na rys. 6. Natomiast efektywnością algorytmu jest odwrotność kosztu, która została przedstawiona na rys. 7.

3.3.2 Wyniki

Z wykresów 6 i 7 można by wyciągnąć wniosek, że najefektywniejszym algorytmem jest algorytm losowy, ponieważ zajmuje najmniej czasu. Takie wyniki są jednak konsekwencją tego, że przy każdy krok algorytmu przeszukiwania lokalnego jest dość kosztowny, ponieważ przegląda się wiele rozwiązań, a rozwiązanie jest poprawiane nieznacznie (zgodnie z założeniem algorytmów przeszukiwania lokalnego, że funkcja celu sąsiadów niewiele się różni). Podobnie przy heurystyce — nawet złożoność $\theta(n^2)$ nie gwarantuje ulepszenia rozwiązania n-krotnie, a jedynie kilkukrotnie, więc koszt algorytmu jest stosunkowo wysoki.

Na wykresie kosztu widać również, że Greedy jest kosztowniejszy od Steepesta, ponieważ daje zbliżone rozwiązania w dłuższym czasie.

3.4 Liczba kroków algorytmów lokalnego przeszukiwania

Wykres 8 przedstawia liczbę kroków wykonanych przez algorytmy lokalnego przeszukiwania. Widać, że Greedy wykonuje ich kilkakrotnie więcej niż Steepest, a także, że liczba wykonanych kroków przez ten algorytm jest dużo bardziej zróżnicowana, w zależności od przypadku startowego. Jest to spowodowane tym, że algorytm Steepest we wcześniejszym kroku osiąga lokalnie najlepsze rozwiązanie, ponieważ za każdym razem wybiera to, które najbardziej poprawia wynik z całego sąsiedztwa.

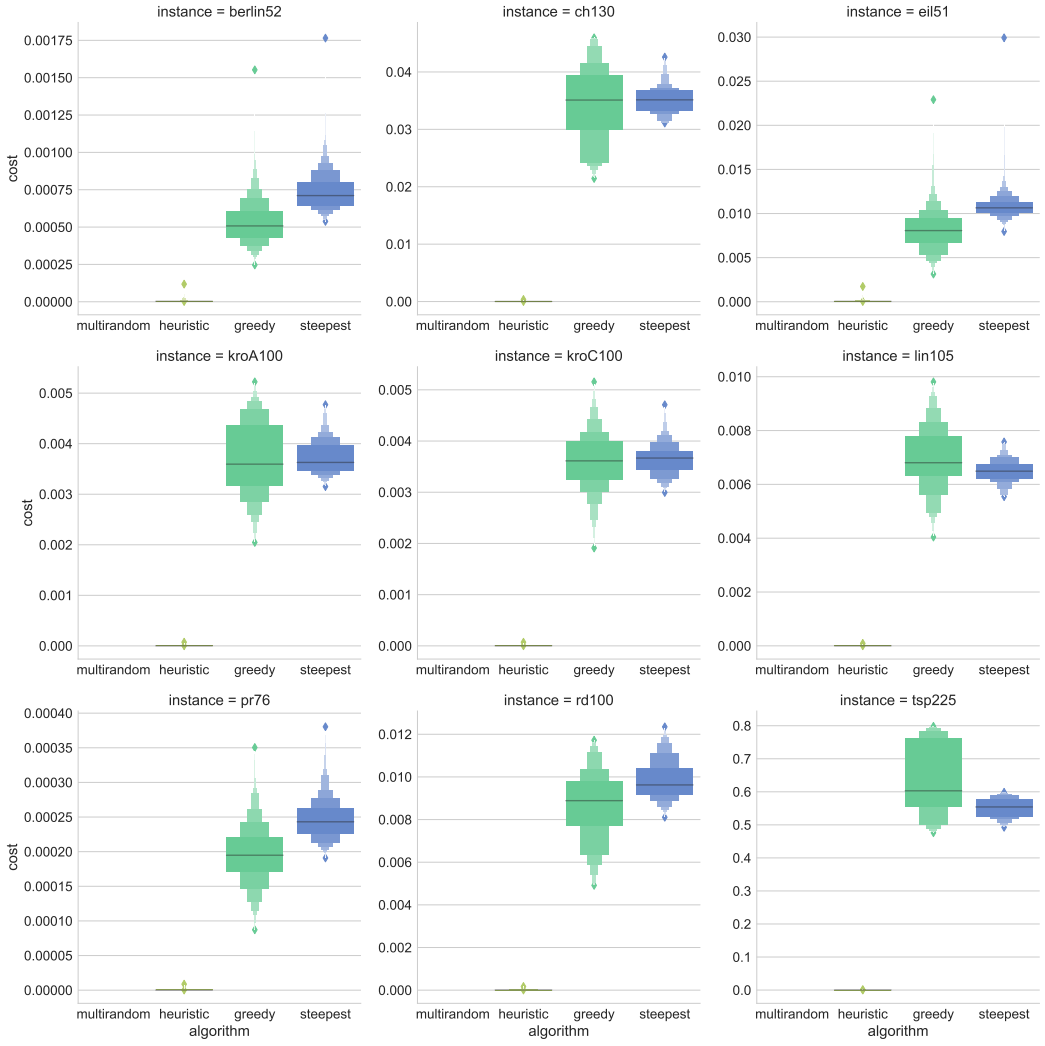
3.5 Średnia liczba przeszukanych rozwiązań

