Sprawozdanie z laboratorium: Metaheurystyki i Obliczenia Inspirowane Biologicznie

Część II: Algorytmy optymalizacji lokalnej i globalnej, problem STSP

12 grudnia 2017

Prowadzący: dr hab. inż. Maciej Komosiński

Autorzy:	**Patryk Gliszczyński**	inf117288	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 [4].

Patryk Gliszczyński

PG zaimplementował algorytmy lokalnego przeszukiwania, przygotował środowisko badawcze, napisał skrypt do automatycznego generowania wykresów, oraz opisał teoretyczną stronę tego zagadnienia.
PG zaimplementował i opisał algorytm symulowanego wyżarzania.

Mateusz Ledzianowski

ML zaimplementował funkcję mierzącą czas, algorytm losowy i prostą heurystykę, przeprowadził eksperymenty na różnych instancjach problemu, oraz opisał wnioski i spostrzeżenia z przeprowadzonych badań.
ML zaimplementował i opisał algorytm przeszukiwania tabu w wersji master list.

1 Symetryczny problem komiwojażera (STSP)

1.1 Opis problemu

Problem komiwojażera modeluje sytuację znaną ze świata rzeczywistego, w której pewien obiekt stara się odwiedzić wszystkie punkty z danego zbioru punktów oraz wrócić do miejsca początkowego, w kolejności minimalizującej całkowity koszt podróży. Z tego typu zadaniem mierzą się między innymi wszelkiego rodzaju dostawcy usług transportowych, listonosze, czy akwizytorzy. W symetrycznym problemie komiwojażera koszt pomiędzy dowolnymi dwoma wierzchołkami w grafie połączeń jest identyczny w obydwie strony. W naszym problemie sieć połączeń pomiędzy poszczególnymi wierzchołkami opisana jest grafem pełnym, co oznacza że istnieje połączenie między każdą parą wierzchołków w grafie.

1.2 Złożoność

W tak postawionym problemie istnieje $n!$ różnych rozwiązań, gdzie n oznacza liczbę wierzchołków w grafie. Punkty możemy odwiedzać w dowolnej kolejności, zatem jeżeli zostaną one ponumerowane od 1 do n, to 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 punktu startowego. Przestrzeń rozwiązań jest bardzo duża, oraz kosztowna obliczeniowo przy chęci wyznaczenia każdej istniejącej permutacji. 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 rozwiązanie można reprezentować w postaci permutacji, możliwym jest wygenerowanie rozwiązania losowego poprzez przeprowadzenie losowej permutacji wierzchołków z rozwiązania początkowego. Złożoność obliczeniowa takiej operacji wynosi $\theta(n)$, gdzie n jest długością rozwiązania początkowego. Aby wygenerować losową permutację należy zastosować poniższą procedurę:

1. Wypełnij tablicę liczbami od 1 do n.
2. $i := n$. – zainicjuj zmienną wskazującą na ostatni element rozwiązania.
3. $random := random(0,\ i-1)$ – wylosuj liczbę z zakresu od 0 do $i-1$ włącznie.
4. Zamień element z pozycji $i-1$ z elementem na pozycji wskazywanej przez zmienną $random$.
5. $i := i-1$.
6. Jeżeli $i > 1$, wróć do kroku 3.

Aby algorytmowi losowemu dać większe szanse na znalezienie lepszego rozwiązania, powtarzamy go 1 000 000 razy i zwraca on najlepsze spośród znalezionych rozwiązań.

1.4 Heurystyka

Dla symetrycznego problemu komiwojażera możliwe jest znalezienie prostej heurystyki o złożoności $\theta(n^2)$ dającej satysfakcjonujące rezultaty, przeciętnie odległe od rozwiązania optymalnego o $10-15\%$ [5]. W celu wyznaczenia rozwiązania z wykorzystaniem algorytmu heurystycznego, należy postępować zgodnie z następującymi krokami:

1. Wybierz losowy wierzchołek startowy x i dodaj go do rozwiązania.
2. Znajdź najbliższy nieodwiedzony wierzchołek y względem punktu x.
3. Dodaj wierzchołek y do rozwiązania i ustaw wartość zmiennej x na y.
4. Jeżeli długość rozwiązania jest mniejsza niż oczekiwana, wróć do punktu 2.
5. Wróć do wierzchołka początkowego.

1.5 Instancje problemu

Problem komiwojażera jest bardzo popularny i możliwe jest znalezienie wielu gotowych instancji, których można użyć w celu przeprowadzenia badań. W naszych eksperymentach wybraliśmy 9 instancji z udostępnionego przez uniwersytet w Heidelberg zbioru [2]. Są to:

1. eil51,
2. berlin52,
3. pr76,
4. kroA100,
5. kroC100,
6. rd100,
7. lin105,
8. ch130,
9. tsp225.

Przy doborze konkretnych instancji zależało nam na tym, aby nie były one zbyt duże (do ok. 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.

3 Symulowane wyżarzanie

3.1 Opis

Algorytm symulowanego wyżarzania jest heurystyczną metodą przeszukującą przestrzeń alternatywnych rozwiązań w celu znalezienia najlepszego rozwiązania, czyli takiego będącego najbliżej optimum pod względem wartości funkcji kosztu. Polega on na iteracyjnym wyborze losowych rozwiązań z przestrzeni lokalnych solucji i akceptacji ich w przypadku gdy zmniejszają one wartość funkcji kosztu, bądź spełniają odpowiednie warunki probabilistyczne degradowane wraz z upływem kolejnych iteracji. Algorytm opisujący kolejne kroki tej metody został przedstawiony poniżej.

1. Wygeneruj temperaturę startową T_0, rozwiązanie początkowe $x = x_{init}$, liczbę kroków L co które aktualizowana jest temperatura, oraz stopień redukcji temperatury $\alpha \, \epsilon \, (0, 1)$.
2. Powtarzaj dopóki warunek stopu nie został spełniony:
    2.1. Powtarzaj L razy:
        2.1.1. Generuj losowe rozwiązanie x' w sąsiedztwie rozwiązania x.
        2.1.2. Jeżeli nowe rozwiązanie jest lepsze od poprzedniego $f(x') <= f(x)$, lub warunek wyboru gorszego rozwiązania zostanie spełniony:
            2.1.2.1. Zaakceptuj nowe rozwiązanie $x = x'$.
    2.2. Aktualizuj temperaturę $T_k = T_{k-1} * \alpha$.

Powyżej przedstawiony generyczny algorytm może zostać wykorzystany w wielu scenariuszach optymalizacyjnych. Dla zadania postawionego w tym zadaniu należało odpowiednio dostroić powyższy algorytm, tak aby najskuteczniej radził sobie w znajdowaniu rozwiązania dla problemu STSP.

3.2 Dobór temperatury początkowej

Jednym z ważniejszych kroków w procesie symulowanego wyżarzania jest dobór odpowiedniej temperatury początkowej. Wartość ta decyduje w dużej mierze o czasie zbieżności do rozwiązania finalnego oraz liczbie zaakceptowanych gorszych lokalnie rozwiązań, które mogą prowadzić do nieoczekiwanej poprawy jakości rozwiązania w ujęciu globalnym. W naszej implementacji zdecydowaliśmy się dobierać ten parametr w sposób adaptatywny, ściśle zależny od instancji problemu oraz wygenerowanego pierwszego rozwiązania. Wartość ta wyliczana jest zgodnie z poniższym wzorem.

$$T_0 = \frac{\sum_{k=0}^{K} f(G(x)) - f(x)}{K * \ln \theta} \tag{1}$$

W powyższym wzorze, wektor x jest oryginalnym wygenerowanym rozwiązaniem, funkcja $G(x)$ generuje nowe rozwiązanie x' w sąsiedztwie rozwiązania x, a funkcja $f(x)$ zwraca wartość kosztu danego rozwiązania. Parametr K określa liczbę rozwiązań które chcemy wygenerować, a θ procent akceptowanych ruchów. Tak przedstawiona metoda pozwala na lepszy dobór temperatury początkowej w odniesieniu do konkretnej instancji problemu. W postaci przedstawionej powyżej jest ona jednak wrażliwa na jakość wygenerowanego rozwiązania początkowego i może nie zawsze być prawidłowo dobrana. Wartość ta może być również obliczana rekurencyjnie biorąc pod uwagę szersze spektrum rozwiązań co pozwala na dokładniejszy dobór parametru [1].

