Algorytm genetyczny dla permutacyjnego problemu przepływowego (PFSP)

2025-10-24

Problem Przepływowy (PFSP)

Rozważany problem to **Permutacyjny Problem Przepływowy** (Permutation Flow Shop Scheduling Problem, PFSP). Celem jest znalezienie optymalnej sekwencji (permutacji) `N` zadań, które muszą być przetworzone na `M` maszynach w tej samej kolejności. Kryterium optymalizacji jest minimalizacja czasu zakończenia ostatniego zadania na ostatniej maszynie, znanego jako **makespan**.

Założenia dotyczące problemu PFSP

- Wszystkie zadania (`N`) są dostępne do przetwarzania od samego początku.
- Każde zadanie musi przejść przez wszystkie maszyny (`M`) w tej samej, z góry ustalonej kolejności.
- Kolejność przetwarzania zadań jest identyczna na każdej maszynie (stąd nazwa "permutacyjny").
- Czasy przetwarzania każdego zadania na każdej maszynie są znane i nieujemne.
- Operacje nie mogą być przerywane.
- Każda maszyna może w danym momencie przetwarzać tylko jedno zadanie.

Funkcja celu

Główną metryką oceny jakości rozwiązania jest **makespan**. Jest to czas, w którym zakończy się przetwarzanie ostatniego zadania w sekwencji na ostatniej maszynie. Im mniejszy makespan, tym lepsze rozwiązanie.

Dane

Plik wejściowy zawiera macierz czasów przetwarzania zadań na maszynach:

- `Dane_PFSP_50_20.xlsx` - instancja z 50 zadaniami i 20 maszynami.
- `Dane_PFSP_200_10.xlsx` - instancja z 200 zadaniami i 10 maszynami.
- `Dane_PFSP_100_10.xlsx` - instancja z 100 zadaniami i 10 maszynami.

Algorytm wykorzystuje tę macierz do oceny (obliczenia makespanu) każdej permutacji.

Wstęp

Badamy algorytm genetyczny (GA rozwiązujący problem PFSP. 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 powtarzalność eksperymentów

- **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 zadań - kolejność wykonania zadań określa sekwencję. Populacja osobników ewoluuje w kierunku coraz niższych wartości makespan.

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 permutacja zostanie losowo zmieniona; w tym przypadku polega na zamianie (swap) dwóch zadań w kolejności; nie gwarantuje poprawy rozwiązania; (!nie należy mylić mutacji z ulepszaniem local search (ruchami sąsiedzkimi), które działają już po mutacji i próbują celowo poprawić permutację (obniżyć jej makespan), 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 (osobnicy o niższym makespan) 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 najniższym makespanie (najlepszy), pozwala kontrolować siłę selekcji przez tournament_k; im większe k, tym silniejsza selekcja.
2) roulette (selekcja ruletkowa) - każdy osobnik dostaje „wagę” na ruletce proporcjonalną do swojej jakości (u nas odwrotnie do wartości makespan); daje większe szanse dobrym permutacjom, 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 wartości makespan. Działa stabilniej niż ruletka, gdy np. jeden osobnik jest drastycznie lepszy od reszty.

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

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

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

3) CX (Cycle Crossover) - tworzy cykle powiązań między genami rodziców (np. zadanie A u pierwszego jest w tym samym miejscu co zadanie 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 makespan permutacji:

1) swap - zamiana dwóch zadań miejscami,

2) 2-opt - odwrócenie fragmentu permutacji zadań,

3) insertion - wyjęcie zadania 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. Taki model pozwala zbadać kluczową różnicę wpływu włączenia lokalnej poprawy (nawiązanie do Algorytmu Memetycznego) w porównaniu do jej braku (co stanowi czysty Algorytm Genetyczny).

- elite (elitaryzm) oznacza liczbę najlepszych osobników (o najniższym makespan) kopiowanych bez zmian do następnej generacji; chroni najlepsze znalezione rozwiązania, ale zbyt duża wartość elite mogłaby zmniejszyć różnorodność populacji.

Usprawnienie czyli mechanizmu „Wnuka” (grandchild)

Mechanizm „wnuka” (grandchild) to proste rozszerzenie standardowego schematu rozmnażania. Po wygenerowaniu potomka (child) 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), czyli ten o niższym makespanie. Intuicyjnie, zamiast przyjmować bezwarunkowo pojedynczego potomka, sprawdzamy jedną dodatkową opcję „bliźniaczą” i zachowujemy tę, która daje lepszy wynik.

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 (wykorzystujący tę samą funkcję local_improve, co główna pętla).
- grandchild_local_tries - liczba prób dla lokalnego ulepszania wnuka (jeśli None, używany jest ogólny 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.

Efekty obserwowane w eksperymentach:

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

Konfiguracja eksperymentu (OFAT)

Parametry domyślne (Baseline)

Konfiguracja bazowa, od której wychodzimy przy testowaniu każdego parametru:

- `n_pop`: 100,
- `n_gen`: 100,
- `p_mut`: 0.03,
- `p_cx`: 0.9,
- `selection_method`: 'tournament',
- `crossover_method`: 'pmx',
- `local_search_tries`: 3,
- `local_moves`: ['swap','two_opt','insertion'],
- `elite`: 3,
- `tournament_k`: 3,
- `p_make_grandchild`: 0.0,
- `grandchild_method`: 'local_improve',
- `grandchild_local_tries`: 3

Parametry testowane

Dla każdego z poniższych parametrów testowano podane wartości, podczas gdy reszta pozostawała zgodna z konfiguracją `baseline`. Każda konfiguracja została uruchomiona **10 razy**.