Na wykresie 9 jest przedstawiona liczba rozwiązań przeszukiwanych przez oba rozważane algorytmy lokalnego przeszukiwania. Można na nim zauważyć, że średnio Steepest przeszukuje większą przestrzeń, choć zdarzają się wykonania algorytmu Greedy, które sprawdzają większą liczbę rozwiązań. Jest to o tyle ciekawe, że Steepest wykonuje mniej kroków, niż Greedy, a i tak aby je wykonać, przegląda więcej rozwiązań. Co ciekawe, ta tendencja jest odmienna dla największej instancji, podobnie też, czas działania algorytmu Greedy dla niej jest większy od Steepesta.

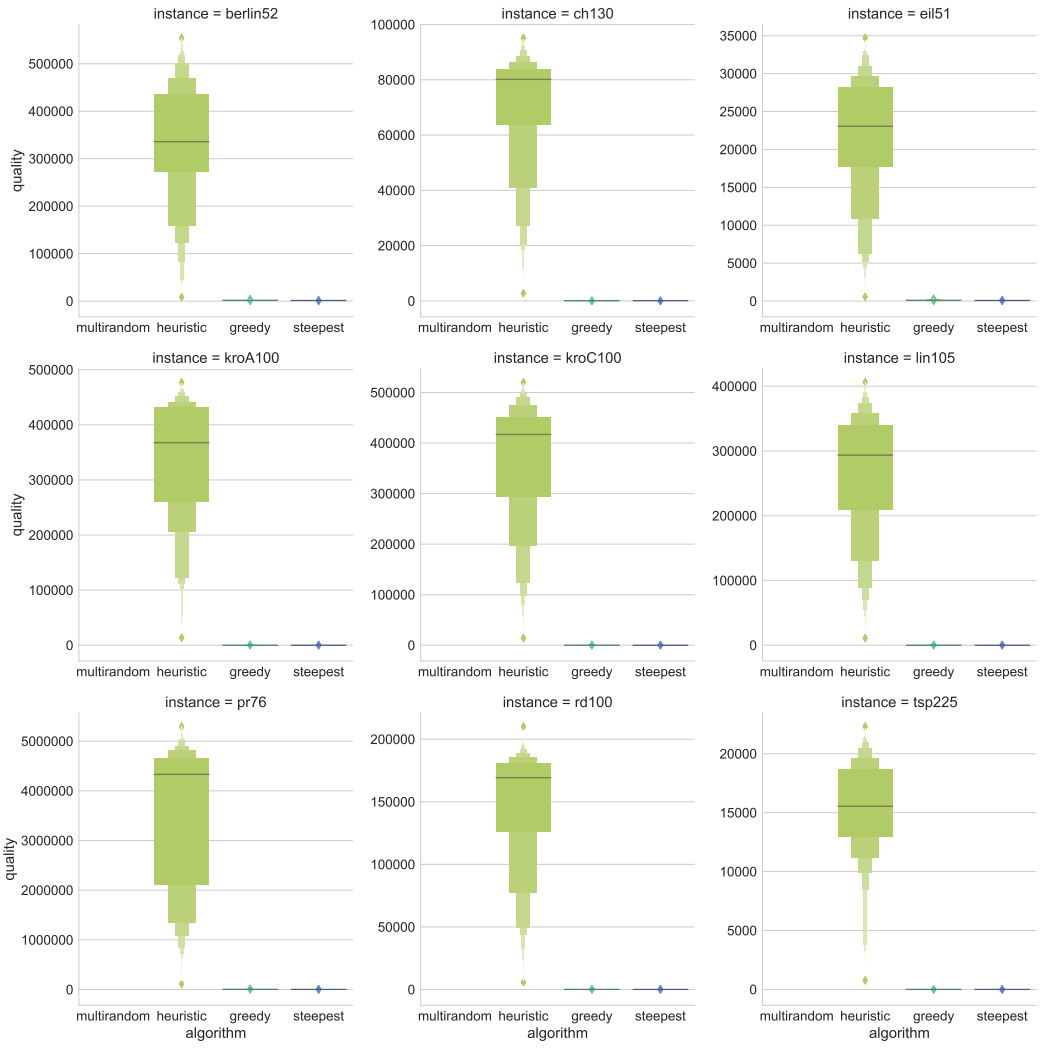