3.3 Warunek akceptacji rozwiązania

Rozwiązanie x' generowane w każdej iteracji algorytmu zostaje zaakceptowane tylko w przypadku, gdy jest ono nie gorsze od rozwiązania poprzedniego $f(x') <= f(x)$, lub spełniony jest warunek probabilistyczny:

$$e^{-\frac{f(x')-f(x)}{T_k}} > random(0,1) \tag{2}$$

Dzięki takiemu podejściu, pozwalamy algorytmowi wybrać rozwiązanie gorsze z pewnym prawdopodobieństwem zależnym od aktualnej wartości temperatury. Może to mieć pozytywny skutek i pozwolić nam wyjść z lokalnego optimum, w którym moglibyśmy utknąć gdybyśmy heurystycznie dążyli tylko ku lokalnej poprawie. Należy zauważyć, iż poprzez degradacje wartości temperatury w kolejnych iteracjach, prawdopodobieństwo wybrania gorszego rozwiązania maleje.

3.4 Warunek stopu

Istotą każdego algorytmu optymalizacyjnego jest dobrze zdefiniowany warunek stopu. W naszej implementacji zdecydowaliśmy się kończyć przeszukiwanie przestrzeni rozwiązań, w przypadku gdy wartość temperatury T_k spadła poniżej minimalnej wartości temperatury T_ω, lub gdy w przeciągu pewnej liczby iteracji σ nie udało się poprawić jakości rozwiązania.

3.5 Parametry

Jakość algorytmu symulowanego wyżarzania zależy od wielu parametrów, które decydują o czasie zbieżności do rozwiązania finalnego, liczbie przejrzanych lokalnie rozwiązań, czy szansie na akceptacje rozwiązań gorszych. Metodą prób, błędów i analizy dobraliśmy zestaw parametrów, które pozwalają w relatywnie krótkim czasie znajdować rozwiązania bliskie optimum. Poniżej przedstawione zostały wartości parametrów przedstawionych w tej sekcji.

- $\alpha = 0.9$ – stopień redukcji temperatury w kolejnych iteracjach.
- $\theta = 0.95$ – procent akceptowanych ruchów przy doborze temperatury początkowej.
- $T_\omega = 0.1$ – minimalna wartość temperatury kończąca algorytm.
- $K = 1000$ – liczba powtórzeń generowania rozwiązania przy doborze temperatury.

4 Przeszukiwanie tabu

4.1 Opis

Tabu Search jest drugim algorytmem optymalizacyjnym przedstawionym w sprawozdaniu, który pozwala na wychodzenie z optimów lokalnych poprzez dopuszczenie ruchów pogarszających rozwiązanie. Istnieją dwie popularne wersje tego rozwiązania – z listą tabu, czyli zakazanych ruchów oraz z elitarną listą kandydatów. Na potrzeby eksperymentu zdecydowaliśmy się zaimplementować tę drugą wersję.
Algorytm Tabu Search jest można opisać w następujący sposób[3]:

Algorithm .1: Integer division.

```
1  procedure PRZESZUKIWANIE TABU
2  begin
3    INICJALIZUJ(xstart, xbest, T)
4    x := xstart
5    repeat
6      GENERUJ(V ∈ N(x))
7      WYBIERZ(x0) //najlepsze f w V + aspiracja
8      UAKTUALNIJ LISTE TABU(T)
9      if f(x0) ⩽ f(xbest) then xbest := x0
10     x := x0
11   until WARUNEK STOPU
12 end
```

Po zainicjalizowaniu parametrów początkowych, należy powtarzać aż do osiągnięcia warunku stopu, którym w naszym rozwiązaniu jest określona liczba iteracji bez poprawy rozwiązania, kolejne kroki: najpierw należy ustalić listę kandydatów na ruch, następnie wybrać z niej ten, który prowadzi do najlepszego rozwiązania, usunąć wybrany krok z listy kandydatów, aby go nie powtórzyć w przyszłości, na końcu uaktualnić najlepsze znalezione dotychczas rozwiązanie, jeśli aktualne jest lepsze od dotychczas najlepszego.

4.2 Elitarna lista kandydatów

W naszym rozwiązaniu postanowiliśmy zastosować elitarną listę kandydatów. Polega ona na tym, że w momencie, kiedy dotychczas posiadana lista kandydatów jest niewystarczająca, co może się zdarzyć z dwóch powodów – jest pusta lub najlepszy kandydat na liście jest gorszy od najsłabszego kandydata w momencie generowania tej listy, należy stworzyć nową listę kandydatów.
Aby stworzyć nową listę kandydatów, należy przejrzeć całe sąsiedztwo aktualnego rozwiązania i wybrać spośród sąsiadów określoną listę najlepszych. Ruchy, które prowadzą do ich osiągnięcia tworzą nową elitarną listę kandydatów.
W następnych krokach algorytmu lista ta jest utrzymywana i sprawdza się tylko tych sąsiadów aktualnego rozwiązania, do których prowadzą ruchy z listy i wybiera się spośród nich najlepszy. Następnie ruch ten jest z listy usuwany, aby nie został powtórzony w przyszłości, aż do czasu wygenerowania nowej elitarnej listy kandydatów.
Dzięki takiemu rozwiązaniu, można znacznie ograniczyć liczbę przeszukiwanych sąsiadów każdego rozwiązania i wychodzić z optimów lokalnych – gdy nie ma rozwiązania, które poprawia wynik, też tworzy się listy kandydatów, które pogarszają go w najmniejszym stopniu.

4.3 Cechy algorytmu

Jedną z głównych cech algorytmu i zarazem przyczyn jego powstania jest dopuszczenie wychodzenia z maksimów lokalnych. Algorytm ten pozwala także na ograniczenie się do przeszukiwania jedynie części sąsiadów, zamiast całego ich zbioru, co znacznie przyspiesza obliczenia.

Jednak aby możliwe było pokonanie słabości algorytmów optymalizacji lokalnej, przeszukiwanie tabu naraża się na wpadanie w cykle. Trzeba też uważać, aby nie ograniczyć za bardzo jego ruchów.

4.4 Parametry

Nasza implementacja tabu search ma zasadniczo dwa główne parametry – wielkość elitarnej listy kandydatów k i liczba iteracji bez poprawy najlepszego rozwiązania do zakończenia algorytmu $P \cdot L$. Początkowo wyszliśmy od parametrów $k = sizeofinstance()/10$ i $P \cdot L = 10 \cdot sizeofinstance()$. Próbowaliśmy je ręcznie zmieniać i konfigurować, jednak nie poprawiało to znacząco wyników. Zauważyliśmy jedynie, że zmniejszając k lub $L \cdot P$, algorytm szybciej się kończył, często jednak z gorszym rezultatem.