- `n_pop`: [50, 100, 200, 400]
- `n_gen`: [100, 300, 600, 1000]
- `p_mut`: [0.01, 0.03, 0.07, 0.15]
- `p_cx`: [0.6, 0.75, 0.9, 1.0]
- `elite`: [1, 3, 5, 10]
- `local_search_tries`: [0, 3, 7, 15]
- `p_make_grandchild`: [0.0, 0.1, 0.3, 0.5]
- `crossover_method`: ['pmx', 'ox', 'cx'],
- `selection_method`: ['tournament', 'roulette', 'rank']

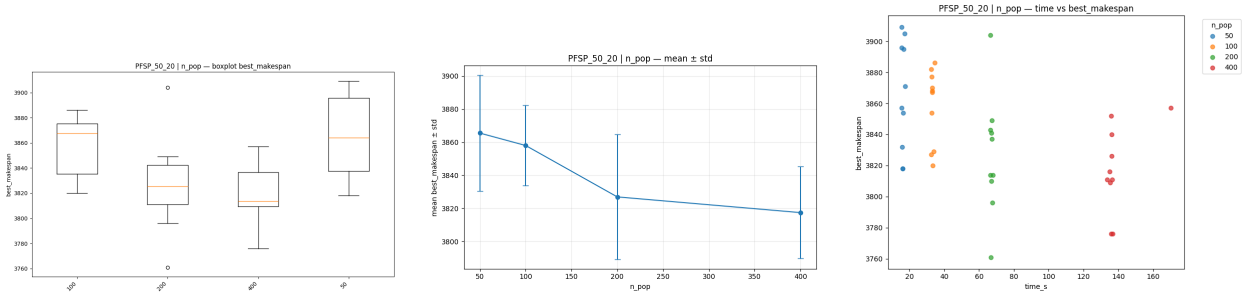
Analiza wyników

Dla każdego parametru przedstawiono:

1. **Wykres pudełkowy (boxplot)** - pokazuje rozkład, medianę i wartości odstające (outliery) końcowego makespanu.
2. **Wykres średniej i odchylenia standardowego** - obrazuje średnią jakość i stabilność wyników.
3. **Wykres rozrzutu (scatter plot)** - przedstawia zależność między czasem obliczeń a osiągniętym makespanem.

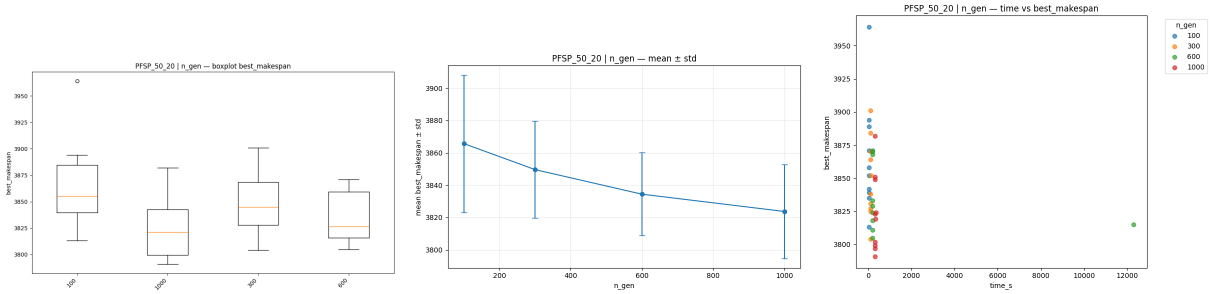
Dane PFSP_50_20

parametr `n_pop` (wielkość populacji)



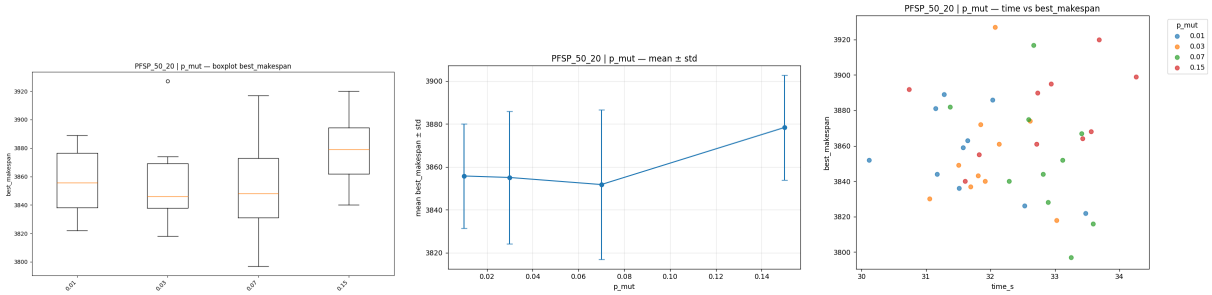
Zwiększanie wielkości populacji z 50 do 200 przynosi wyraźną poprawę zarówno średniego makespanu, jak i stabilności wyników (mniejszy rozrzut). Dalsze zwiększanie populacji do 400 nie daje już znaczącej poprawy jakości, natomiast znacząco wydłuża czas obliczeń, co widać na wykresie rozrzutu. Optymalny kompromis między jakością rozwiązania a czasem działania dla tej instancji problemu stanowi populacja o wielkości **200 osobników**.

Parametr `n_gen` (liczba generacji)



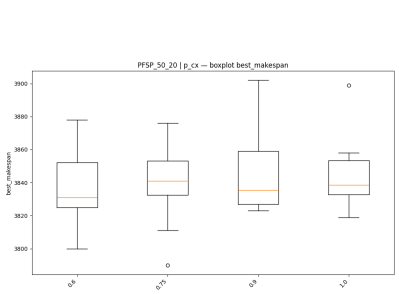
Wraz ze wzrostem liczby generacji, średni makespan maleje, co jest oczekiwanym zachowaniem. Wartość **300-600 generacji** wydaje się być najlepszym wyborem, zapewniając dobre wyniki w akceptowalnym czasie.

