Sprawozdanie

Ćw. 11 – Algorytm poszukiwania harmonii (HS)

Filip Horst 311257

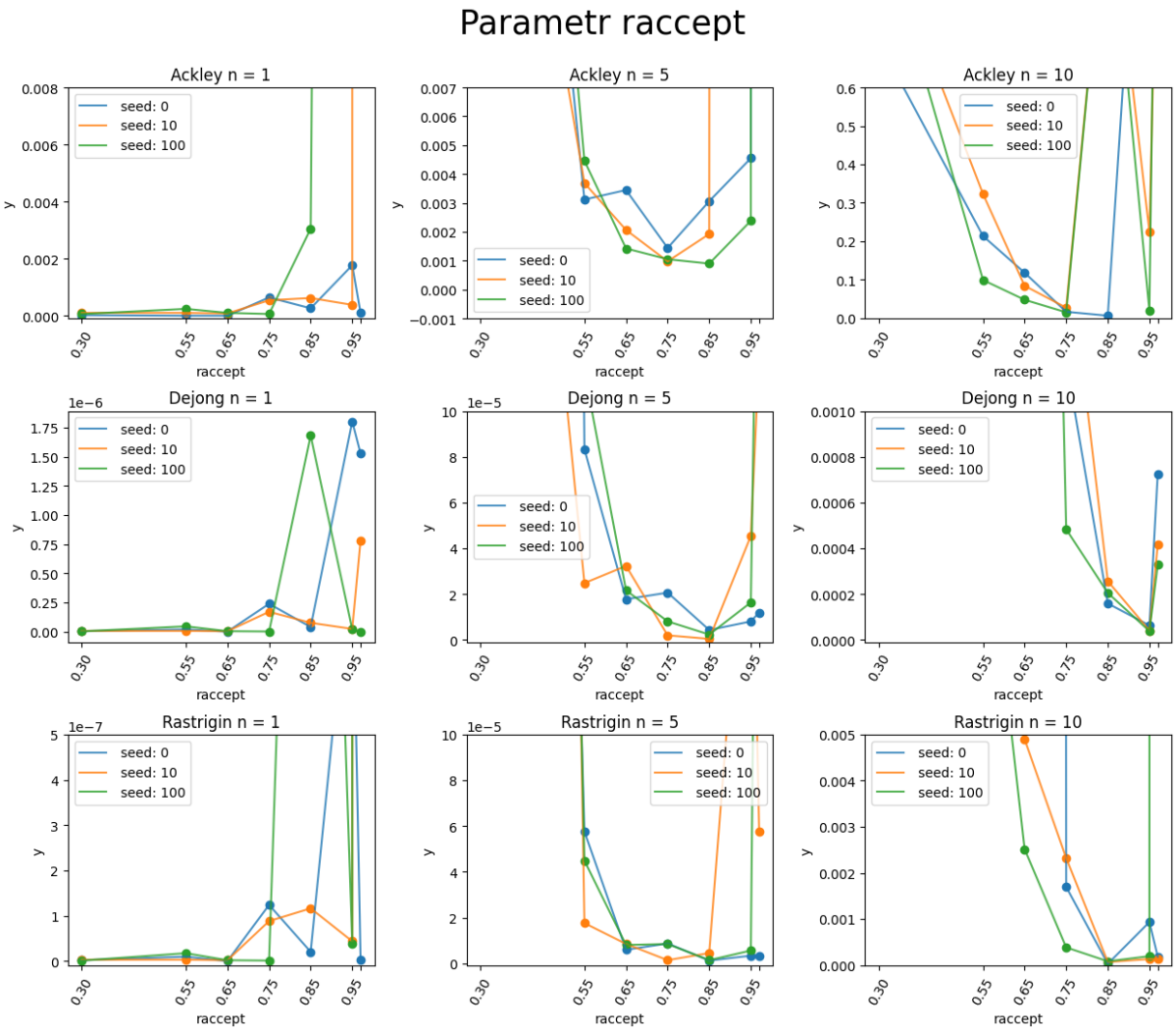
1 Analiza zależności parametrów z wynikami – analiza jednoczynnikowa