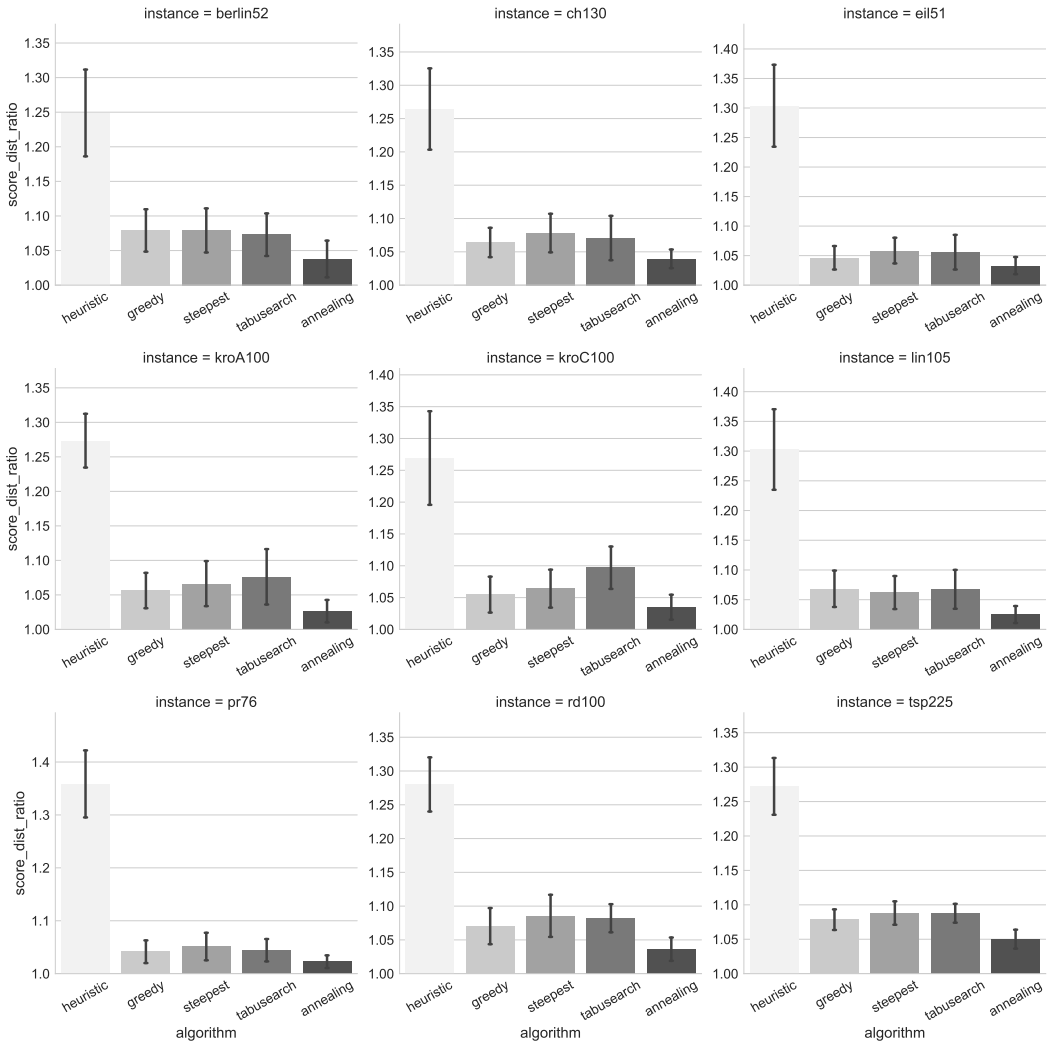
4.5 Podsumowanie

Algorytm przeszukiwania Tabu pozwala przezwyciężyć słabości algorytmów przeszukiwania lokalnego, poprzez wychodzenie z maksimów lokalnych i zapamiętywania dotychczasowego najlepszego rozwiązania. Ponadto w czasie swego działania sprawdza on dużo więcej rozwiązań, jednocześnie działając dłużej. Szybciej jednak podejmuje decyzję o wyborze następnego rozwiązania niż algorytm steepest, ponieważ nie przeszukuje całego sąsiedztwa.

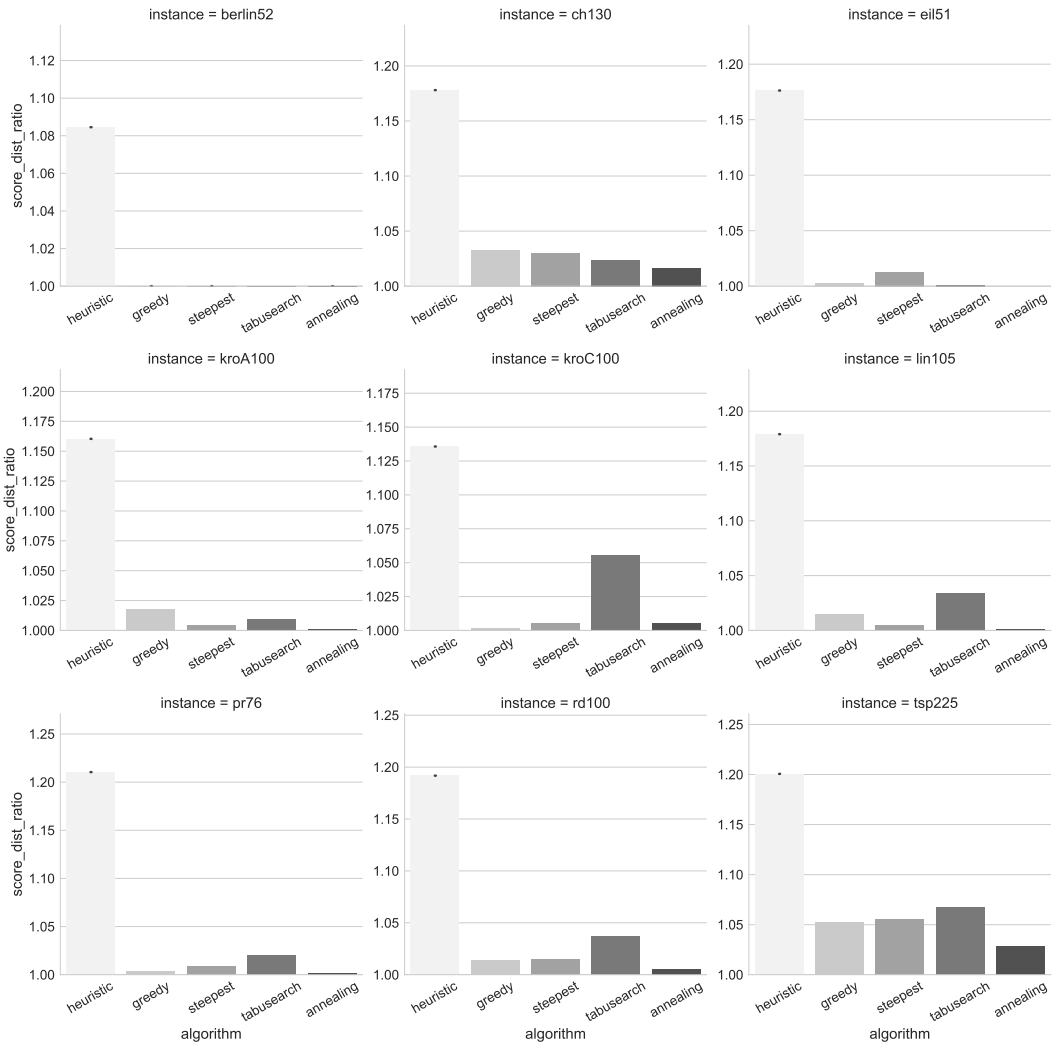
5 Eksperymenty

5.1 Odległość od optimum

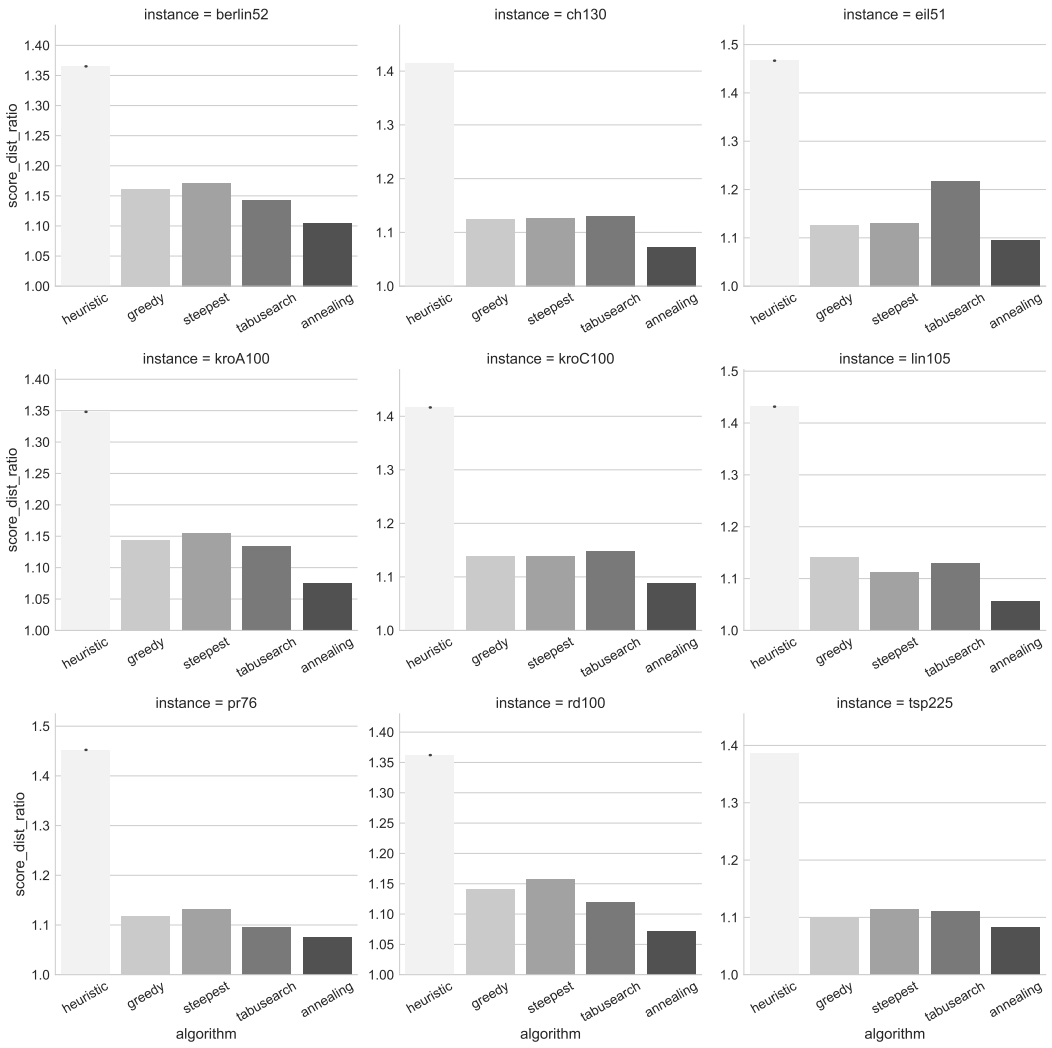
Rozwiązanie końcowe zaprezentowane na wykresach jest to bezwzględna suma odległości trasy komiwojażera.
Z Rys. 1 wynika, że dla każdej instancji problemu, heurystyka i algorytmy przeszukiwania lokalnego osiągają podobne wyniki, które są kilkukrotnie lepsze od rozwiązania losowego, z którego startują 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.