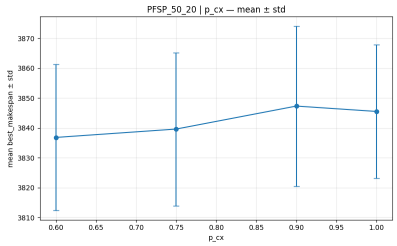
parametr `p_mut` (prawdopodobieństwo mutacji)

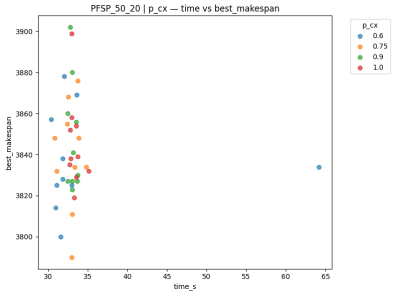


Zbyt duża mutacja pogarsza stabilność. Optymalne wartość to 0,03.

parametr `p_cx` (prawdopodobieństwo krzyżowania)

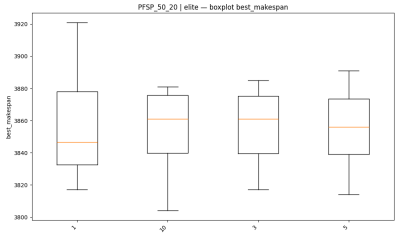


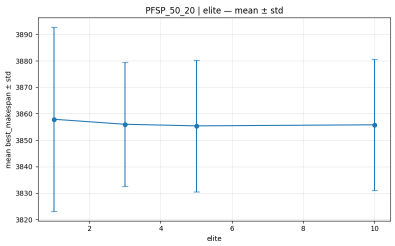


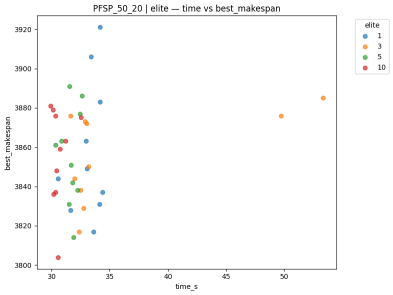


Najlepsze wyniki pomiędzy 0,6-0,75.

parametr `elite` (liczba osobników elitarnych)

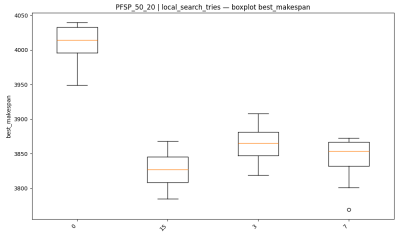


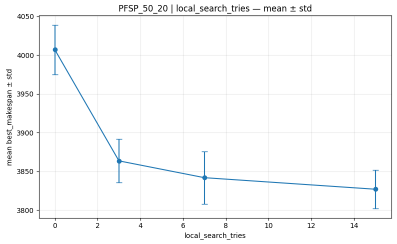


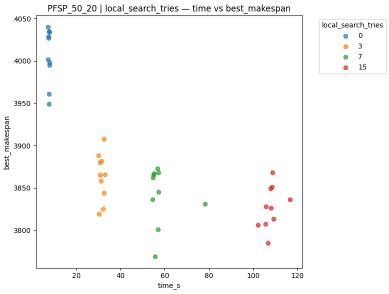


Zwiększenie elity do 3-5 osobników stabilizuje wyniki i chroni najlepsze rozwiązania, bez utraty różnorodności. Powyżej 5 efekt jest marginalny.

parametr `local_search_tries` (liczba prób ulepszenia lokalnego)

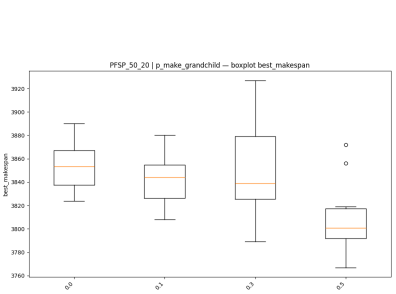


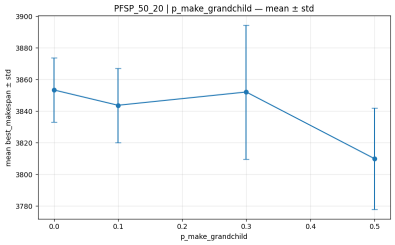


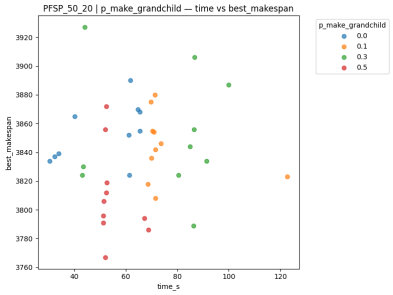


Wyłączenie mechanizmu lokalnego poszukiwania (`tries = 0`) daje zdecydowanie najgorsze wyniki. Już włączenie 3 prób powoduje drastyczną poprawę makespanu. Dalsze zwiększanie liczby prób (do 7 i 15) przynosi dodatkową, choć już mniejszą, poprawę. Wykres czasu do jakości pokazuje, że lepsze wyniki osiągane są kosztem dłuższego czasu działania. Mechanizm lokalnego ulepszenia jest **kluczowy dla jakości**. Wartość w zakresie **3-7 prób** stanowi dobry kompromis.

parametr `p_make_grandchild` (prawdopodobieństwo “wnuka”)