3.6 Przeszukiwanie lokalne

3.6.1 Jakość rozwiązania początkowego a końcowego

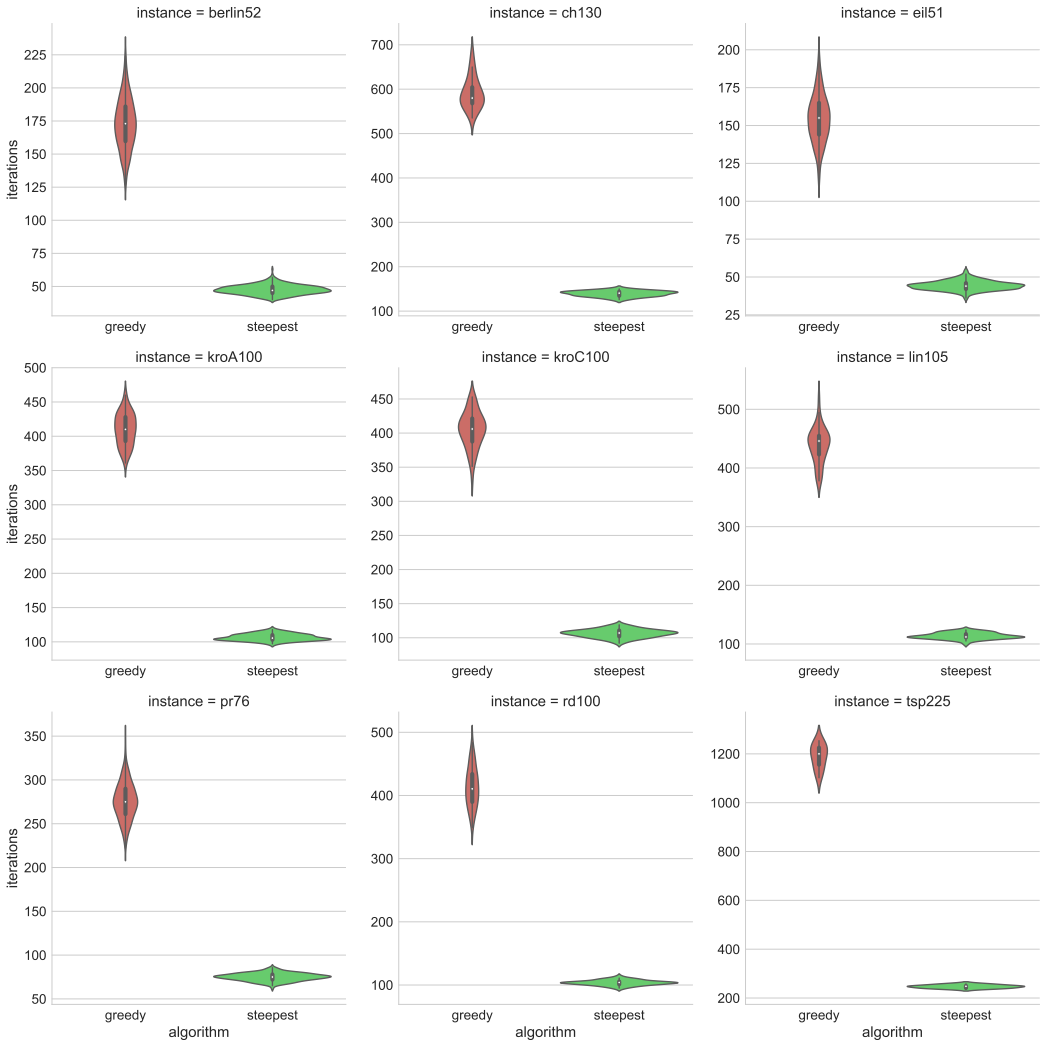
Badając zależność między rozwiązaniem początkowym, a końcowym, nie udało nam się zaobserwować związku na wykresie punktowym 10. Punkty są bardzo rozrzucone i nie daje się ich odpowiednio pogrupować. Na wykresie skrzypcowym 11 zostały przedstawione rozwiązania początkowe i końcowe dla obu algorytmów. Widać, że są wyraźnie od siebie oddalone i zgrupowane w osobne chmury. Można zatem przypuszczać, że jakość rozwiązania końcowego nie zależy od jakości rozwiązania początkowego.