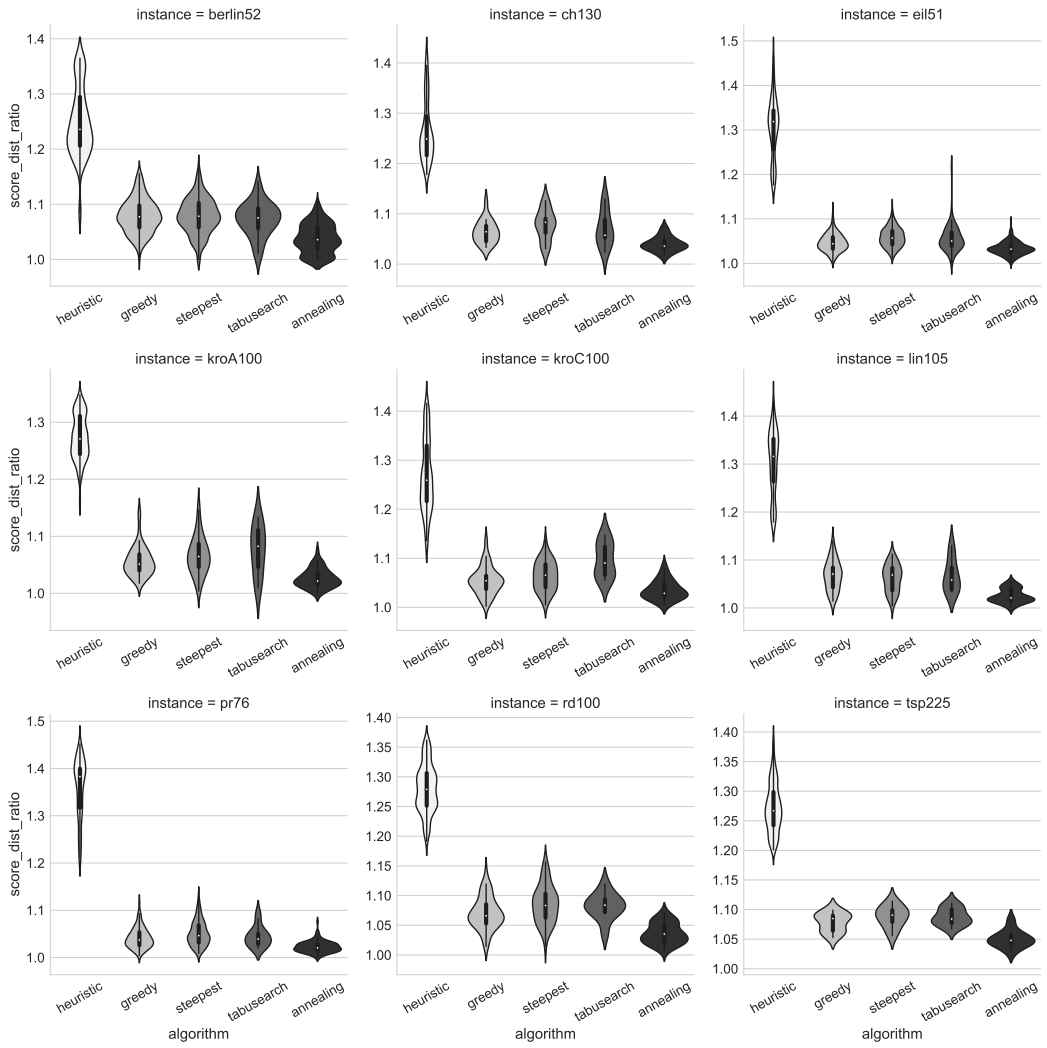
Rysunek 1: Porównanie średnich rozwiązań na różnych instancjach.



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



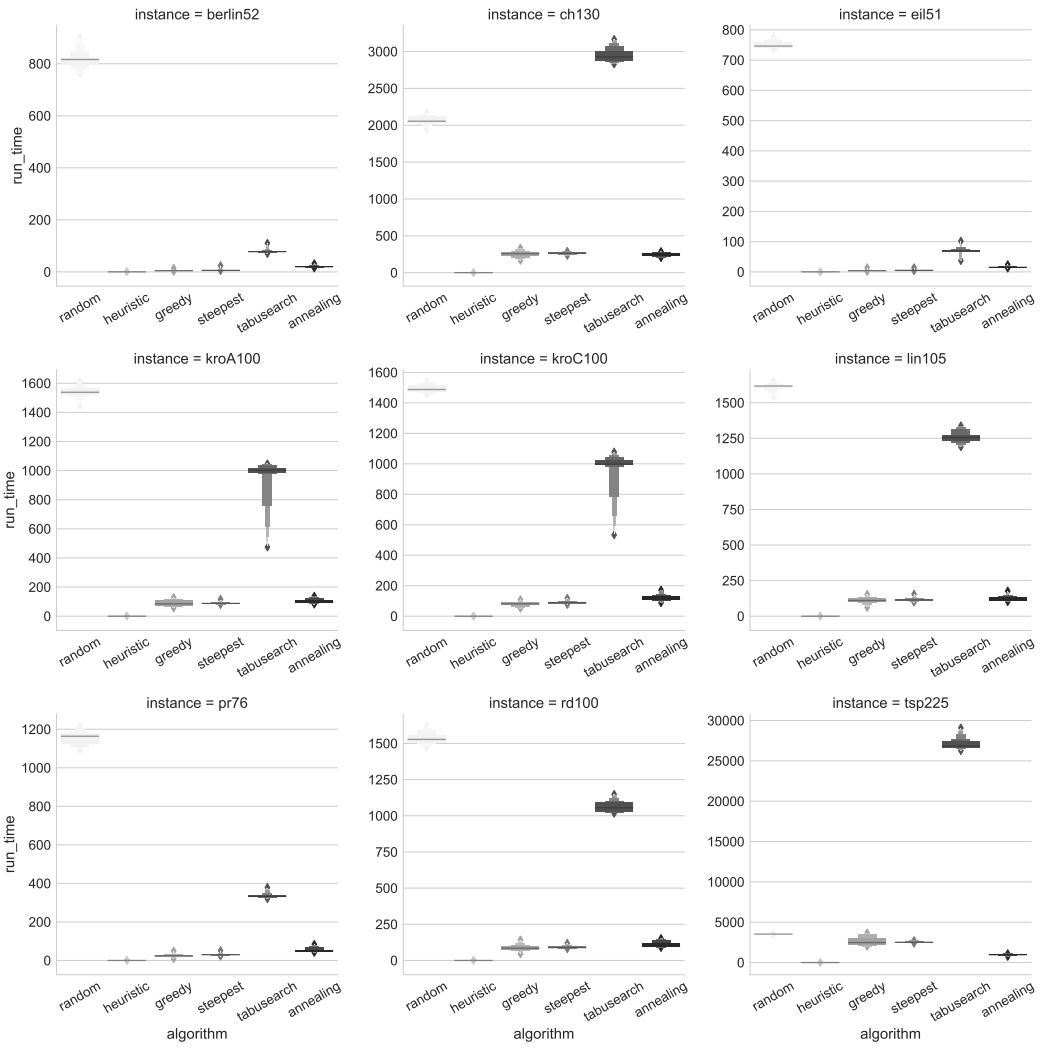
Rysunek 3: Porównanie najgorszych znalezionych rozwiązań przez algorytmy na różnych instancjach.