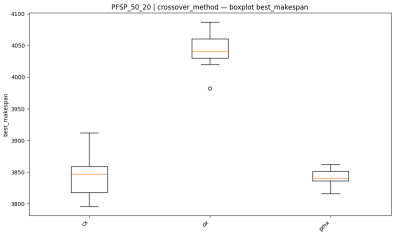


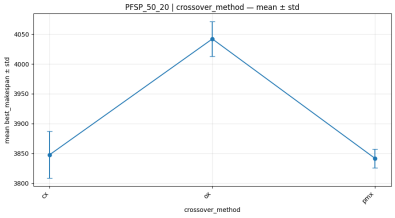


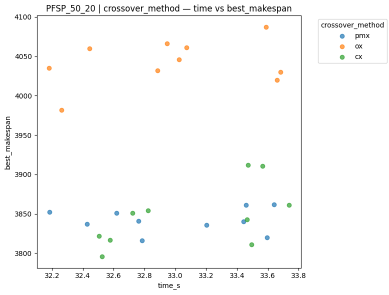


Wraz ze wzrostem prawdopodobieństwa p_make_grandchild obserwuje się umiarkowaną poprawę jakości rozwiązań. Przy 0.5 jakość przestaje się poprawiać. Optymalna wartość znajduje się pomiędzy 0,1-0,2.

parametr crossover_method

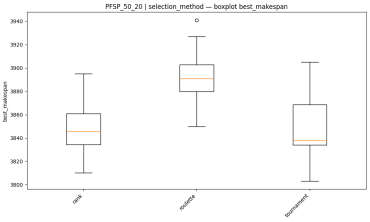


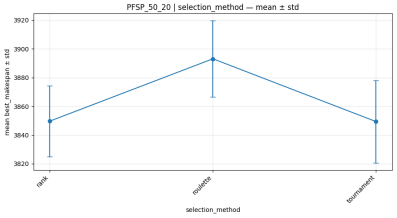


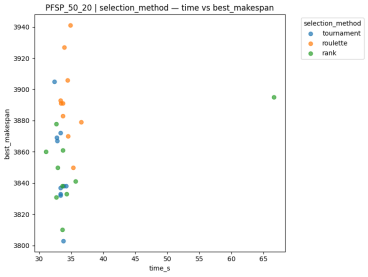


Metody PMX i OX zapewniają podobną jakość, ale PMX daje stabilniejsze wyniki. To PMX wykorzystujemy jako podstawową metodę krzyżowania.

parametr selection_method



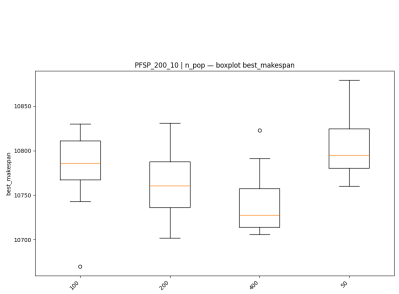


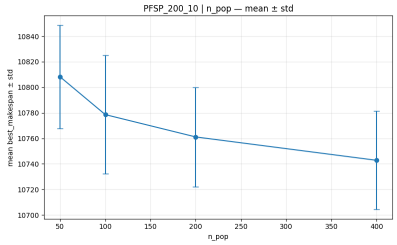


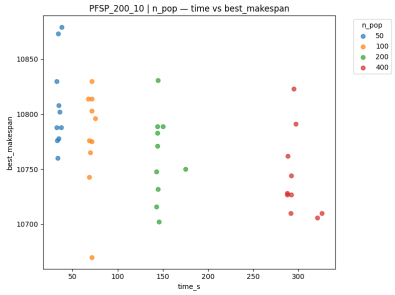
Selekcja turniejowa jest najbardziej niezawodna, zapewnia szybkie zbieganie i stabilność wyników.

Dane PFSP_200_10

parametr `n_pop` (wielkość populacji)

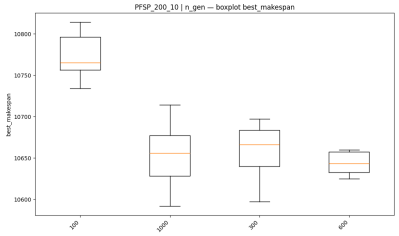


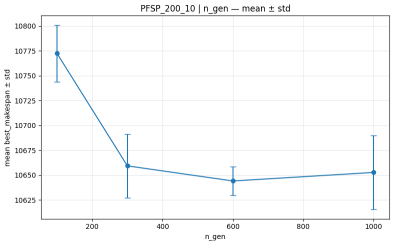


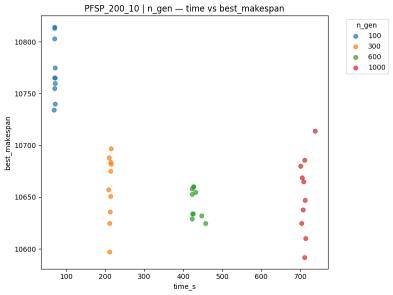


Wraz ze wzrostem populacji poprawia się jakość rozwiązań, ale przy znacznym wzroście czasu. Populacja 200-400 osobników to najlepszy kompromis.

parametr `n_gen` (liczba generacji)

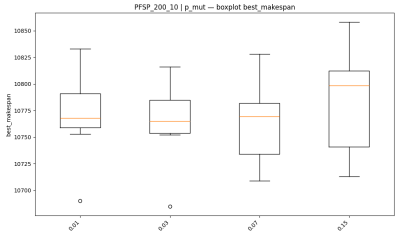


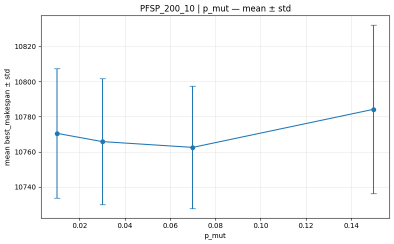


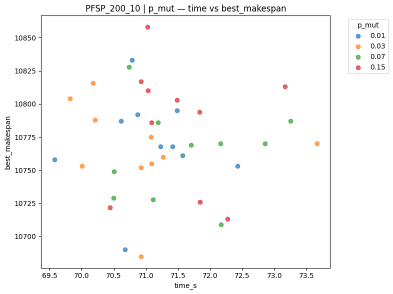


Większa liczba generacji przynosi korzyści aż do około 600, po czym zysk jakościowy maleje.

parametr `p_mut` (prawdopodobieństwo mutacji)