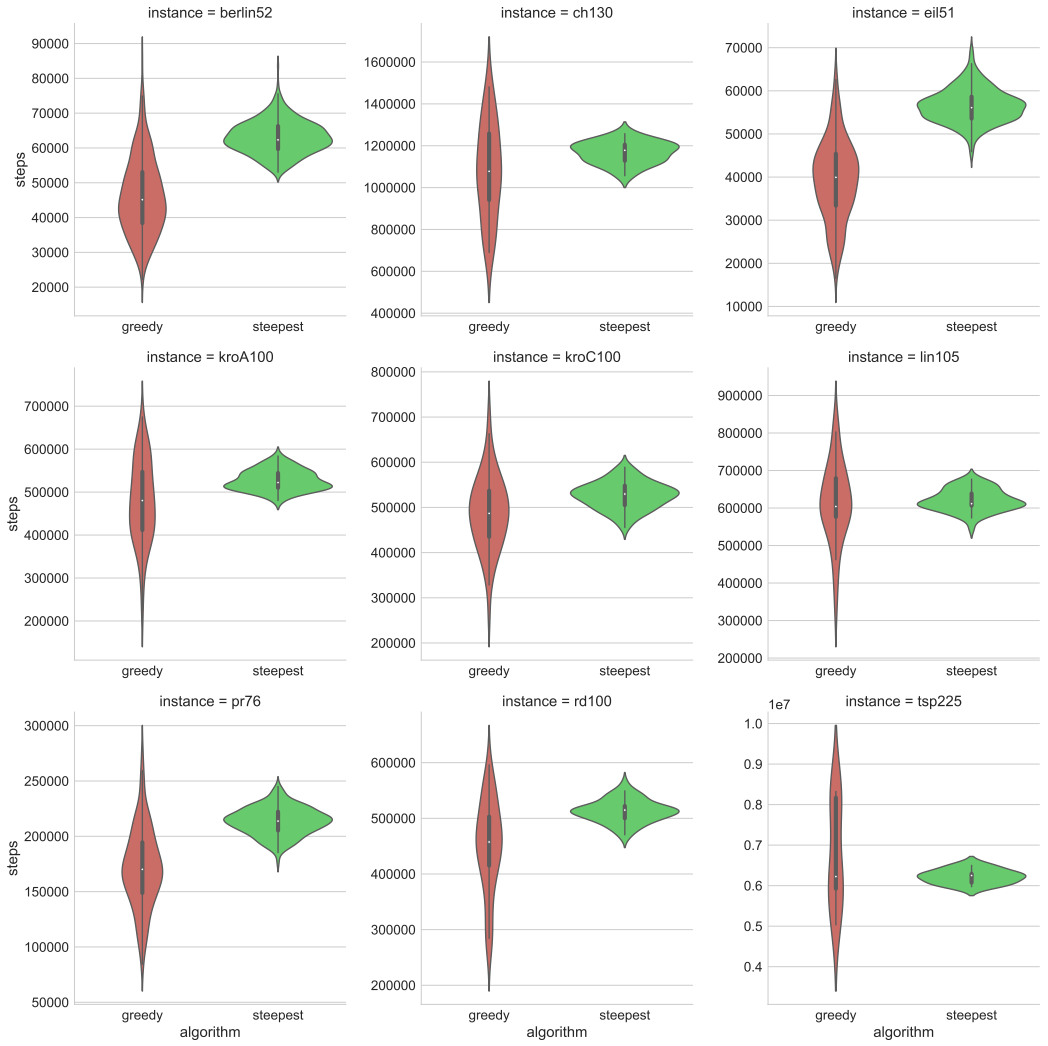
Rysunek 6: Porównanie kosztów algorytmów na poszczególnych instancjach.



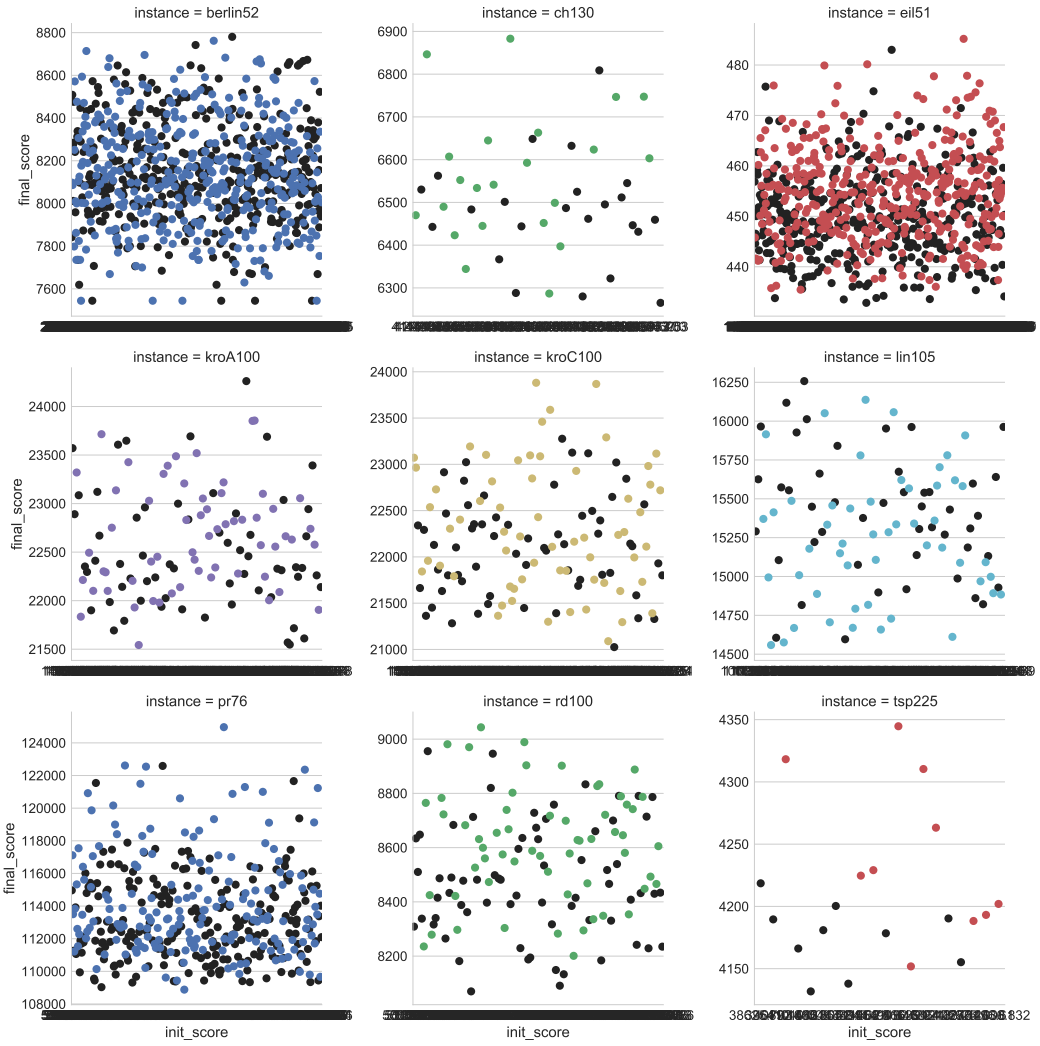
Rysunek 7: Porównanie efektywności algorytmów na poszczególnych instancjach.



