Algorytm genetyczny dla problemu komiwojażera (TSP)

2025-10-24

Problem komiwojażera TSP

Założenia dotyczące problemu TSP

Rozważany problem to klasyczny Problem Komiwojażera (Traveling Salesman Problem, TSP), w którym należy znaleźć najkrótszą możliwą trasę odwiedzającą każde miasto dokładnie raz i wracającą do punktu startowego.

- Odległości między miastami są znane i symetryczne, czyli droga z miasta A do B ma tę samą długość co z B do A.
- Wartości w macierzy odległości są nieujemne i odpowiadają rzeczywistym dystansom.
- Każde miasto musi być odwiedzone dokładnie jeden raz, a trasa jest cykliczna (powrót do punktu początkowego).

Funkcja celu

Główną metryką jest długość trasy (im mniejsza, tym lepsza).

Dane

Pliki wejściowe (macierze odległości):

- Dane_TSP_127.xlsx - 127 miast
- Dane_TSP_76.xlsx - 76 miast
- Dane_TSP_48.xlsx - 48 miast

Algorytm wykorzystuje bezpośrednio tę macierz do oceny tras.

Wstęp

Badamy algorytm genetyczny (GA) rozwiązujący problem komiwojażera (TSP). Celem jest porównanie wpływu kluczowych parametrów algorytmu na jakość znalezionych tras, przyjmując podejście one-factor-at-a-time (OFAT) czyli zmieniamy kolejno po jednym parametrze, a pozostałe pozostają stałe (`baseline`). Wyniki zapisujemy do pliku Excela i analizujemy wykresy.

Środowisko i ustawienia powtarzalności

- **Globalny seed**: `SEED = 42` jest ustawiany na początku działania skryptu.
- **Seed dla uruchomienia**: Dla każdego pojedynczego uruchomienia algorytmu, seed jest ustawiany deterministycznie (`SEED + run_id`), co pozwala na dokładne odtworzenie każdego eksperymentu.

Założenia dotyczące algorytmu genetycznego

Algorytm genetyczny jest metodą metaheurystyczną czyli nie gwarantuje znalezienia rozwiązania optymalnego, ale pozwala znaleźć rozwiązania bliskie optymalnym w rozsądnym czasie.

Rozwiązania (osobniki) są reprezentowane jako permutacje miast - kolejność odwiedzin miast określa trasę.

Populacja osobników ewoluuje przez wiele pokoleń (n_gen) w kierunku coraz krótszych tras.

Opis parametrów

- n_pop oznacza rozmiar populacji; populacja początkowa jest tworzona losowo, a potem każda generacja jest w pełni zastępowana przez potomków + ewentualnie elity
- n_gen oznacza liczbę generacji; im więcej generacji, tym dłużej trwa ewolucja i algorytm ma więcej szans na poprawę wyniku ale zbyt wiele generacji może nie przynosić dalszych korzyści
- p_mut czyli prawdopodobieństwo mutacji, pomaga uciekać z lokalnych minimów; szansa, że po utworzeniu dziecka, jego trasa zostanie losowo zmieniona;w tym przypadku polega na zamianie (swap) dwóch miast w trasie; nie gwarantuje poprawy rozwiązania; (!nie należy mylić mutacji z ulepszaniem local search (ruchami sąsiedzkimi) które działają już po mutacji i próbują celowo poprawić trasę, a nie losowo ją zmienić)
- p_cx czyli prawdopodobieństwo krzyżowania, częstotliwość rekombinacji rodziców; to szansa, że z dwóch rodziców powstanie nowy potomek, który łączy ich cechy; jeśli losowanie się nie powiedzie, potomek jest po prostu kopią jednego rodzica;
- selection_method czyli metody selekcji; decyduje kto zostanie wybrany jako rodzic do stworzenia potomka, wszystkie metody dążą do tego, by lepsi mieli większą szansę na rozmnożenie, ale każda robi to trochę inaczej:

1) tournament (selekcja turniejowa) - losuje k osobników z populacji, z tych k wybiera tego o najkrótszej trasie (najlepszy), pozwala kontrolować siłę selekcji przez tournament_k; im większe k, tym silniejsza selekcja (bo częściej wygra najlepszy z losowanych)
2) roulette (selekcja ruletkowa) - każdy osobnik dostaje „wagę" na ruletce proporcjonalne do swojej jakości (u nas odwrotnie do długości trasy); daje większe szanse dobrym trasom, ale też pozwala czasem „słabszym" wejść do gry (bo utrzymuje różnorodność)
3) rank (selekcja rankingowa) - najlepszy osobnik dostaje najwyższy „ranking", najgorszy najniższy; szansa na wybór zależy od pozycji w rankingu, a nie bezpośrednio od długości trasy.

- crossover_method czyli metody krzyżowania, różne sposoby zachowania porządku i fragmentów rodziców w permutacji, czyli w jaki sposób łączone są ich geny (kolejności miast):

1) PMX (Partially Mapped Crossover) - wybierany jest losowy fragment (np. 4-8 miast) z pierwszego rodzica i kopiowany do dziecka, reszta miast uzupełniana jest na podstawie drugiego rodzica, zgodnie z mapowaniem między fragmentami.

2) OX (Order Crossover) - fragment z jednego rodzica zostaje zachowany, pozostałe miasta są wypełniane w kolejności ich występowania w drugim rodzicu; dziecko dziedziczy „kolejność odwiedzin", a niekoniecznie dokładne pozycje.

3) CX (Cycle Crossover) - tworzy cykle powiązań między genami rodziców (np. miasto A u pierwszego jest w tym samym miejscu co B u drugiego), naprzemiennie bierze cykle z rodzica 1 i 2.

- local_search_tries: liczba prób lokalnego ulepszenia; czyli liczba ruchów sąsiedzkich, które są losowo testowane na każdym dziecku, by sprawdzić, czy da się poprawić trasę:

1) swap - zamiana dwóch miast miejscami,

2) 2-opt - odwrócenie fragmentu trasy,

3) insertion - wyjęcie miasta i wstawienie go gdzie indziej

! Uwaga ! Zamiast testować ruchy swap, insertion i two_opt osobno, funkcja local_improve losowo wybiera jeden z nich w każdej próbie.

- elite (elitaryzm) oznacza liczbę najlepszych osobników kopiowanych bez zmian; chroni najlepsze rozwiązania, ale zbyt duża wartość elite mogłaby zmniejszyć różnorodność.

Usprawnienie mechanizm „wnuka" (grandchild)

Mechanizm „wnuka" (grandchild) to proste rozszerzenie standardowego schematu rozmnażania w algorytmie genetycznym. Po wygenerowaniu potomka (child) z pary rodziców istnieje pewne prawdopodobieństwo p_make_grandchild, że z potomka powstanie dodatkowy twór - „wnuk" (grandchild). Wnuk powstaje przez zastosowanie kolejnego, krótkiego operatora naprawczego (np. prostej mutacji lub szybkiego lokalnego ulepszania). Do populacji przyjmowany jest lepszy z pary (child vs grandchild). Intuicyjnie, zamiast przyjmować bezwarunkowo pojedynczego potomka, sprawdzamy jedną dodatkową opcję „bliźniaczą" i zachowujemy tę, która daje krótszą trasę.