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

5.2 Czas działania

Heurystyka jest zdecydowanie najszybsza, ponieważ sprawdza tylko jedno rozwiązanie. 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. Algorytm losowy zawsze sprawdza określoną liczbę rozwiązań, więc czas jego działania jest stały.



Rysunek 5: Porównanie czasu działania algorytmów na poszczególnych instancjach.

5.3 Efektywność algorytmów

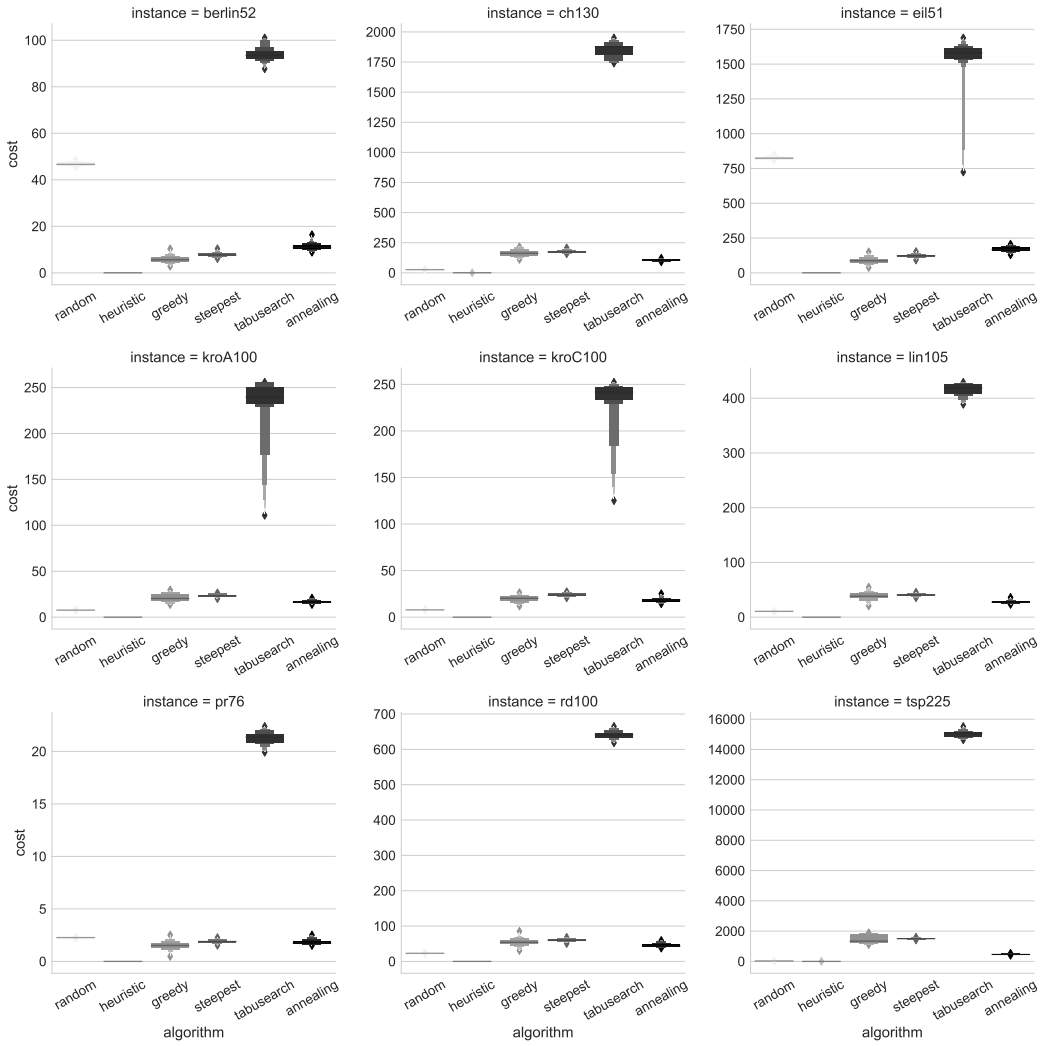
5.3.1 Wybrana miara

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.

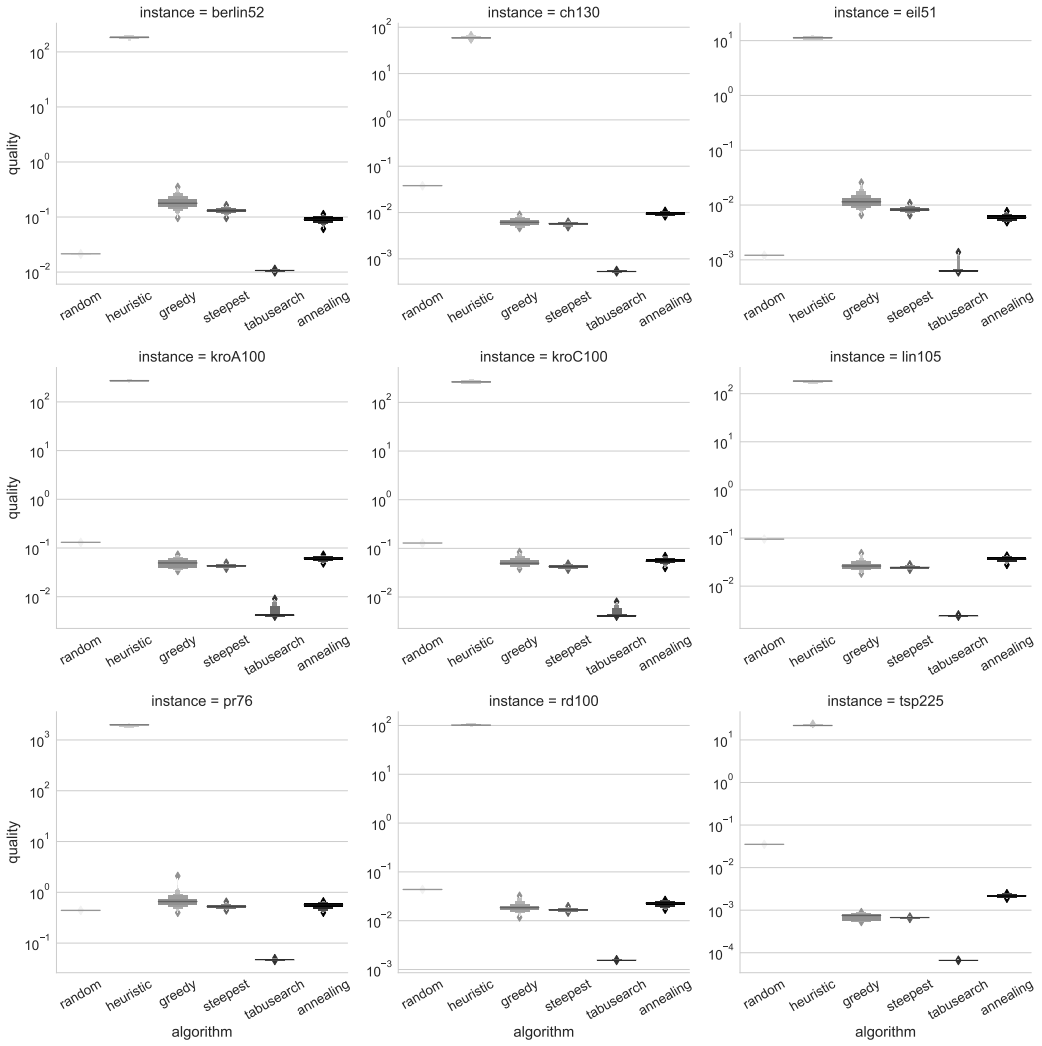
5.3.2 Wyniki

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.