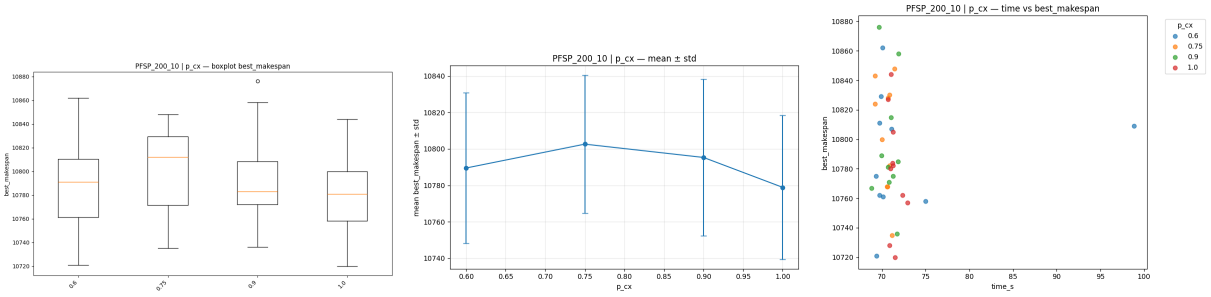






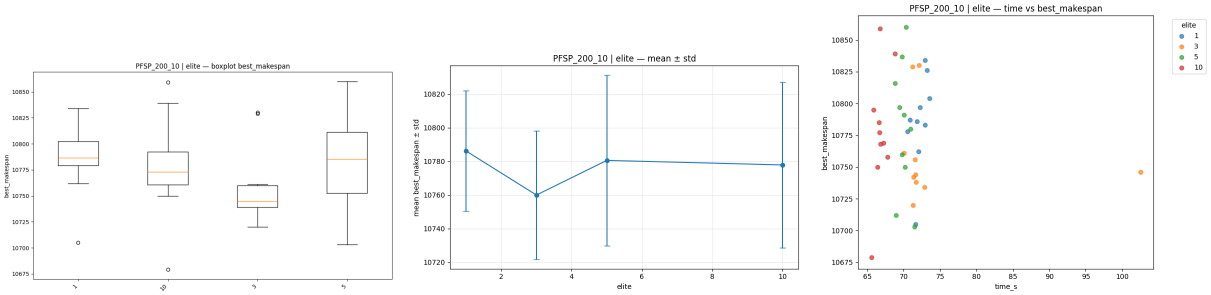
Wartość 0,07 daje najlepszy efekt.

parametr `p_cx` (prawdopodobieństwo krzyżowania)



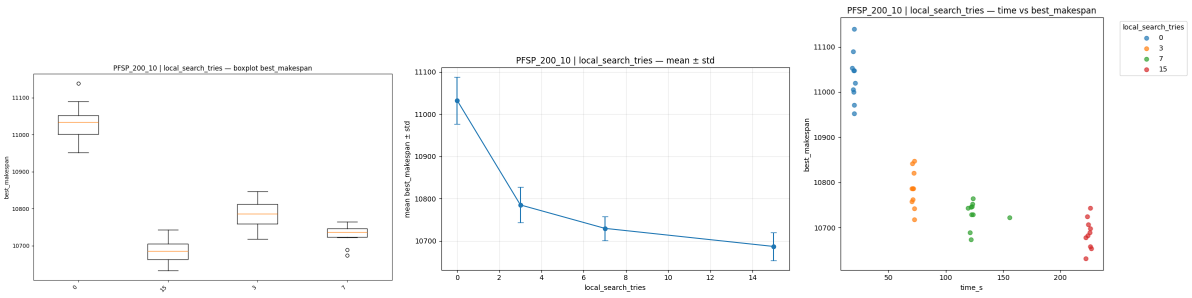
Wysokie prawdopodobieństwo krzyżowania zwiększa szansę na dobre kombinacje genów, 0,9 jest najbardziej efektywne.

parametr `elite` (liczba osobników elitarnych)



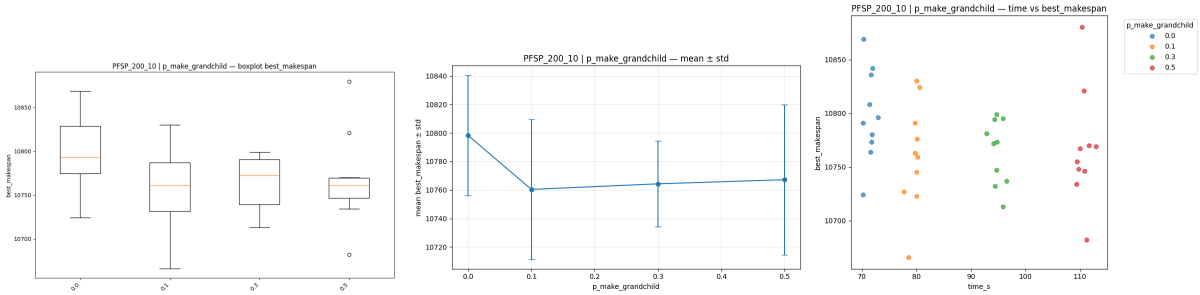
Umiarkowany elitaryzm pozwala zachować balans między ochroną najlepszych a utrzymaniem różnorodności. Optymalnie 3 elity.

parametr `local_search_tries` (liczba prób ulepszenia lokalnego)