Parametry:

- p_make_grandchild (= [0,1]) - prawdopodobieństwo wygenerowania wnuka dla każdego potomka,
- grandchild_method = {'mutate', 'local_improve'} - sposób wytworzenia wnuka: prosta mutacja (swap) lub krótki lokalny search (np. local_improve z ruchem two_opt albo mieszany),
- grandchild_local_tries - liczba prób dla lokalnego ulepszania wnuka (jeśli None, używany jest parametr local_search_tries).

Mechanizm wnuka daje prosty kompromis:

- zwiększa szansę na znalezienie natychmiastowej ulepszonej wersji potomka (przy niskim koszcie - jedno wywołanie mutacji lub krótkiego local search);
- jest prosty do kontrolowania przez pojedynczy parametr p_make_grandchild.

Typowe efekty obserwowane w eksperymentach:

- umiarkowane zwiększenie jakości końcowych rozwiązań (niższe średnie i lepsze mediany),
- niewielki wzrost wariancji (jeśli p_make_grandchild zbyt duże),

Jednak po analizie okazuje się, że żaden z najlepszych wyników nie miał włączonego mechanizmu wnuka.

Baseline (parametry domyślne)

baseline = { - ‘n_pop’: 100, - ‘n_gen’: 300, - ‘p_mut’: 0.03, - ‘p_cx’ : 0.9, - ‘selection_method’: ‘tournament’, - ‘crossover_method’: ‘pmx’, - ‘local_search_tries’: 5, - ‘local_moves’: [‘swap’,‘two_opt’,‘insertion’], - ‘elite’: 3, - ‘tournament_k’: 3, - ‘p_make_grandchild’: 0.0, - ‘grandchild_method’: ‘mutate’, - ‘grandchild_local_tries’: None }

Parametry testowane (OFAT)

Dla każdego parametru testujemy wartości:

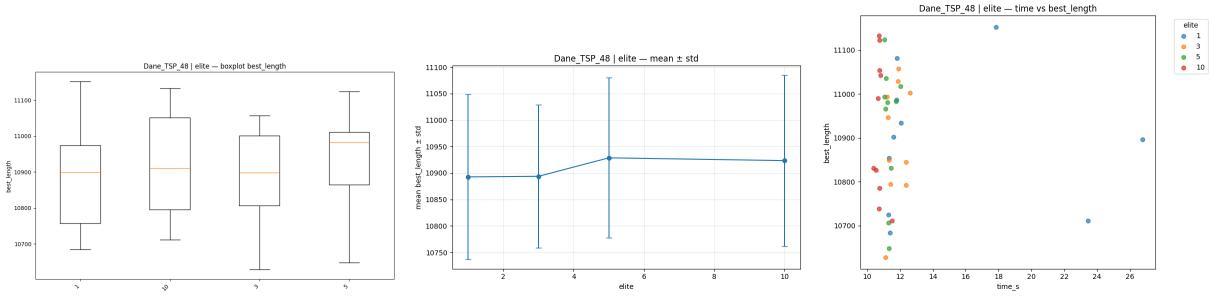
- n_gen : [100, 300, 600, 1200]
- n_pop : [50, 100, 200, 400]
- p_mut : [0.01, 0.03, 0.07, 0.15]
- p_cx : [0.6, 0.75, 0.9, 1.0]
- selection_method : [‘tournament’, ‘roulette’, ‘rank’] (kategorialnie)
- crossover_method : [‘pmx’, ‘ox’, ‘cx’] (kategorialnie)
- local_search_tries : [0, 5, 20, 50]
- crossover_method‘: [‘pmx’, ‘ox’, ‘cx’],
- selection_method‘: [‘tournament’, ‘roulette’, ‘rank’]

Dla każdej konfiguracji powtarzamy uruchomienie repetitions razy 10.

Dokładna analiza wyników

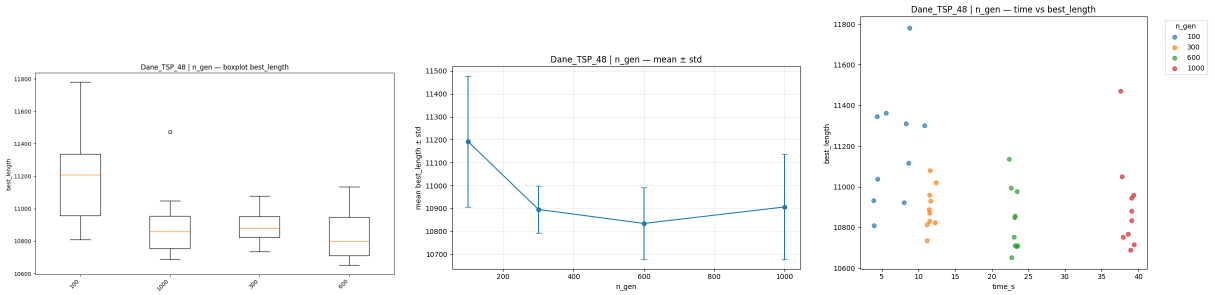
Dla 48 miast

parametr elite



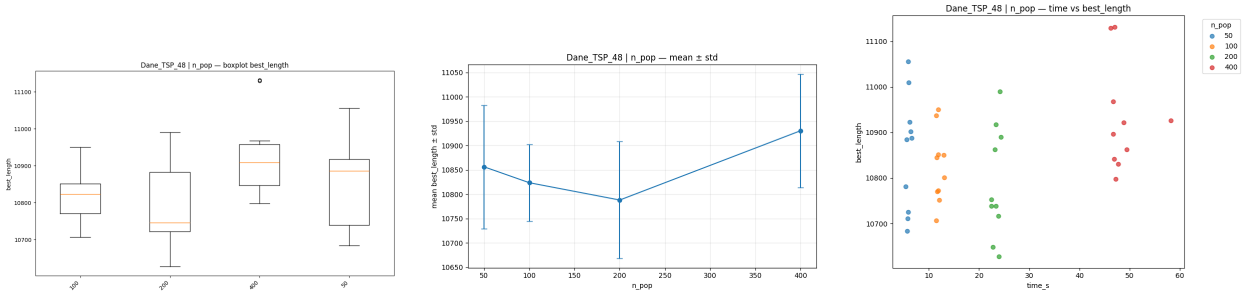
Wykres pudełkowy pokazuje, jak zmieniała się długość najlepszej trasy w zależności od liczby osobników elity. Dla wartości elite = 3 (na wykresie pudełka nie są po kolei) długości tras są mniejsze i bardziej stabilne niż dla pozostałych wartości. Dla zbyt dużej elity, wyniki są bardziej rozproszone, co oznacza większą losowość i gorszą powtarzalność. Możemy stwierdzić, że umiarkowana wartość elity (około 2-3) daje najlepszy kompromis między jakością a stabilnością wyników. Na drugim wykresie przedstawiamy średnią długość najlepszej trasy oraz odchylenie standardowe dla różnych wartości elity. Najniższa średnia wartość występuje przy elite = 3, więc mamy potwierdzenie, że ta konfiguracja pozwala osiągać najlepsze rezultaty. Dla większej elity, widać wyraźnie większe odchylenie standardowe, czyli mniejszą stabilność działania algorytmu. Zbyt duża liczba elitarnych osobników może utrudniać znalezienie optymalnej trasy.