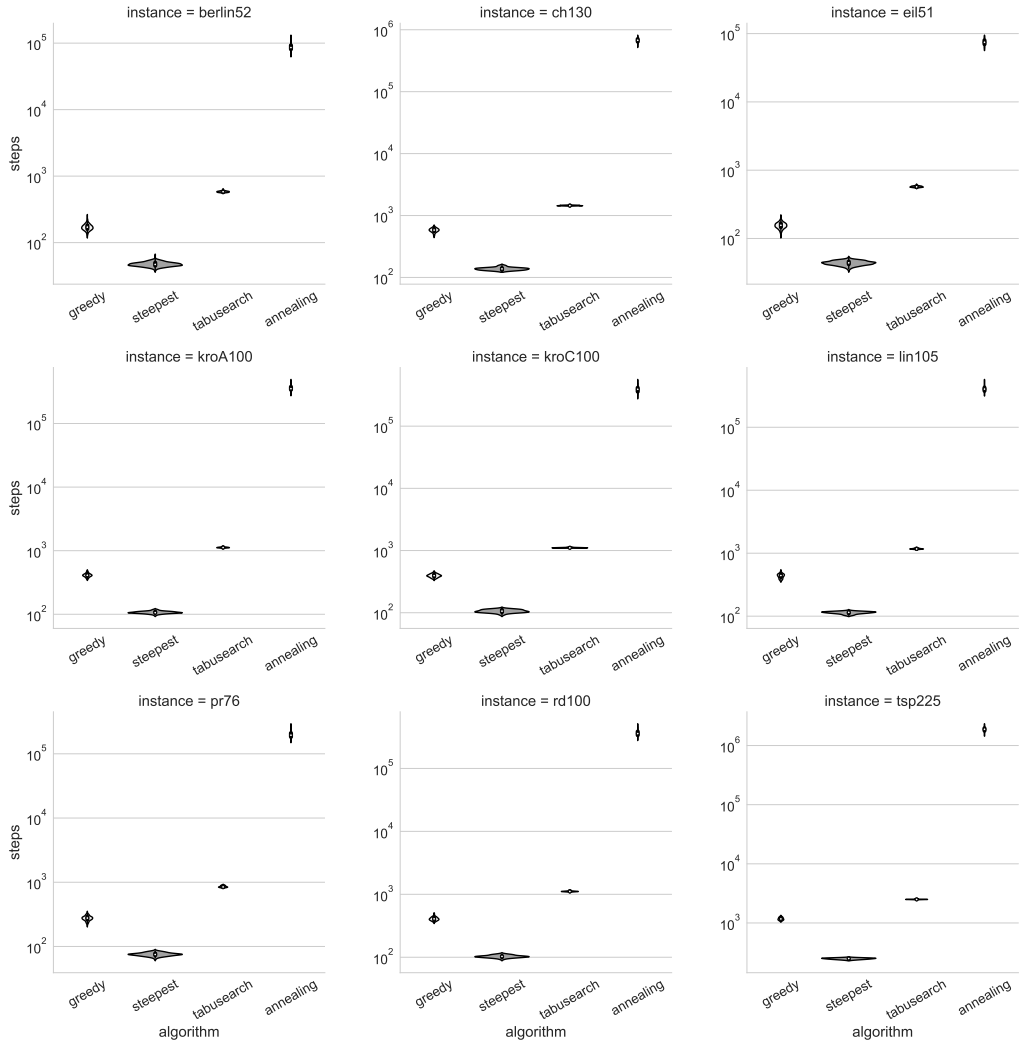
Rysunek 6: Porównanie kosztów algorytmów na poszczególnych instancjach.



Rysunek 7: Porównanie efektywności algorytmów na poszczególnych instancjach.

5.4 Liczba kroków algorytmów lokalnego przeszukiwania

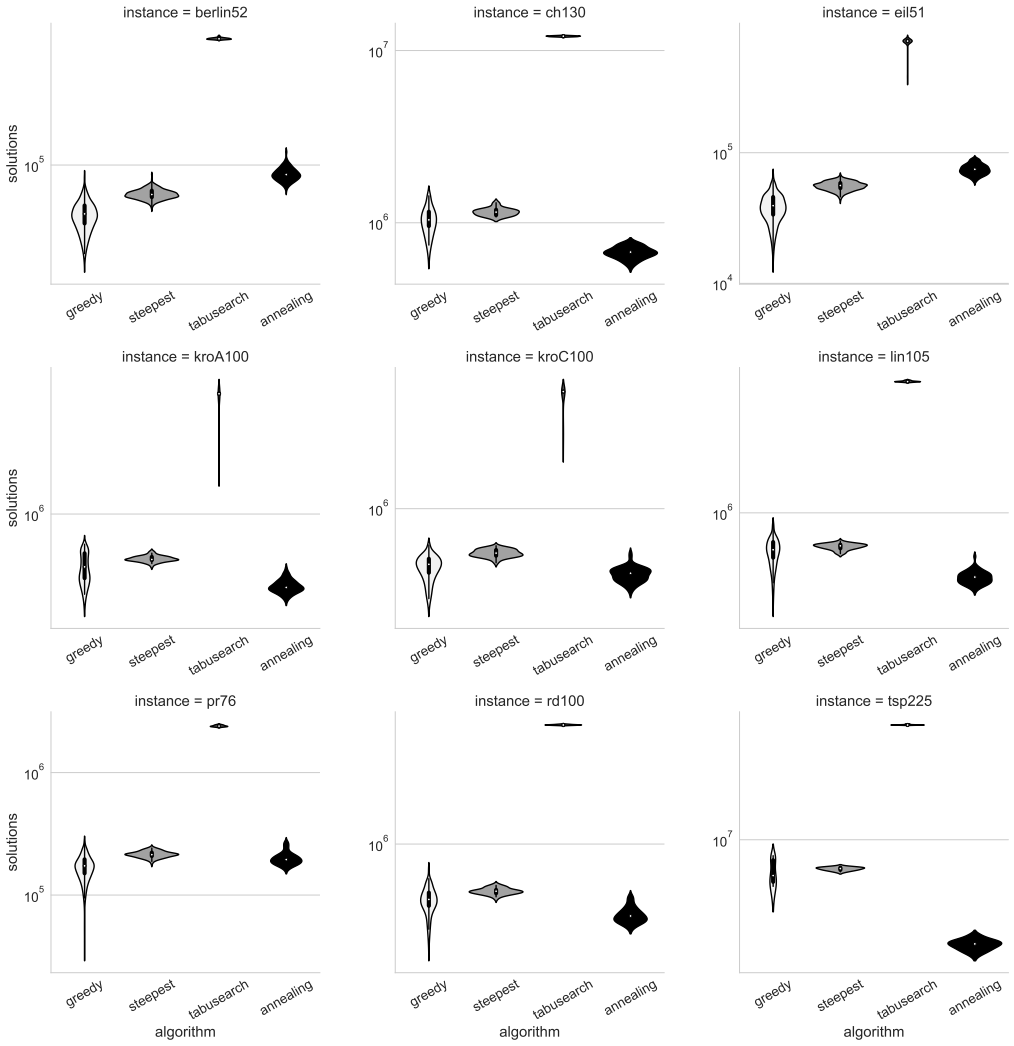
Wykres 8 przedstawia liczbę kroków wykonanych przez algorytmy lokalnego przeszukiwania. Widać, że Greedy wykonuje ich kilkakrotnie więcej niż Steepest, a także, że liczba wykonanych kroków przez ten algorytm jest dużo bardziej zróżnicowana, w zależności od przypadku startowego. Jest to spowodowane tym,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



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

5.5 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.