Pierwszym etapem podobnie jak w poprzednich ćwiczeniach była analiza jednoczynnikowa. Domyślnymi ustawieniami były:

- Accept_rate (raccept) 0.95
- Pa_rate (rpa) 0.7
- Pa_band_coef 200
- Popsize 30
- Niter 500

Wszystkie testowane zadania posiadały optimum w punkcie 0, co pozwala na utożsamienie wyniku y z błędem (większe y = większy błąd). Na wykresach podobnie jak w poprzednich ćwiczeniach przedstawione zostały zależności wyników od ustawień dla trzech różnych ziaren losowych – takie podejście jest trudniejsze w wizualizacji i analizie, ale w przeciwieństwie do użycia miar statystycznych nie trzeba się obawiać, że jakaś zależność zostanie przypadkowo ukryta (jak przy użyciu średniej), bądź utrudniona w analizie (jak przy użyciu mediana + odchylenie itp.).

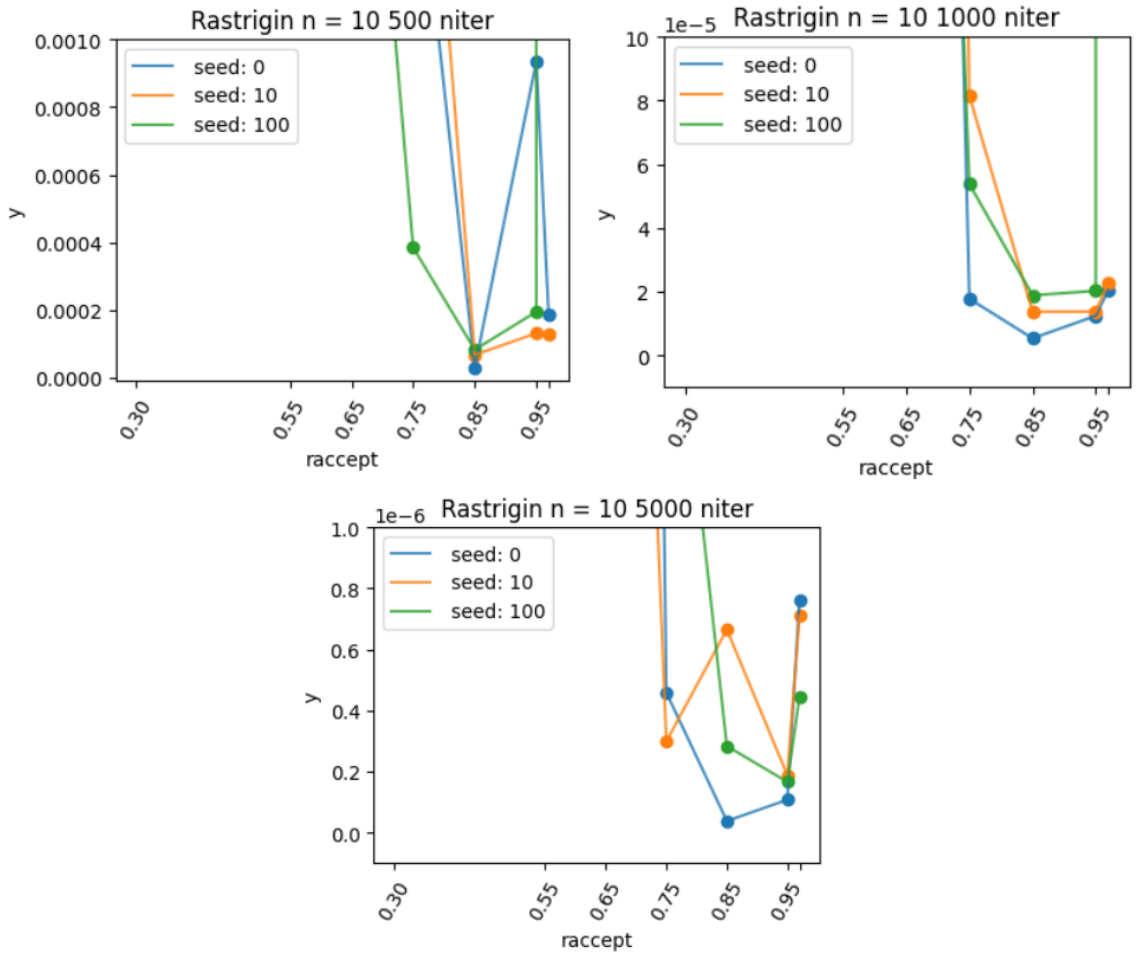
1.1 Accept_rate (raccept)