parametr n_gen



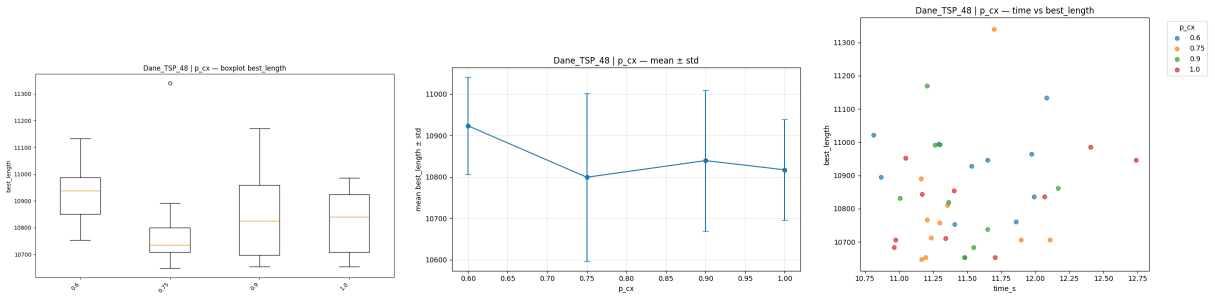
Wraz ze wzrostem liczby generacji obserwujemy wyraźną poprawę jakości rozwiązań czyli mniejszą długość najlepszej trasy. Na boxplocie widać, że przy małej liczbie generacji wyniki są gorsze i różne, natomiast przy 300-600 generacjach rozrzut jest mniejszy, a wartości lepsze. Średnie wartości na wykresie 2. potwierdzają, że około 300-600 generacji daje najniższe średnie długości tras. Dla 1000 generacji wynik nie poprawia się znacząco, natomiast czas obliczeń rośnie, co widać na wykresie 3. Zatem zwiększanie liczby generacji po pewnym progu przynosi coraz mniejsze korzyści, a tylko wydłuża działanie algorytmu. Optymalny kompromis między jakością a czasem obliczeń osiągamy przy około 300-600 generacjach.

parametr n_pop



Parametr liczby osobników w populacji wpływa na jakość rozwiązań i czas działania algorytmu. Na wykresach widać, że dla małych populacji wyniki są bardziej zmienne, ale średnia długość tras jest zbliżona do najlepszych przypadków. Przy populacji 200 uzyskano najniższe wartości długości tras. Dla większych populacji (jak 400), poprawa jakości nie jest już zauważalna, natomiast czas obliczeń znacznie wzrasta. Z wykresu 3. widać, że im większa populacja, tym dłuższy czas działania, ale nie zawsze idzie to w parze z lepszym wynikiem. Optymalna liczebność populacji w tym przypadku to około 200 osobników.

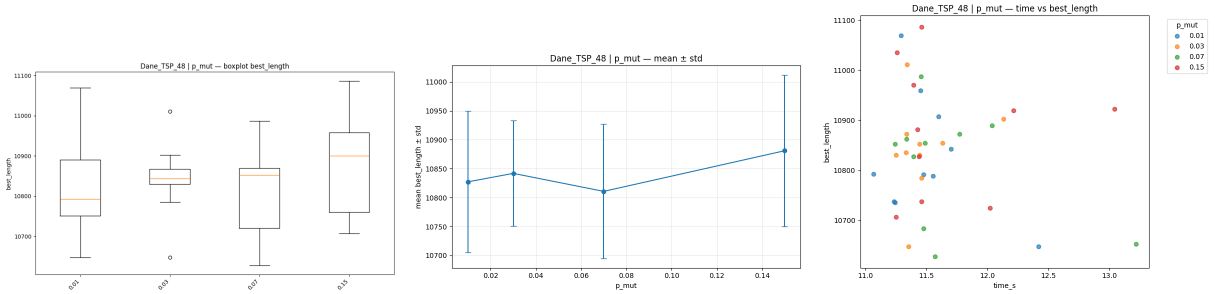
parametr p_cx



Prawdopodobieństwo krzyżowania wpływa na intensywność wymiany materiału genetycznego między osobnikami. Na wykresach widać, że wartości pośrednie prowadzą do uzyskania najkrótszych tras oraz sto-

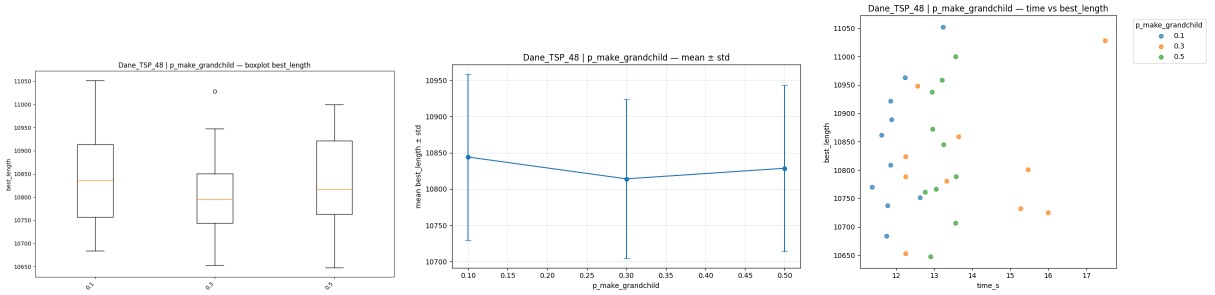
sunkowo niskiego rozrzutu wyników. Przy zbyt niskim p_cx (0.6) wyniki są gorsze i bardziej niestabilne.Na wykresie czasu widać, że różnice w czasie działania są niewielkie, więc ten parametr nie wpływa istotnie na wydajność. Można więc uznać, że optymalny zakres p_cx to około 0,75-0,9, gdzie algorytm osiąga najlepsze wyniki jakościowe.