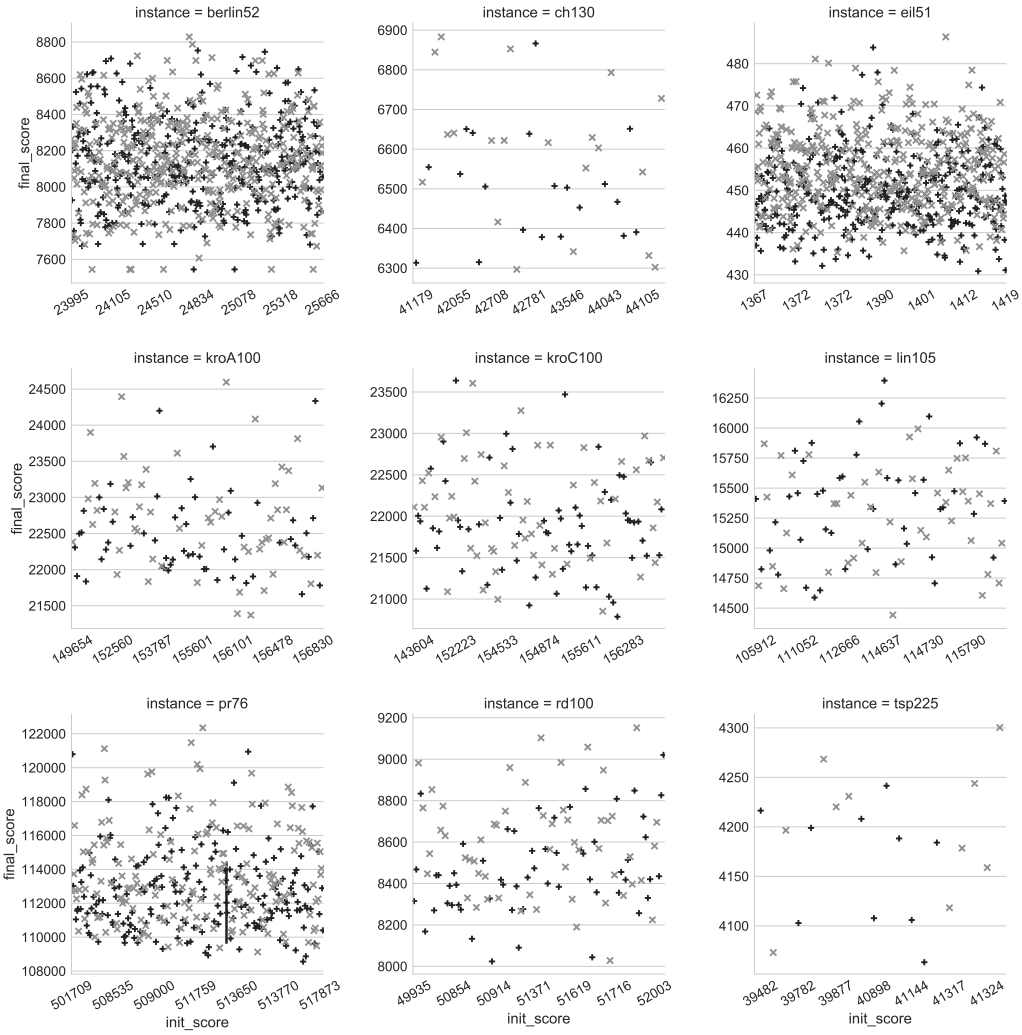


Rysunek 9: Porównanie algorytmów Greedy Search i Steepest pod względem liczby przeszukanych rozwiązań.

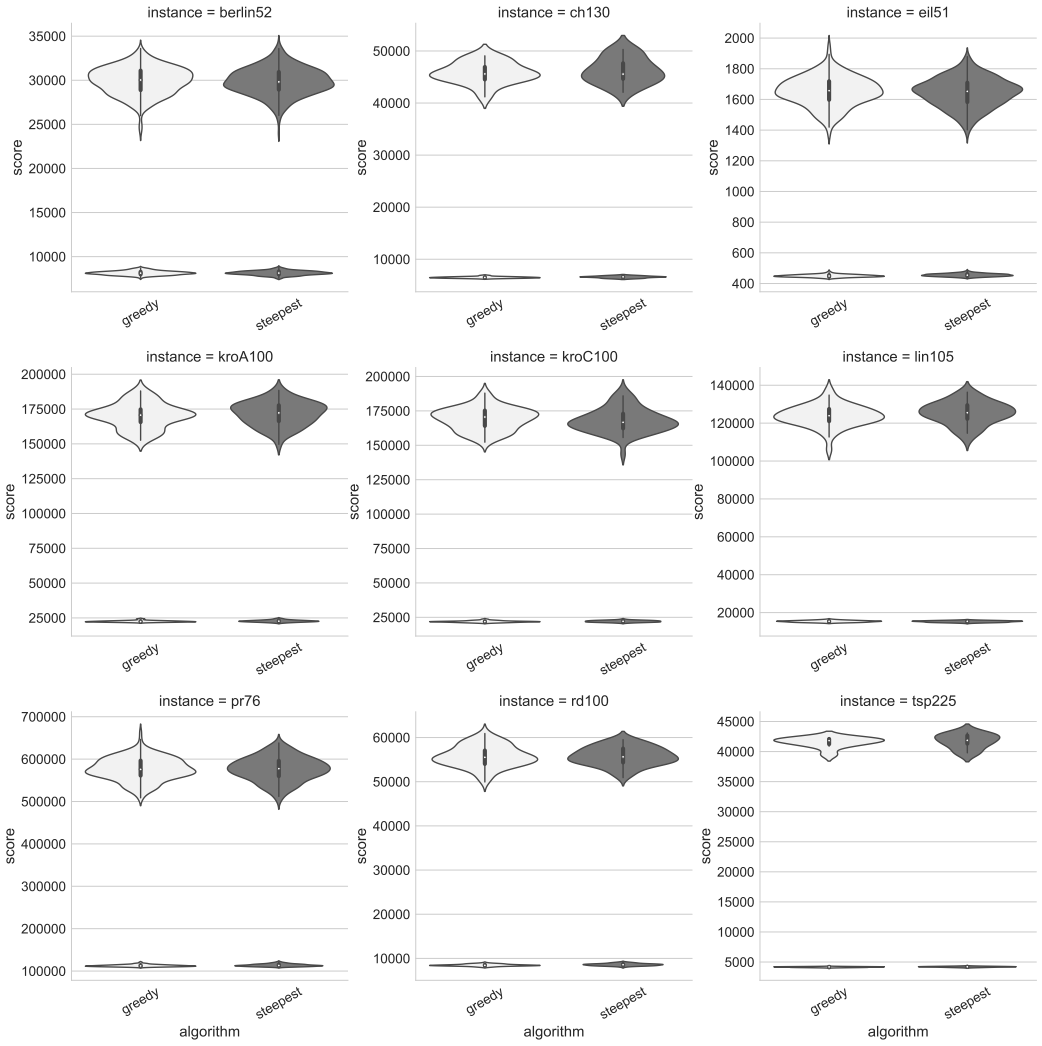
5.6 Przeszukiwanie lokalne

5.6.1 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.



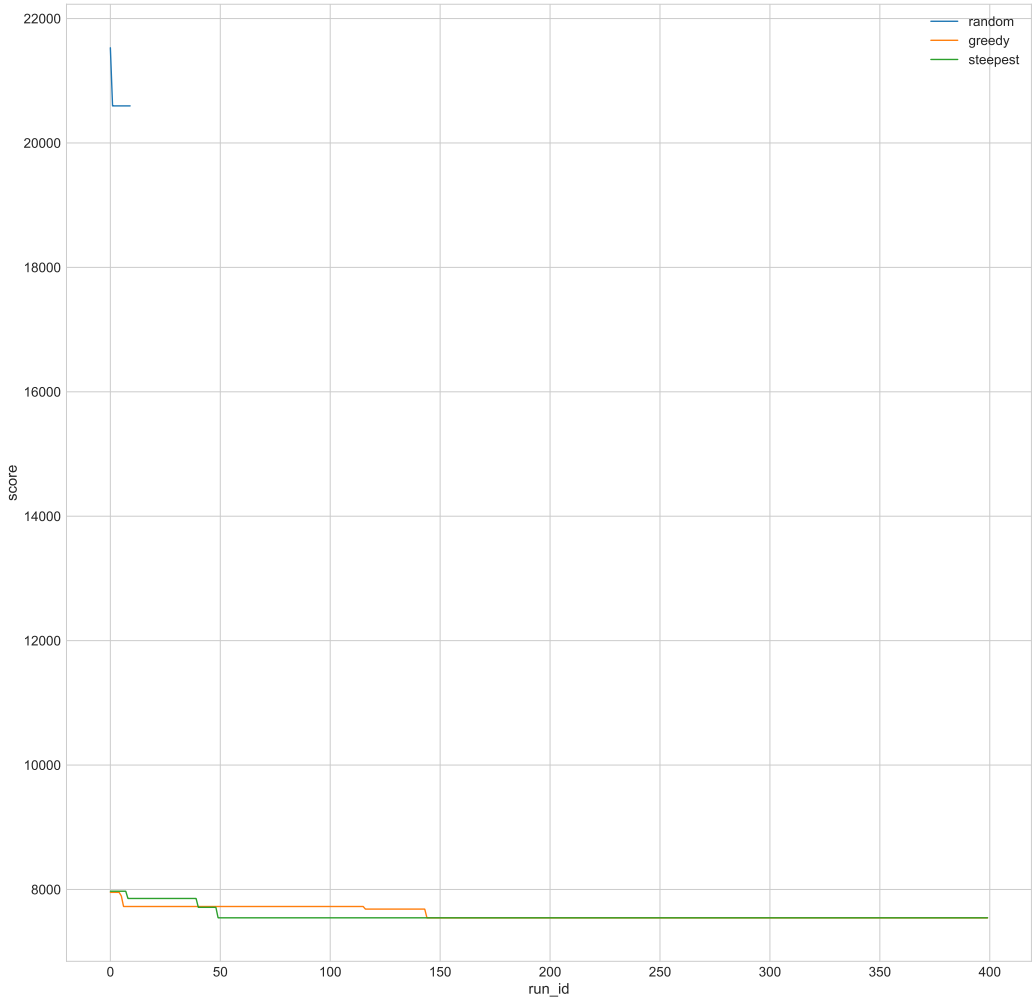
Rysunek 10: Porównanie jakości rozwiązań początkowych i końcowych przez algorytmy Greedy Search i Steepest przedstawione na wykresie punktowym.