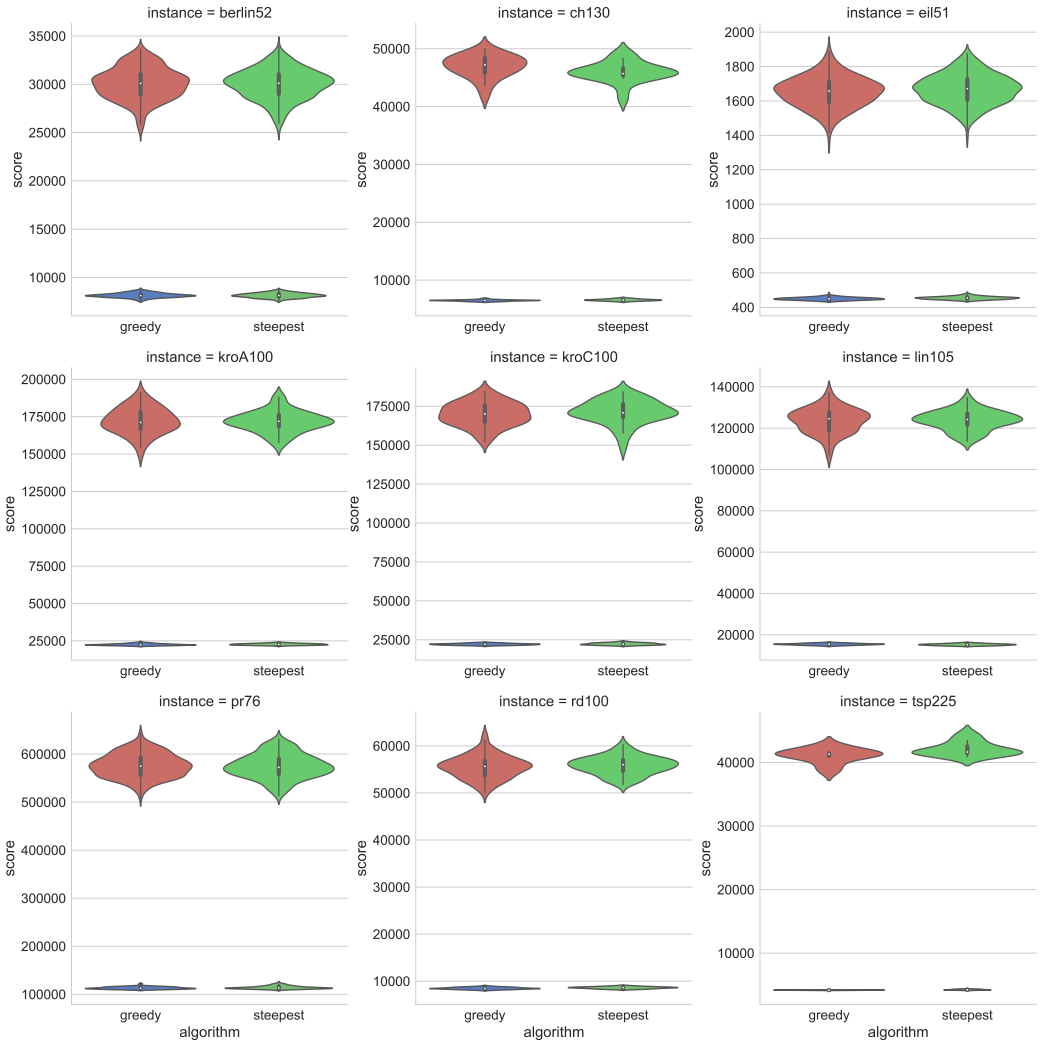
Rysunek 8: Porównanie algorytmów Greedy Search i Steepest pod względem liczby kroków do zatrzymania.