parametr p_mut



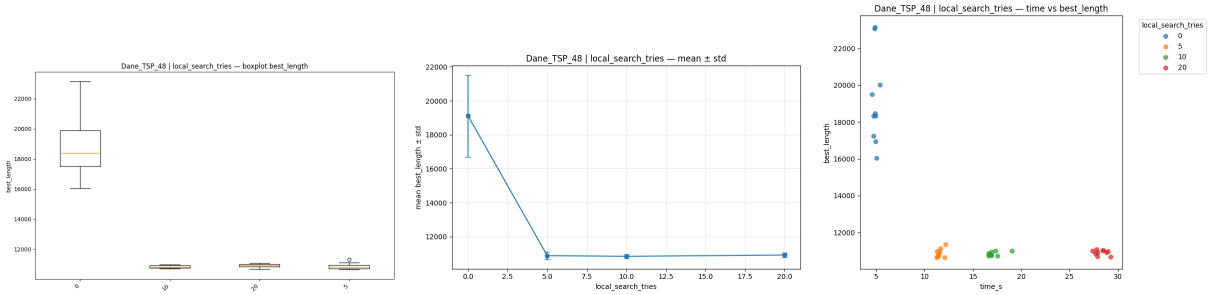
Zbyt małe wartości p_mut (0,01) powodowały przedwczesną zbieżność i gorsze wyniki. Wraz ze wzrostem do 0,03 i , poprawiała się zarówno średnia, jak i stabilność rezultatów. Przy p_mut = 0,15 obserwowano już spadek jakości, zbyt częste mutacje zaburzały proces ewolucji. Najlepszy balans uzyskano przy prawdopodobieństwie wynoszącym około 0,05-0,07.

parametr p_make_grandchild



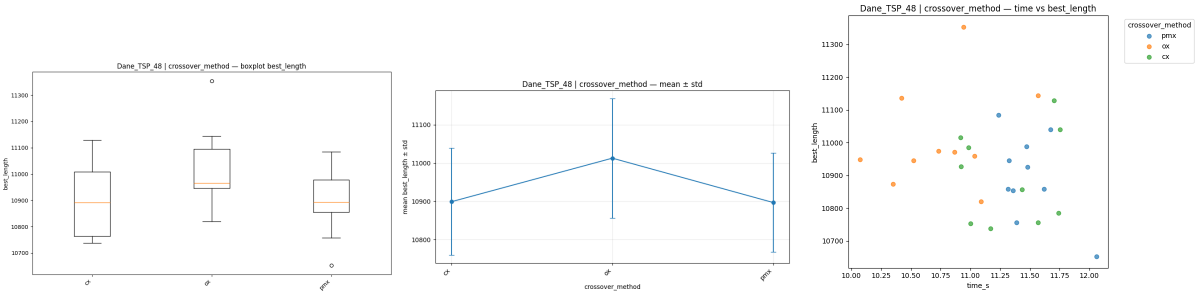
Większy udział „wnuków” w populacji (czyli osobników tworzonych z dodatkowej rekombinacji) prowadził do nieco lepszych wyników, ale też większej zmienności. Dla p_make_grandchild = 0,5 wyniki były najbardziej stabilne, a dalsze zwiększanie tej wartości nie dawało już korzyści. Niskie wartości parametru skutkowały szybkim utknięciem w lokalnych minimach. Optymalny zakres to 0,4-0,5.

parametr local_search_tries



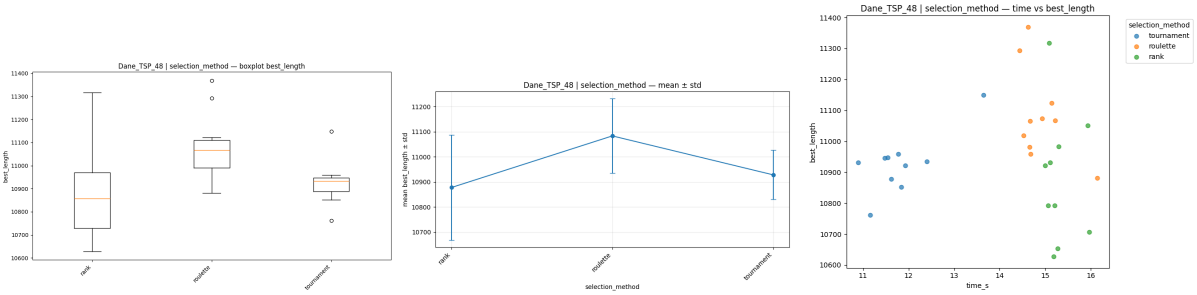
Zwiększenie liczby prób lokalnego przeszukiwania z 0 do 5 powoduje drastyczną poprawę wyników (średnia długość trasy spada z bardzo wysokich wartości) i znacznie zmniejsza rozrzut wyników. Dalsze zwiększenie prób do 10 i 20 daje już niewielkie dalsze korzyści. Scatter time vs best_length pokazuje, że konfiguracje z większą liczbą prób dają lepsze długości ale wymagają więcej czasu.

parametr crossover_method



PMX pozostaje najbardziej stabilny.

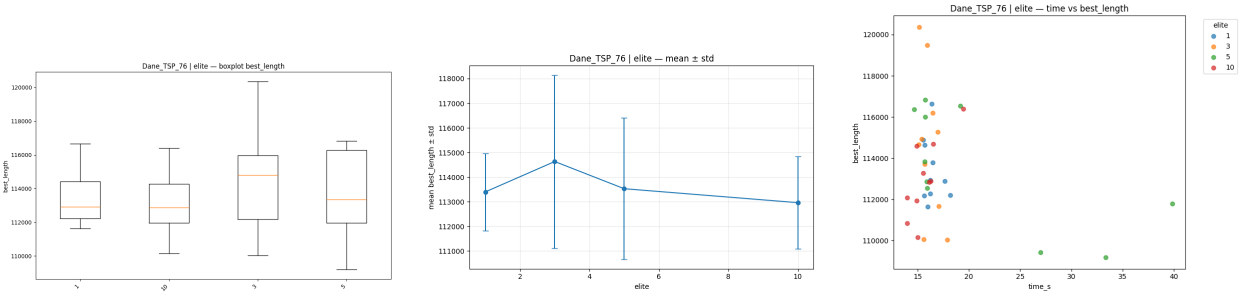
parametr selection_method



Selekcja poprzez turniej zdaje się być najlepszym wyborem.