Zależność wyniku y od parametru accept_rate (raccept) dla różnych zadań

We wszystkich przypadkach oprócz 1-wymiarowej f. Rastrigina zależność wyniku od ustawień raccept przejawia charakter paraboliczny w tym sensie, że dla zbyt małych i zbyt dużych ustawień wynik zaczyna się pogarszać – istnieje pewien przedział dobrych wyników będący kompromisem. Przedział ten jest różny dla konkretnych zadań (inna szerokość i umiejscowienie), ale można powiedzieć, że najbardziej uniwersalnym ustawieniem było 0.85. Inną obserwacją jest to, że dla funkcji o mniejszych wymiarowościach dobrze sprawdzały się również wartości niższe od 0.85 np. 0.75, natomiast dla 10 wymiarowych funkcji Rastrigina i DeJonga dobre okazywały się wartości wyższe aż do 0.95 (0.97 już było bardzo słabe dla każdego przypadku). Ta zależność nie jest jednak w pełni zachowana dla 10-wymiarowej funkcji Ackley'a.

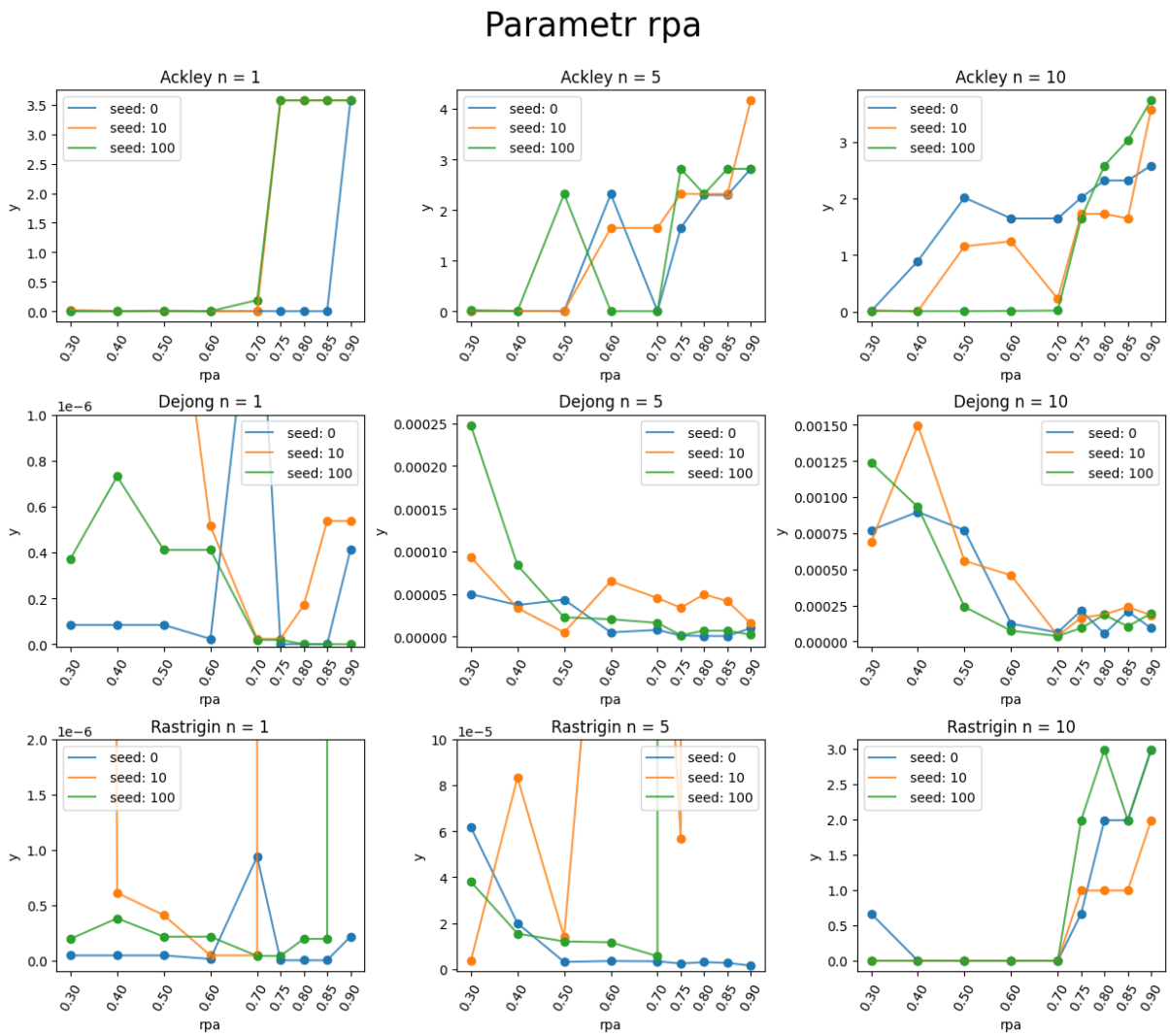
Potencjalną hipotezą jest to, że dla łatwiejszych zadań optymalizator jest w stanie dojść w okolice rozwiązania niezależnie od ustawienia raccept (pomijając bardzo skrajne wartości jak np. 0.3), a wtedy zwiększona eksploracja na skutek większej losowości (powodowanej niskim raccept) pomaga znaleźć jeszcze lepsze rozwiązanie na bardzo małym obszarze. Pewną przesłanką zgodną z ta hipotezą są rzędy wielkości osi Y na wykresach powyżej. Dla łatwiejszych zadań, gdzie skuteczniejsze są niższe ustawienia y są rzędu 1e-5, bądź nawet niższe, podczas gdy dla trudniejszych wyniki są dziesiątki, a nawet setki razy wyższe, czyli element eksploracji nie był jeszcze potrzebny.