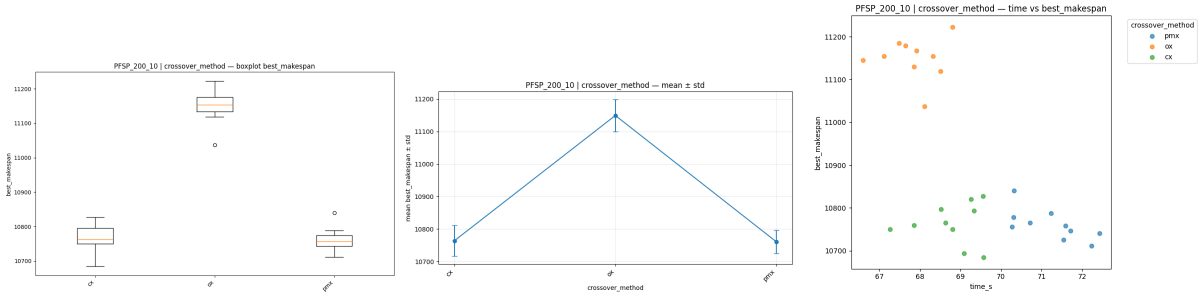
Większa liczba prób ulepszenia lokalnego poprawia wyniki, ale rośnie też czas obliczeń. 7 prób to dobry kompromis.

parametr `p_make_grandchild` (prawdopodobieństwo “wnuka”)



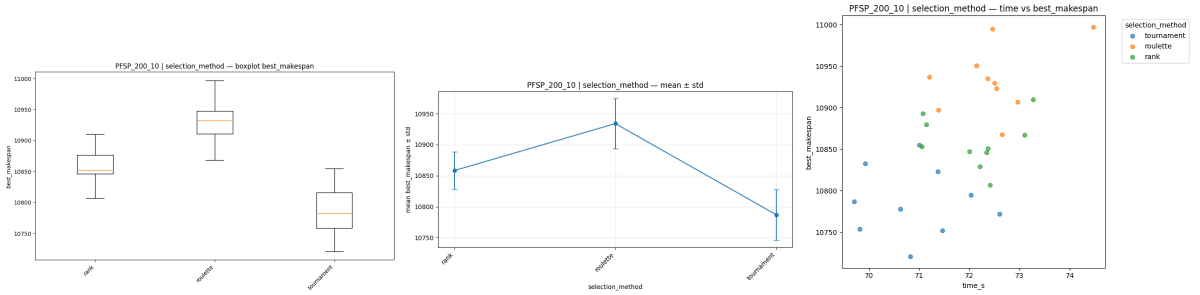
Wartości około 0,3 przynoszą wyraźną poprawę jakości bez dużego wzrostu czasu działania.

parametr `crossover_method` (metody krzyżowania)



PMX pozostaje najbardziej stabilny.

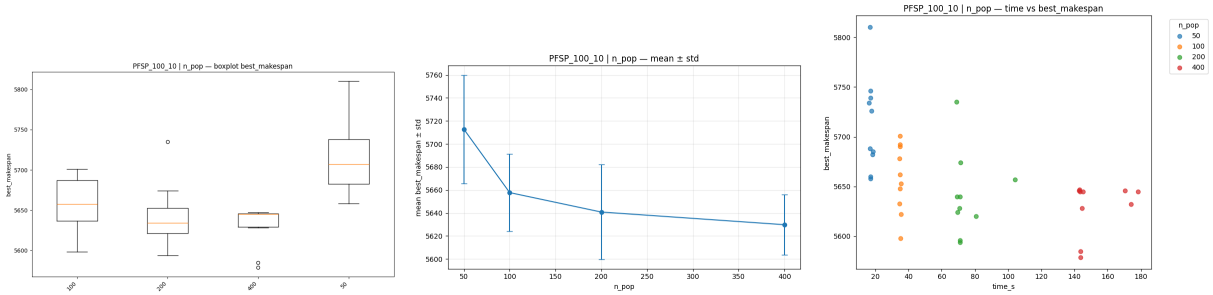
parametr `selection_method` (metody selekcji)



Selekcja turniejowa zapewnia najlepszy stosunek jakości do czasu.

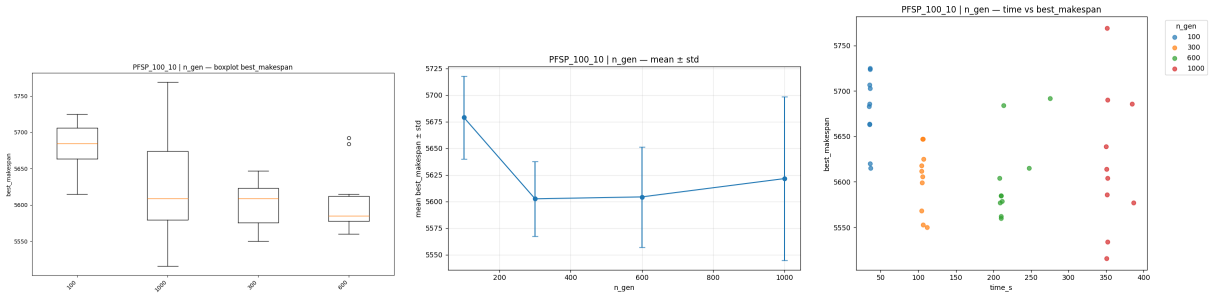
Dane PFSP_100_10

parametr `n_pop` (wielkość populacji)



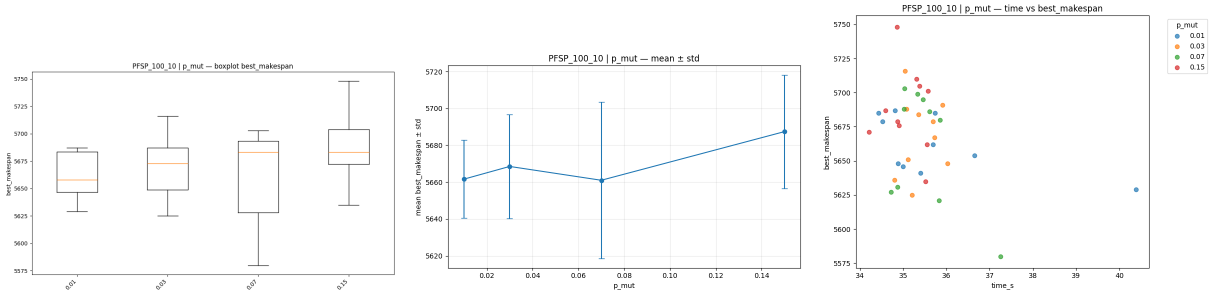
Wielkość populacji 200 zapewnia najwyższą jakość przy rozsądnym czasie działania.