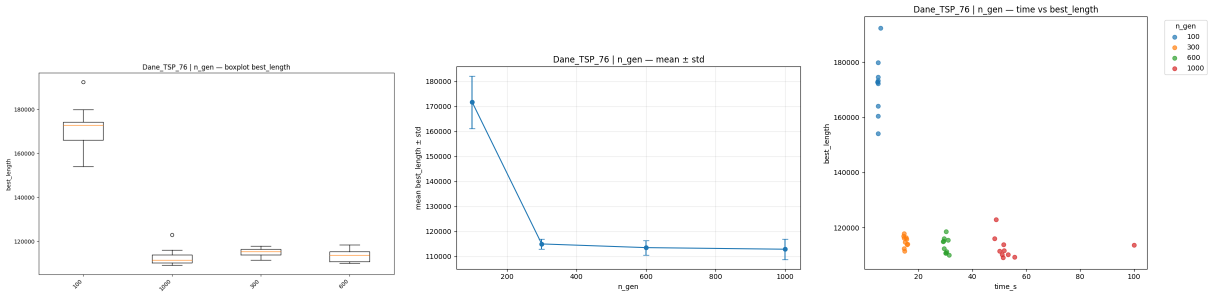
Dane 76 miast

parametr elite



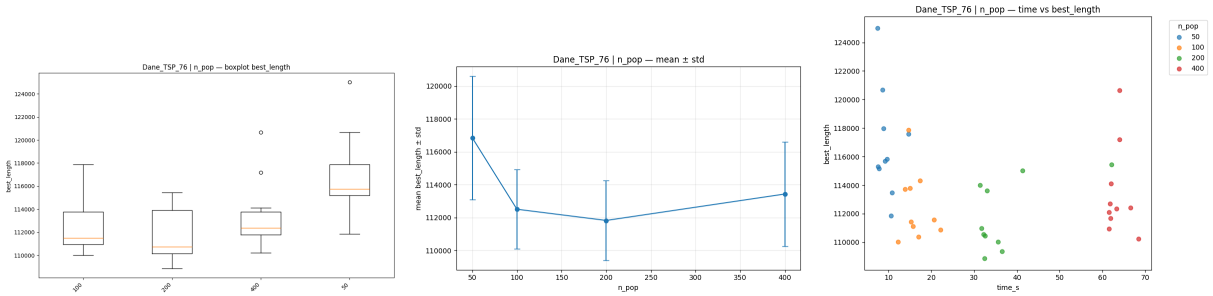
Rozmiar elity znacząco wpływała na stabilność wyników. Dla elite = 1-3 trasy były krótsze i bardziej powtarzalne, natomiast przy elite = 10 pojawiało się więcej outlierów. Zbyt duża elita ograniczała różnorodność populacji, przez co algorytm szybciej tracił zdolność eksploracji. Najbardziej zrównoważone rezultaty uzyskano przy elite = 2.

parametr n_gen



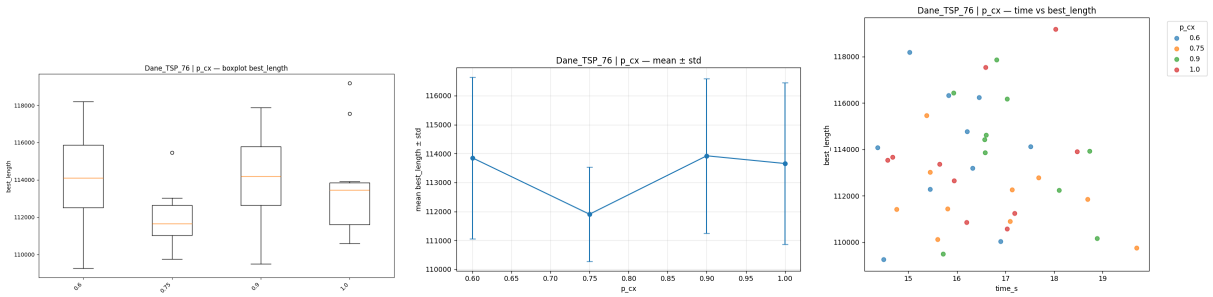
Wraz ze wzrostem liczby generacji poprawiała się jakość rozwiązań, choć przy n_gen $>$ 600 poprawa była już minimalna. Dla n_gen = 300-600 algorytm osiągał stabilne i powtarzalne wyniki. Wykres czasu pokazuje, że wzrost liczby generacji liniowo zwiększał koszt obliczeń, więc rekomendowany kompromis to n_gen = 600.

parametr n_pop



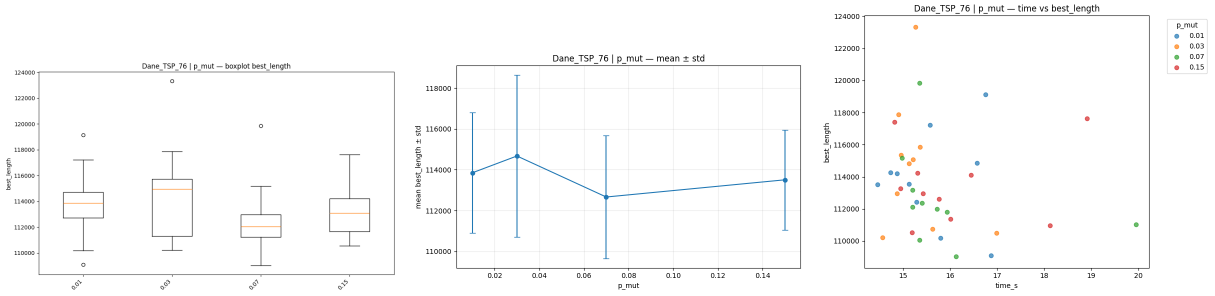
Zwiększenie rozmiaru populacji poprawiało medianę długości tras aż do n_pop = 200. Dla n_pop = 400 efekty się wyrównywały, co sugeruje osiągnięcie wystarczającej różnorodności. Mniejsze populacje charakteryzowały się większą wariancją wyników. Dlatego n_pop = 200 daje najlepszy stosunek jakości do czasu.

parametr p_cx



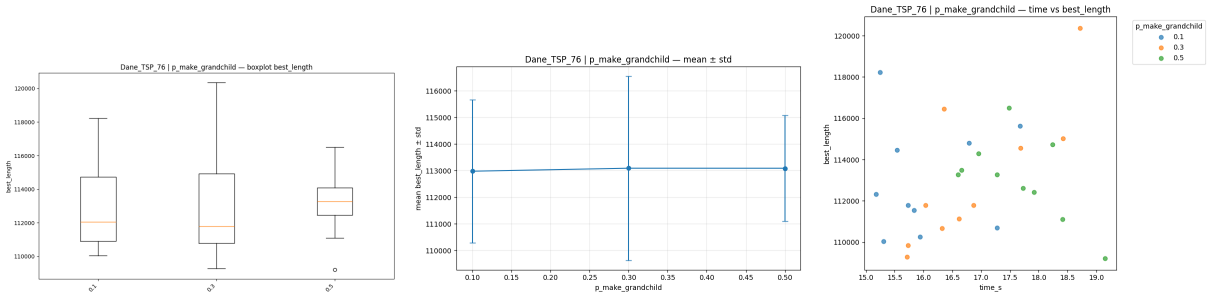
Najlepsze wyniki uzyskano przy p_cx = 0.9, co potwierdza skuteczność częstego krzyżowania dla średniej wielkości zbioru. Dla mniejszych wartości (0.6-0.75) trasy były dłuższe, a rozrzut wyników większy. p_cx = 1.0 nie przyniosło dalszej poprawy, sugerując nasycenie efektu.