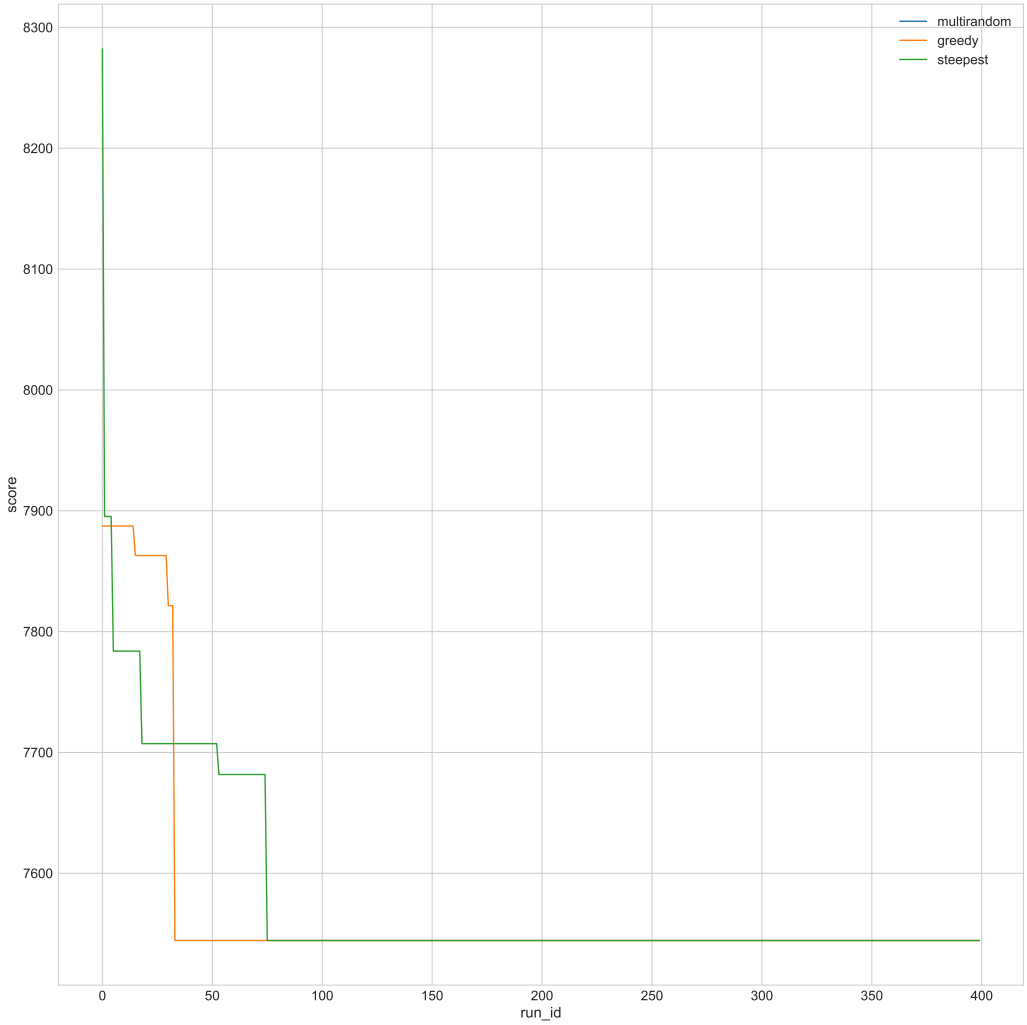
Rysunek 9: Porównanie algorytmów Greedy Search i Steepest pod względem liczby przeszukanych rozwiązań.



Rysunek 10: Porównanie jakości rozwiązań początkowych i końcowych przez algorytmy Greedy Search i Steepest przedstawione na wykresie punktowym.



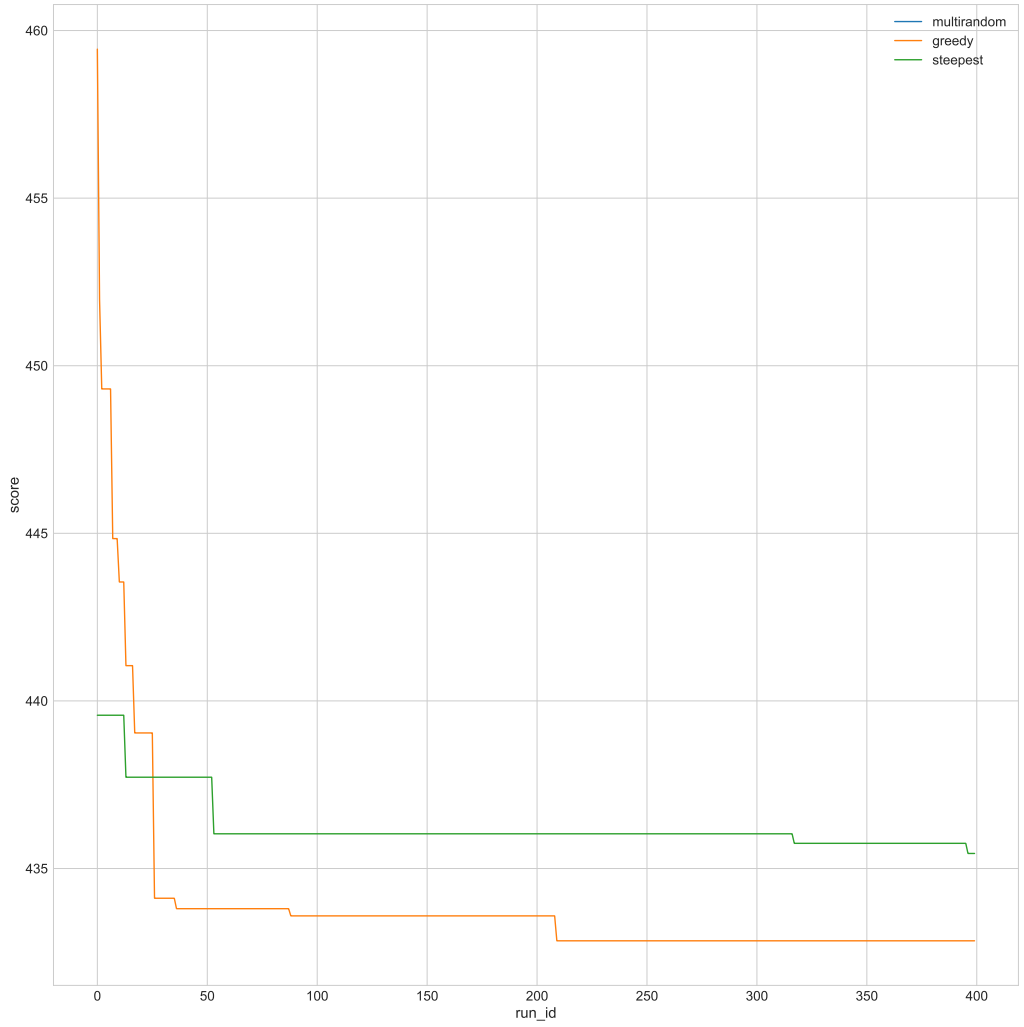
Rysunek 11: Porównanie jakości rozwiązań początkowych i końcowych przez algorytmy Greedy Search i Steepest.