parametr `n_gen` (liczba generacji)



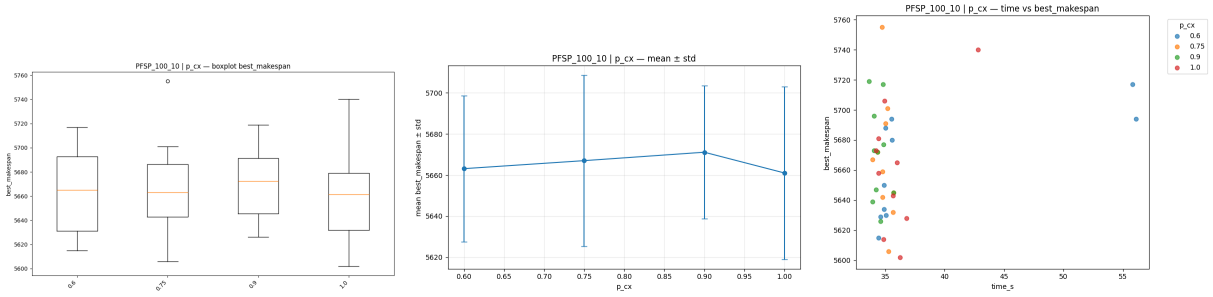
Wyniki poprawiają się wraz z generacjami do około 600, potem korzyści są minimalne.

parametr `p_mut` (prawdopodobieństwo mutacji)



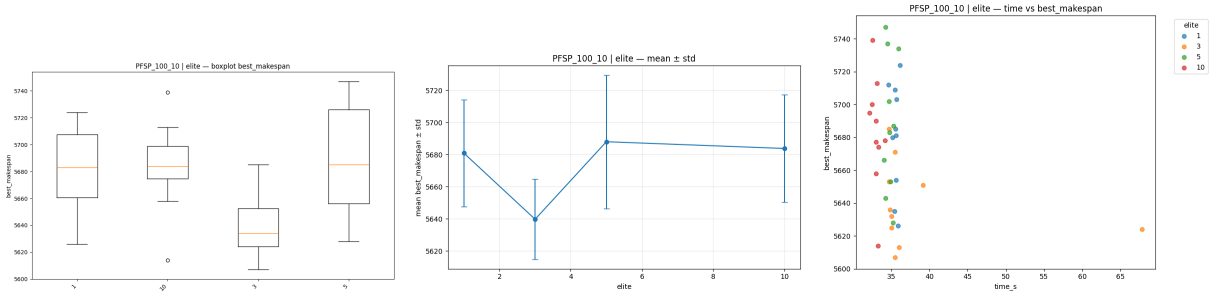
Najlepszą równowagę między różnorodnością a stabilnością zapewnia wartość 0,07.

parametr `p_cx` (prawdopodobieństwo krzyżowania)



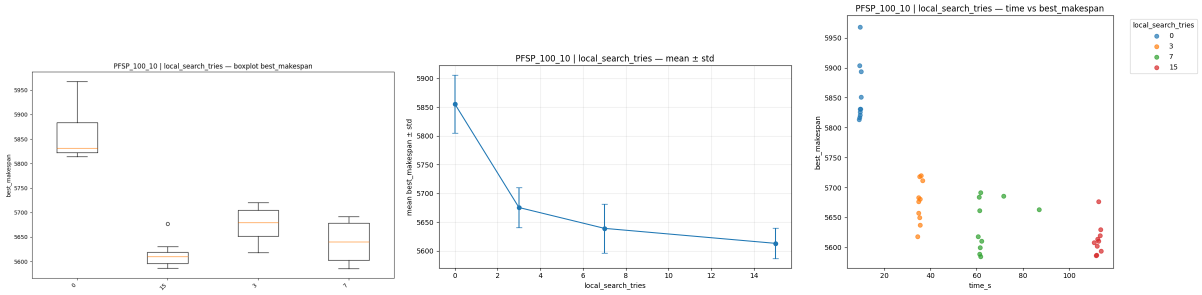
Wartość około 0,75 umożliwia najlepsze łączenie cech rodziców i stabilne wyniki.

parametr `elite` (liczba osobników elitarnych)



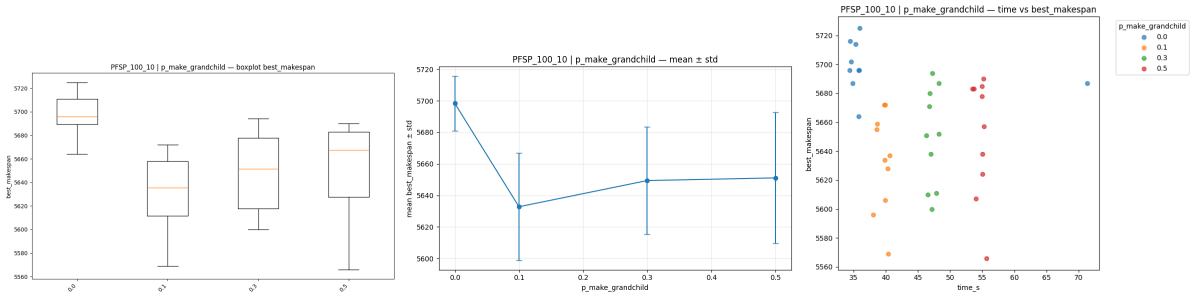
Najlepszy efekt daje 3 elity, zapewniając stabilność i brak utraty różnorodności.

parametr `local_search_tries` (liczba prób ulepszenia lokalnego)