parametr p_mut



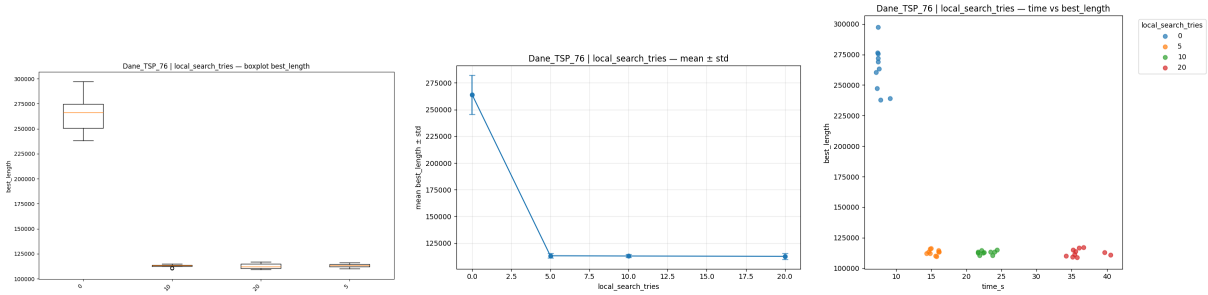
Średni poziom mutacji (0.05-0.07) okazał się najkorzystniejszy. Zbyt niskie wartości prowadziły do stagnacji, a zbyt wysokie (0.15) powodowały losowe skoki i niestabilność wyników. Na wykresie widać, że wariancja była najmniejsza przy p_mut = 0.05, co wskazuje na dobrą równowagę eksploracji i eksploatacji.

parametr p_make_grandchild



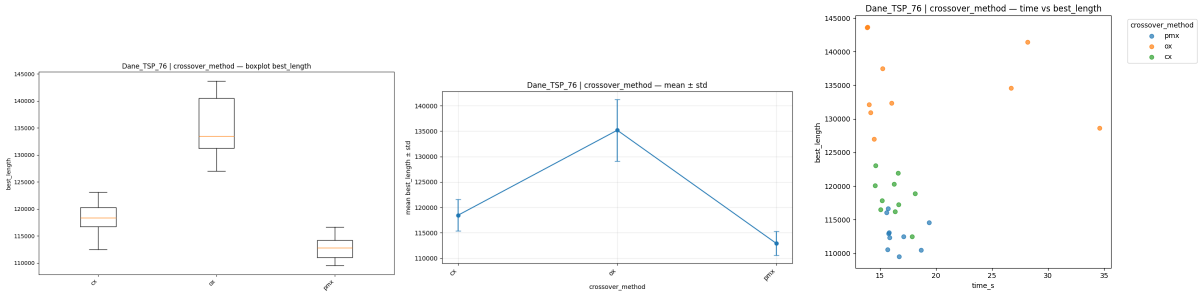
Parametr ten miał umiarkowany wpływ na wyniki, jednak dla średnich wartości (ok. 0,4) uzyskano najlepsze trasy. Niskie wartości powodowały utratę różnorodności, a zbyt wysokie - nadmierną losowość. Na wykresie mean_std widać, że odchylenie standardowe było najmniejsze właśnie w tym zakresie.

parametr local_search_tries



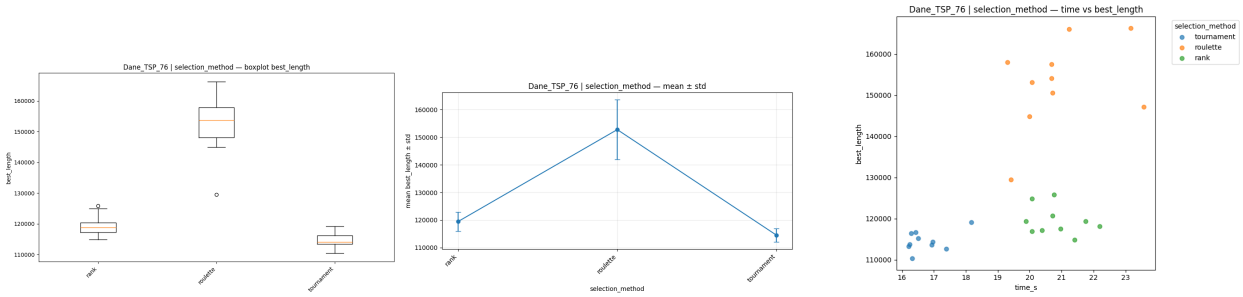
Podobnie jak w mniejszym zbiorze, wyłączenie lokalnego przeszukiwania skutkuje znacznie gorszymi trasami, natomiast przejście do wartości 5-20 daje duży skok jakości (średnie spadają). Wartości 10-20 nie dają już proporcjonalnie większej poprawy jakości. Wnioskiem jest, że dla tej instancji także warto włączyć krótkie lokalne ulepszanie.

parametr crossover_method



PMX daje najlepsze wyniki.

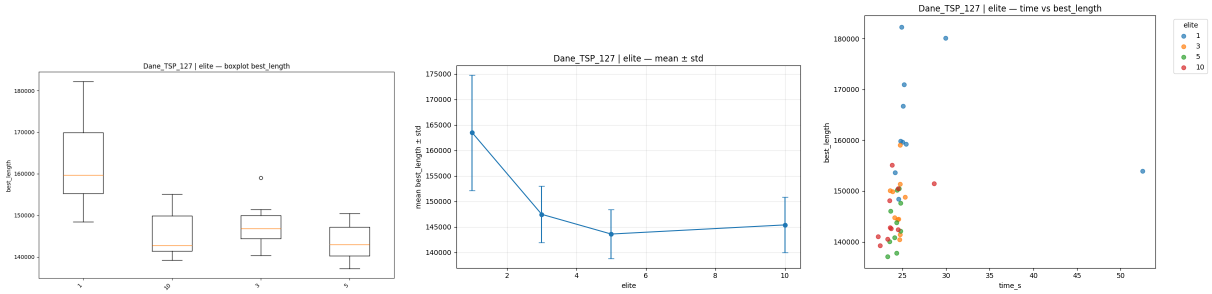
parametr selection_method



Tutaj również optymalnym wyborem jest selekcja poprzez turniej.