Rysunek 12: Porównanie jakości rozwiązań algorytmów Gready Search i Steepest w zależności od liczby uruchomień tych algorytmów dla różnych rozwiązań początkowych dla zbioru berlin52.

3.6.2 Wielokrotne uruchamianie dla różnych rozwiązań początkowych

Wykresy 12 i 13 przedstawiają wartość najlepszego znalezionego rozwiązania po i-tej iteracji. Jak widać, poprawia się ona co pewien czas. Im rozwiązanie jest lepsze, tym ten czas jest dłuższy. Im więcej razy algorytm będzie uruchamiany z różnych rozwiązań początkowych, tym istnieje większa szansa, że osiągnie on lepszy wynik, więc warto powtarzać uruchomienia dla różnych rozwiązań początkowych, aby pełniej przeszukać przestrzeń wszystkich rozwiązań.



Rysunek 13: Porównanie jakości rozwiązań algorytmów Gready Search i Steepest w zależności od liczby uruchomień tych algorytmów dla różnych rozwiązań początkowych dla zbioru eil51.

3.7 Porównanie rozwiązań

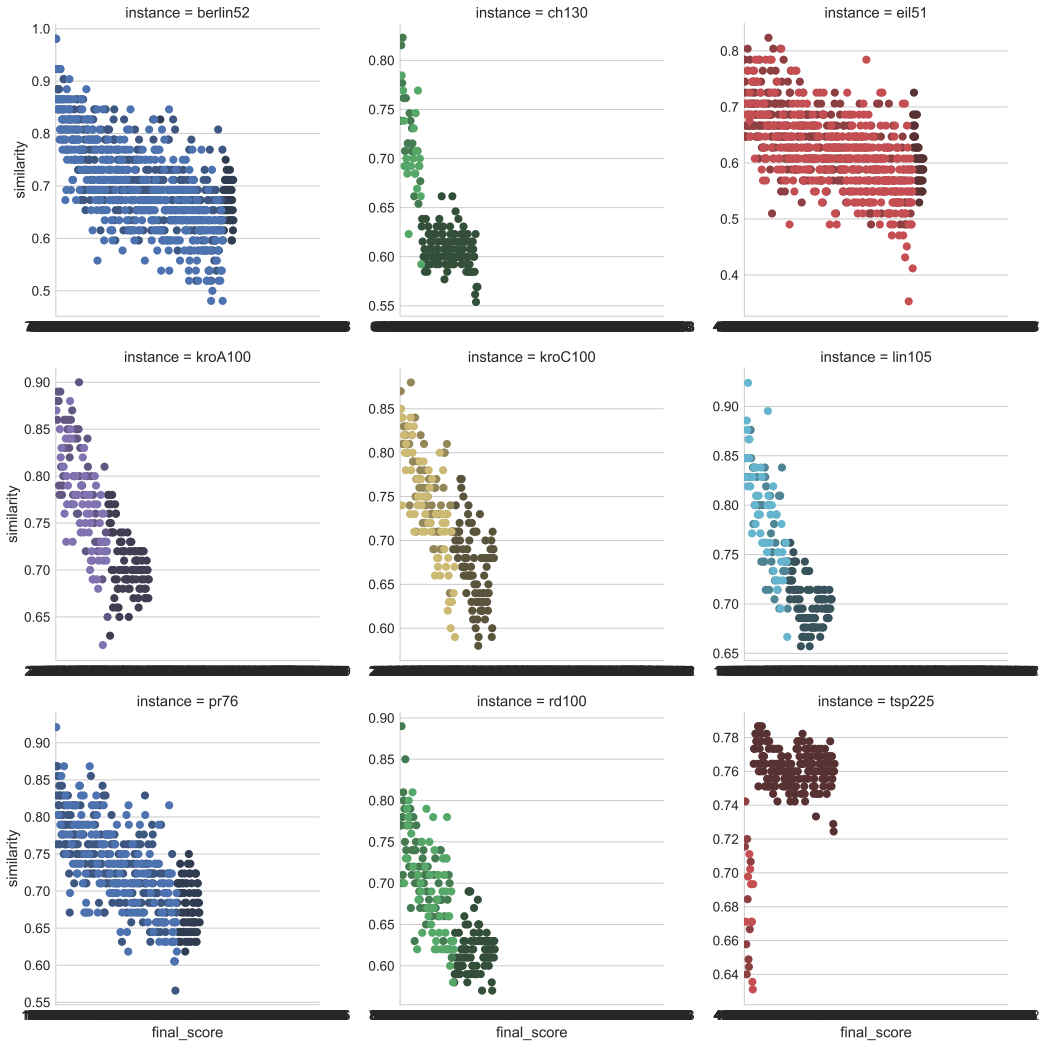
3.7.1 Miara odległości rozwiązań od rozwiązania optymalnego