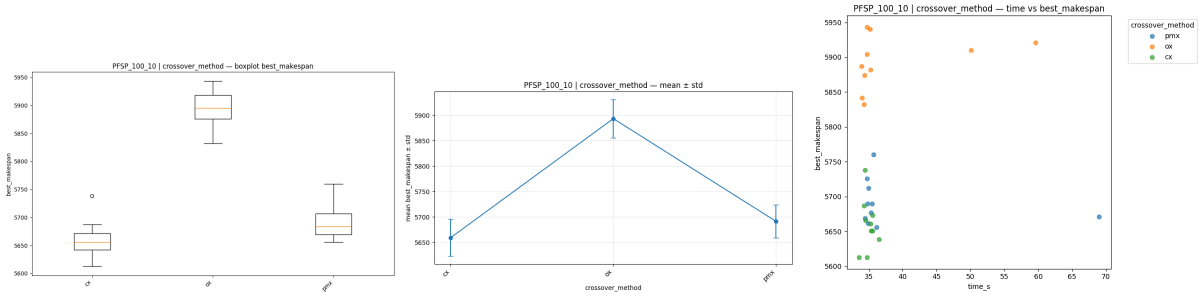
Brak lokalnego ulepszania znacząco pogarsza wyniki; 3-7 prób daje wyraźną poprawę jakości.

parametr `p_make_grandchild` (prawdopodobieństwo “wnuka”)



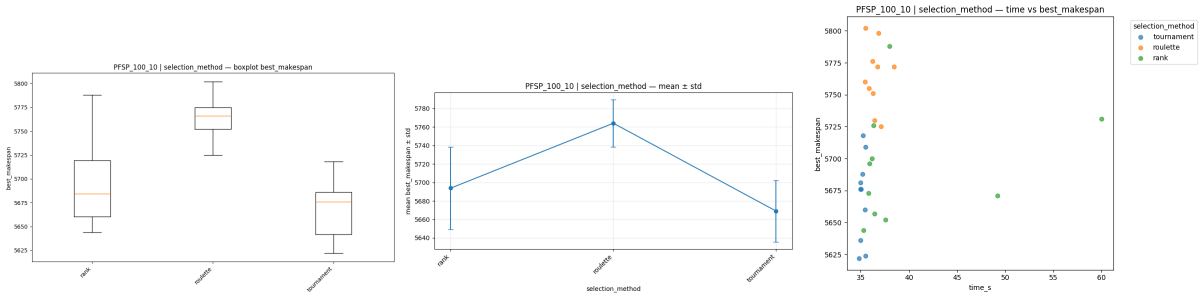
Najlepsze rezultaty uzyskuje się dla p_make_grandchild = 0,1, co umiarkowanie poprawia jakość rozwiązań.

parametr `crossover_method` (metody krzyżowania)



PMX i CX wypadają podobnie, jednak PMX daje bardziej spójne wyniki.

parametr `selection_method` (metody selekcji)



Turniejowa selekcja utrzymuje stabilność i szybkość.

Podsumowanie i najlepszy zestaw parametrów

50 zadań i 20 maszyn

Na podstawie przeprowadzonej analizy dla problemu PFSP z 50 zadaniami i 20 maszynami, dostajemy najlepszy wynik (best_makespan): 3761. To dzięki konfiguracji:

- n_pop = 200

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

Najlepszy wynik uzyskaliśmy przy umiarkowanej populacji i niewielkiej liczbie generacji, co sugeruje, że już w początkowych etapach algorytm zbiega do wysokiej jakości rozwiązań. Optymalne okazały się umiarkowane wartości mutacji i wysokie krzyżowanie, a wyłączenie mechanizmu (rzekomego ulepszenia) „wnuka" skróciło czas obliczeń bez pogorszenia jakości. Konfiguracja ta jest najbardziej efektywna obliczeniowo.

200 zadań i 10 maszyn

Na podstawie przeprowadzonej analizy dla problemu PFSP składającego się z 200 zadań i 20 maszyn, dostajemy najlepszy wynik (best_makespan): 10592. To dzięki konfiguracji:

- n_pop = 100
- n_gen = 1000
- p_mut = 0.03
- p_cx = 0.9
- selection_method = tournament
- crossover_method = pmx
- local_search_tries = 3
- elite = 3
- p_make_grandchild = 0

Najlepszy wynik uzyskaliśmy przy dużej liczbie generacji i umiarkowanej populacji, co wskazuje, że kluczowe znaczenie ma dłuższa ewolucja populacji. Umiarkowane prawdopodobieństwo mutacji i wysokie krzyżowanie pozwoliły zachować równowagę. Brak mechanizmu „wnuka" przyspieszył obliczenia bez utraty jakości, dzięki czemu konfiguracja ta jest najbardziej skuteczna i stabilna dla dużej instancji.

200 zadań i 10 maszyn

Na podstawie przeprowadzonej analizy dla problemu PFSP składającego się z 100 zadań i 10 maszyn, dostajemy najlepszy wynik (best_makespan): 5516. To dzięki konfiguracji:

- n_pop = 100
- n_gen = 1000
- p_mut = 0.03
- p_cx = 0.9
- selection_method = tournament
- crossover_method = pmx
- local_search_tries = 3
- elite = 3
- p_make_grandchild = 0

Najlepszy wynik dostaliśmy przy dużej liczbie generacji i umiarkowanej populacji. Stabilna konfiguracja z niskim prawdopodobieństwem mutacji i wysokim krzyżowaniem pozwoliła na skuteczną eksplorację przestrzeni rozwiązań. W każdym przypadku lepsze wyniki otrzymaliśmy bez mechanizmu wnuka.

Wnioski

W każdym z trzech przypadków dla najlepszych wyników te parametry pozostały niezmienne:

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

Taka konfiguracja okazała się najbardziej uniwersalna, zapewniając jednocześnie wysoką jakość rozwiązań i stabilność działania algorytmu.