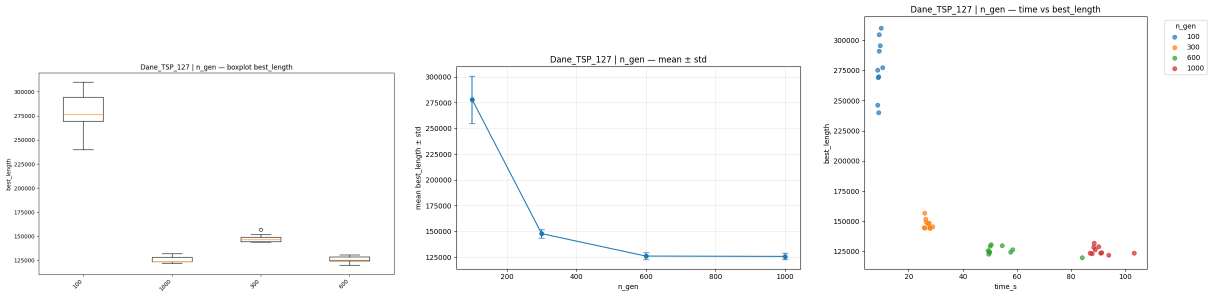
Dane 127 miast

parametr elite



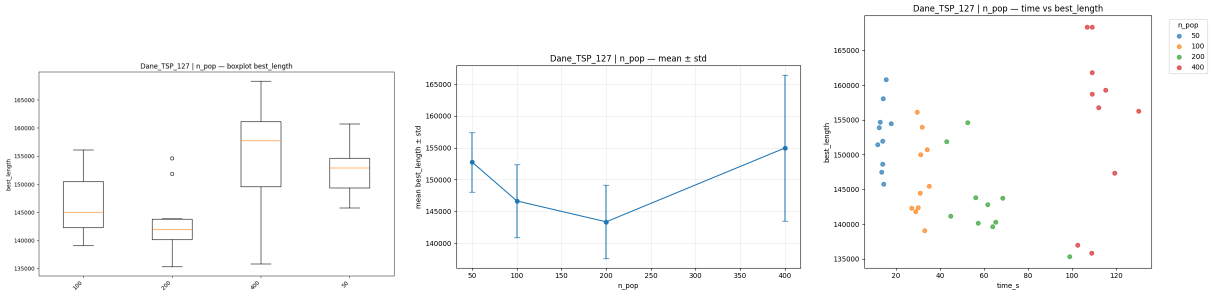
Dla największego zestawu danych umiarkowana elita (2-3 osobniki) była kluczowa dla stabilności. Przy braku elity wyniki były niestabilne, a najlepsze trasy często ginęły między pokoleniami. Natomiast zbyt duża elita (np. 10) powodowała stagnację i brak poprawy po kilku generacjach. Optymalna wartość to 3.

parametr n_gen



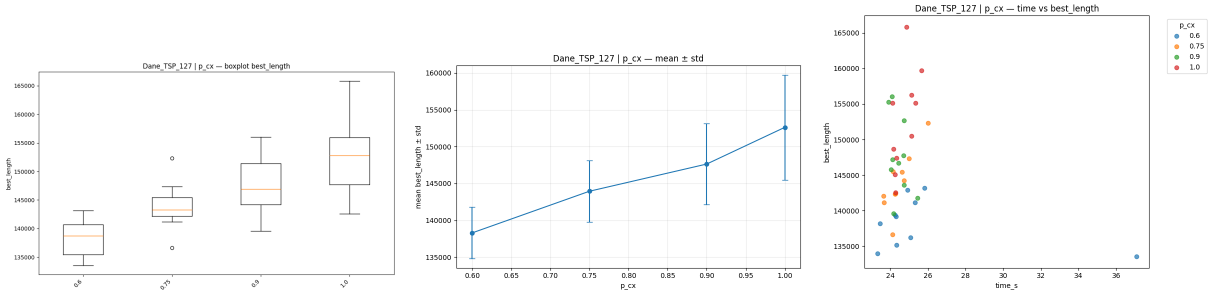
Zwiększanie liczby generacji przynosiło wyraźną poprawę jakości rozwiązań aż do poziomu 1200. Wykresy pokazują, że dla n_gen $>$ 600 algorytm wciąż znajduje lepsze rozwiązania, choć z malejącym tempem przyrostu jakości. Czas obliczeń rośnie jednak proporcjonalnie, więc kompromisowy wybór to n_gen = 600-900.

parametr n_pop



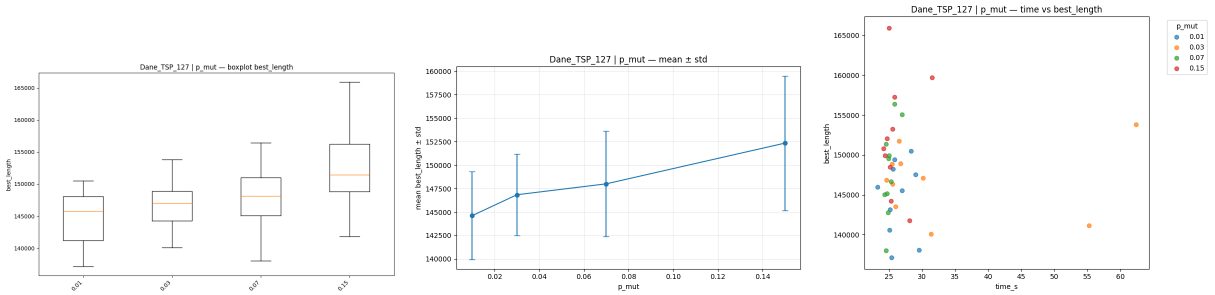
Dla dużej instancji największy wpływ miał rozmiar populacji. n_pop = 200 zapewniał wyraźnie lepsze wyniki niż 50 czy 100, a dalsze zwiększanie do 400 poprawiało medianę już tylko nieznacznie. Jednocześnie czas wzrastał ponad dwukrotnie, dlatego n_pop = 200 uznano za najbardziej efektywny kompromis.

parametr p_cx



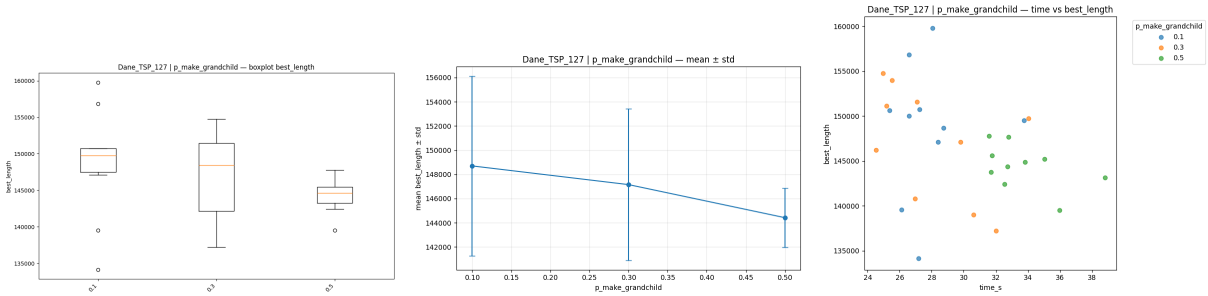
Najniższe długości tras uzyskano przy p_cx = 0.9. Dla mniejszych wartości poprawa była wolniejsza, a dla p_cx = 1.0 - zauważalnie większa wariancja. Wynika to z tego, że całkowite krzyżowanie wszystkich par rodziców zwiększa losowość populacji. Optymalny zakres to p_cx = 0.85-0.9.

parametr p_mut



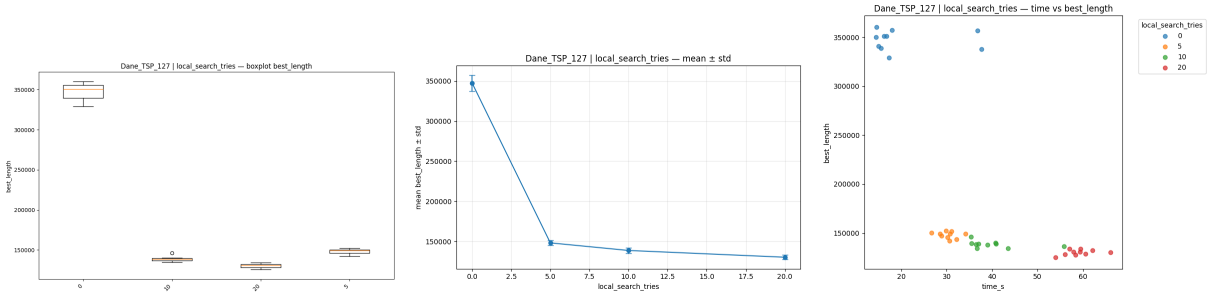
Dla największej liczby miast mutacja miała silniejszy wpływ. Przy p_mut = 0.01 wyniki były bardzo słabe, a przy p_mut = 0.15 - niestabilne. Najlepsze rezultaty pojawiły się przy p_mut = 0.07, gdzie algorytm utrzymywał równowagę między eksploracją a zbieżnością.