Aby porównywać między sobą rozwiązania, postanowiliśmy badać, jak wiele mają takich samych krawędzi. Aby to zmierzyć, dla każdej krawędzi w jednym rozwiązaniu, sprawdzamy, czy istnieje ona w drugim (skierowana w dowolną stronę, ponieważ obie krawędzie są symetryczne).

3.7.2 Wyniki

Na rysunku 14 można zaobserwować podobieństwo rozwiązań znajdowanych przez algorytmy do rozwiązania optymalnego. Istnieje wyraźna zależność polegająca na tym, że im lepsze rozwiązanie, tym jest bardziej podobne do optymalnego.

Bardzo mocno wyróżnia się chmura rozwiązań losowych — jakość rozwiązań jest bardzo różnorodna, ale wszystkie są bardzo mało podobne do rozwiązania optymalnego (poniżej 10%, a im więcej miast w instancji, tym mniej podobne).



Rysunek 14: Porównanie odległości znajdowanych rozwiązań przez algorytmy od rozwiązania optymalnego.

4 Podsumowanie

4.1 Wnioski

W problemie symetrycznego komiwojażera, stosunkowo łatwo znaleźć dość dobre rozwiązanie za pomocą prostej heurystyki, którą też można jeszcze udoskonalić.

Algorytmy przeszukiwania lokalnego przeważnie znajdują jednak jeszcze lepsze rozwiązania, nieraz nawet optymalne (przy instancjach o liczności ok. 50 miast), pracując wielokrotnie dłużej, jednak wciąż nie dłużej niż minutę dla wybranych instancji. Algorytmy przeszukiwania lokalnego mimo, że zaczynają ze stosunkowo słabym rozwiązaniem losowym, wciąż poszukując lepszych w jego sąsiedztwie, potrafią osiągać bardzo dobre wyniki.

Porównując algorytmy Greedy i Steepest, można zauważyć, że ostatecznie oba zwracają podobnej jakości rozwiązania. Steepest wykonuje kilkakrotnie mniej kroków, jednak w każdym z nim musi przeszukać całe sąsiedztwo aktualnego rozwiązania, w konsekwencji czego, sumarycznie, przeszukuje większą przestrzeń rozwiązań i trwa dłużej od algorytmu Greedy. Zaobserwowaliśmy, że ta tendencja jest odwrotna dla dla największej, wybranej instancji.

4.2 Trudności

Jednym z głównych problemów, jakie wystąpiły podczas prezentacji wyników, było odpowiednie dobranie wykresów. Mimo, że 3 z 4 prezentowanych algorytmów dawało zbliżone wyniki, to random zawsze znacznie się od nich różnił, przez co trudno było tak wyskalować wykresy, aby wyraźnie było widać wszystkie zależności.

Innym problemem jest odpowiednie dobranie parametrów, aby każdy algorytm został uruchomiony odpowiednią liczbę razy dla każdej instancji, ale jednocześnie, by generowanie obliczenia nie trwały zbyt długo.

4.3 Propozycje udoskonaleń

Warta zbadania na pewno jest sytuacja zaobserwowana dla największej instancji, w której Steepest okazuje się być lepszy od algorytmu Greedy pod względem czasu wykonania, przegląda też mniejszą liczbę rozwiązań. Warto sprawdzić, czy ta tendencja utrzymuje się dla przykładów z większą liczbą miast.

Literatura

[1] Universität Heidelberg. Discrete and combinatorial optimization. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

[2] Maciej Komosiński. Materiały do wykładu „Metaheurystyki i obliczenia inspirwane biologicznie". Lecture notes, 2014.

[3] Christian Nilsson. Heuristics for the traveling salesman problem. https://web.tuke.sk/fei-cit/butka/hop/htsp.pdf.