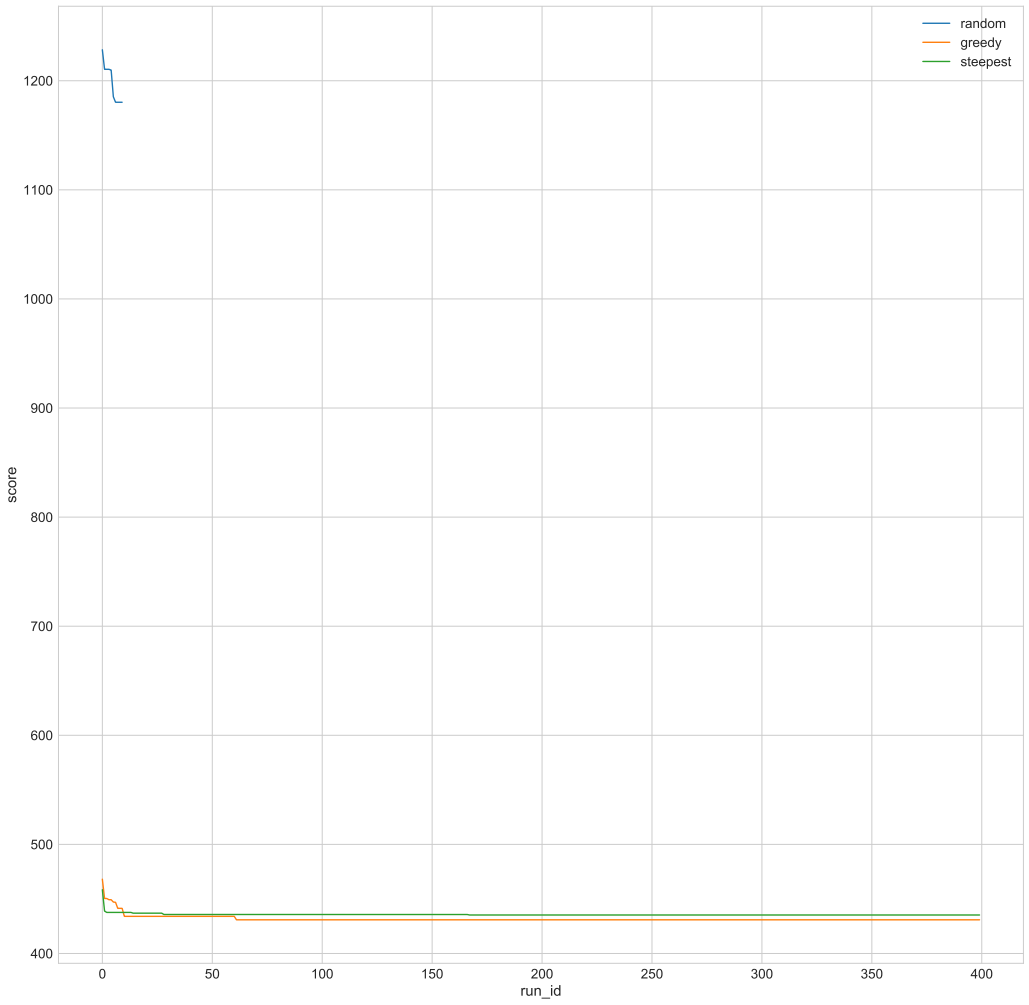
Rysunek 11: Porównanie jakości rozwiązań początkowych i końcowych przez algorytmy Greedy Search i Steepest.

5.6.2 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 12: Porównanie jakości rozwiązań algorytmów Gready Search i Steepest w zależności od liczby uruchomień tych algorytmów dla różnych rozwiązań początkowych dla zbioru berlin52.



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

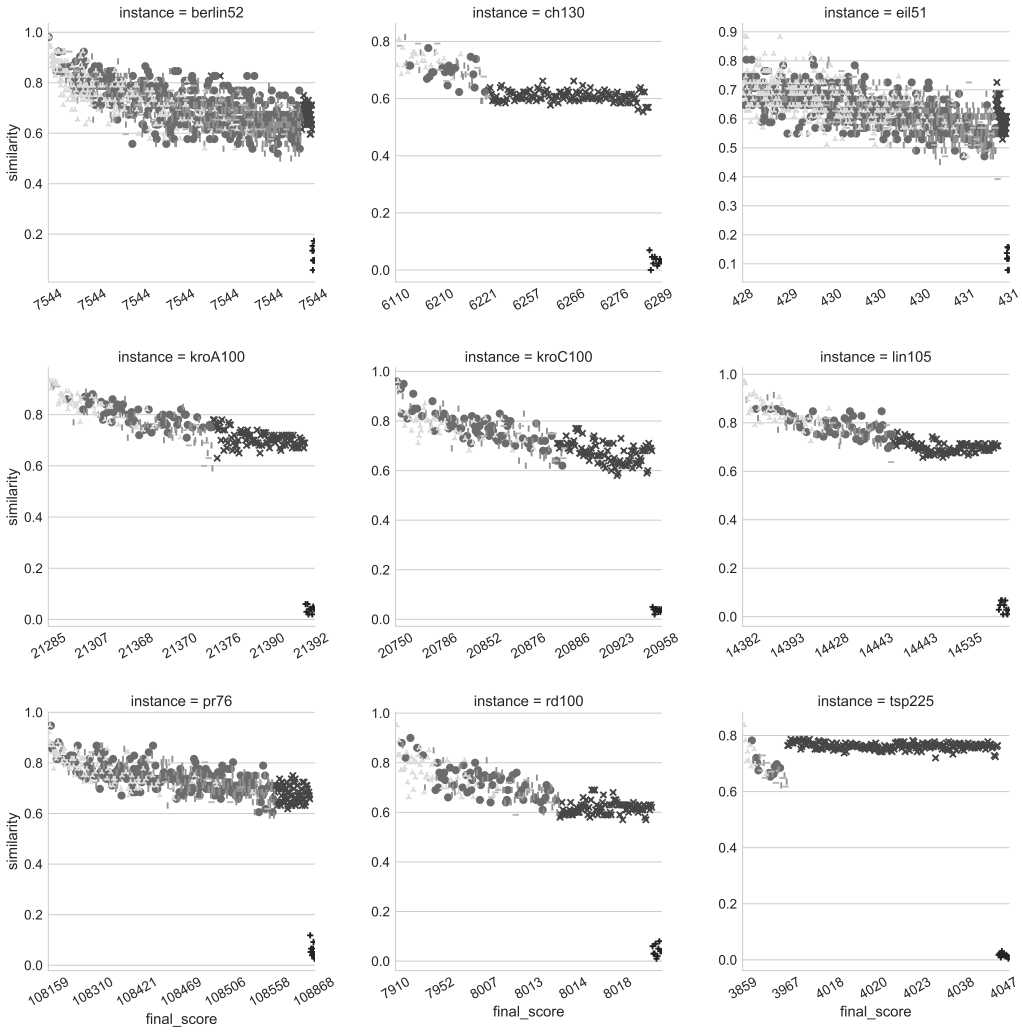
5.7 Porównanie rozwiązań

5.7.1 Miara odległości rozwiązań od rozwiązania optymalnego

Aby porównywać między sobą rozwiązania, postanowiliśmy zbadać, 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).

5.7.2 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.

6 Podsumowanie

6.1 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 największej, wybranej instancji.

6.2 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.

6.3 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.
Ponadto dobrze byłoby też sprawdzić, czy założenia podjęte odnośnie do sąsiedztwa 2-OPT są poprawne i rzeczywiście dla problemu symetrycznego komiwojażera lepiej jest odwracać całe łuki, a nie tylko zamieniać wierzchołki.
Ciekawy do rozważenia jest także wpływ rozwiązania początkowego na ostateczne wyniki algorytmów przeszukiwania lokalnego. Warto sprawdzić, czy gdyby nie startowały z rozwiązania losowego, ale np. z rozwiązania heurystycznego, ich skuteczność byłaby większa.

Literatura

[1] Walid Ben-Ameur. Computing the initial temperature of simulated annealing. 29:369–385, 12 2004.

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

[3] Maciej Komosiński. Prezentacja „optymalizacja. przeszukiwanie tabu". Lecture.

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

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