parametr p_make_grandchild



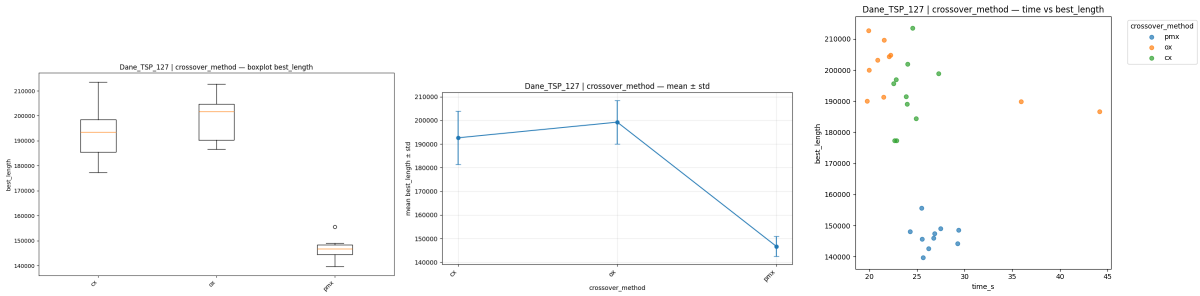
Wraz ze wzrostem p_make_grandchild poprawiała się jakość rozwiązań do wartości około 0.5. Powyżej tej granicy efekty się stabilizowały, ale czas obliczeń wzrósł. Dla p_make_grandchild = 0.4-0.5 wyniki były najrówniejsze i najczęściej dawały najkrótsze trasy.

parametr local_search_tries



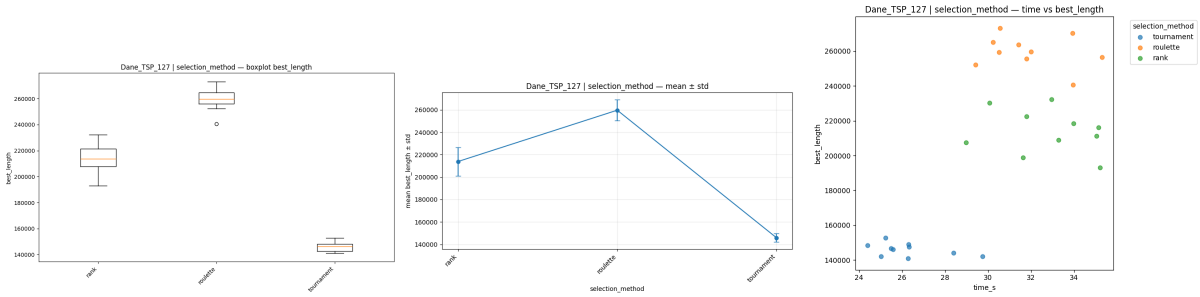
Tutaj efekt jest jeszcze bardziej widoczny, brak lokalnego ulepszania daje najgorsze wyniki (średnio ~340-360k), a włączenie 5 prób obniża średnią do ok. 150k, natomiast dalsze zwiększanie do 10 i 20 prób przynosi stopniowe, ale widoczne dalsze usprawnienia. Najlepsze t rasy uzyskuje się kosztem najdłuższego czasu.

parametr crossover_method



Krzyżowanie PMX daje najlepsze wyniki.

parametr selection_method



I jak w każdym z wcześniejszych przypadków, to selekcja poprzez turniej wypada najlepiej.

Podsumowanie i najlepszy zestaw parametrów

48 miast

Na podstawie przeprowadzonej analizy dla problemu TSP z 48 miastami, dostajemy najlepszy wynik (best_length): 10628. To dzięki konfiguracji:

- n_pop = 200
- n_gen = 300
- p_mut = 0.03
- p_cx = 0.9
- selection_method = tournament
- crossover_method = pmx
- local_search_tries = 5
- elite = 3
- p_make_grandchild = 0

Najlepszy wynik uzyskaliśmy przy umiarkowanie dużej populacji i standardowej liczbie generacji. Konfiguracja ta potwierdza skuteczność parametrów bazowych, zwłaszcza włączenia algorytmu local_search_tries i elitaryzmu. Wyłączenie mechanizmu „wnuka” (p_make_grandchild = 0) okazało się korzystne.

76 miast

Na podstawie przeprowadzonej analizy dla problemu TSP z 76 miastami, dostajemy najlepszy wynik (best_length): 108880. To dzięki konfiguracji:

- n_pop = 200
- n_gen = 300
- p_mut = 0.03
- p_cx = 0.9
- selection_method = tournament
- crossover_method = pmx
- local_search_tries = 5
- elite = 3
- p_make_grandchild = 0

Dla instancji średniej wielkości, najlepsza konfiguracja okazała się identyczna jak dla problemu 48 miast. Potwierdza to wysoką stabilność i skuteczność tej kombinacji parametrów dla problemów tej skali.

127 miast

Na podstawie przeprowadzonej analizy dla problemu TSP ze 127 miastami, dostajemy najlepszy wynik (best_length): 120101. To dzięki konfiguracji:

- n_pop = 100
- n_gen = 600
- p_mut = 0.03
- p_cx = 0.9
- selection_method = tournament
- crossover_method = pmx
- local_search_tries = 5
- elite = 3
- p_make_grandchild = 0

Dla największego problemu optymalna strategia uległa zmianie, algorytm potrzebował mniejszej populacji, ale dwa razy dłuższej ewolucji. Wskazuje to, że dla bardziej złożonych problemów kluczowe znaczenie ma większa liczba iteracji i dłuższy czas na zbieganie. Warto jednak zauważyć, że wszystkie pozostałe kluczowe parametry (mutacja, krzyżowanie, selekcja, local search, elita i "wnuk") pozostały identyczne jak w mniejszych instancjach.

Wnioski

Analizując konfiguracje, które dały absolutnie najlepsze wyniki dla każdej z trzech instancji problemu, można zauważyć zaskakującą spójność. Aż 7 z 9 testowanych grup parametrów było identycznych dla najlepszych przebiegów:

- p_mut = 0.03
- p_cx = 0.9
- selection_method = tournament
- crossover_method = pmx
- local_search_tries = 5
- elite = 3
- p_make_grandchild = 0

Taka konfiguracja okazała się najbardziej uniwersalna. Nasze usprawnienie - mechanizm „wnuka” w żadnym z najlepszych przypadków nie przyczynił się do poprawy wyniku.