Zależność wyniku y od raccept przy różnych niter dla f. Rastrigina n=10.
* osie y się różnią, zostały dobrane tak by ilustrować charakter zależności

Dodatkowym badaniem było sprawdzenie jakości wyników w różnej liczbie iteracji. Obserwacje są częściowo przeciwne wstępnej hipotezie, ponieważ nawet dla bardzo dużej liczby iteracji charakter wykresu się nie zmienia. Oznacza to, że na optymalną wartość accept_rate wpływ ma przede wszystkim kształt optymalizowanej funkcji.

1.2 Pa_rate (rpa)

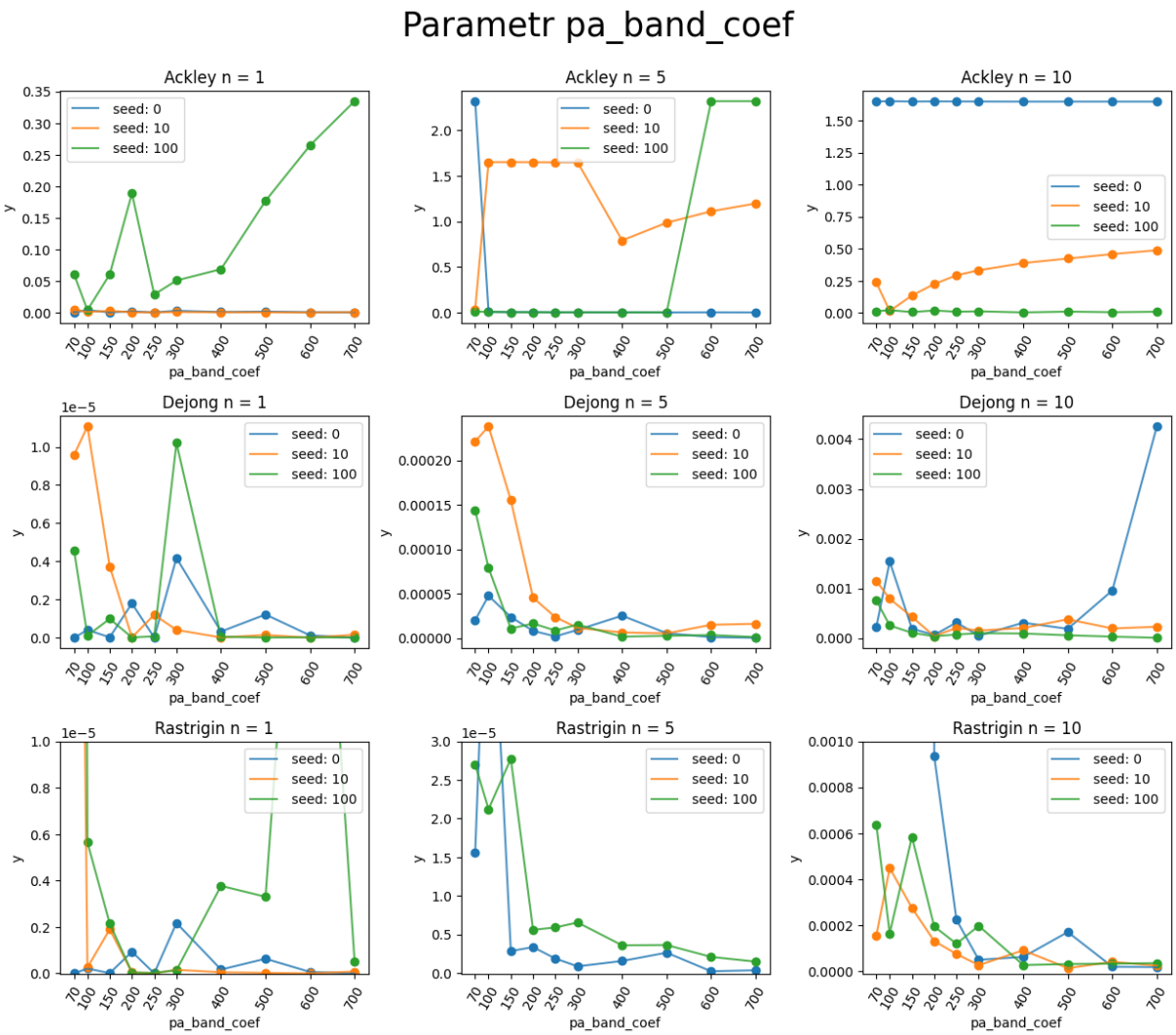


Zależność wyniku y od parametru pa_rate (rpa) dla różnych zadań

Parametr pa_rate podobnie ma pewien optymalny zakres wartości. W tym przypadku jednak od razu w oczy rzuca się duża zależność od zadania – np. dla 10 wymiarowych funkcji DeJonga i Rastrigina w pierwszym przypadku występuje zależność w stylu „im więcej tym lepiej” (do wartości ok. 0.7), natomiast dla Rastrigina „im więcej, tym gorzej” (dla wartości > 0.7). Można to zinterpretować jako to, że w funkcji DeJonga lepiej sprawdza się spokojne podejście z dodawaniem szumu do istniejących rozwiązań, natomiast dla funkcji Rastrigina użyteczniejsza jest dodatkowa eksploracja wynikająca z losowania rozwiązań. Hipotezą jest tutaj trudność funkcji – f. Rastrigina ma więcej minimów lokalnych, więc dodatkowy czynnik eksploracji pomaga zapobiegać utykaniu,

Najlepiej jest wykonać wstępne przeszukanie ogólne dla wykonywanego zadania, by sprawdzić jaki jest jego charakter i na tej podstawie wybrać dobrą wartość.

1.3 Pa_band_coef

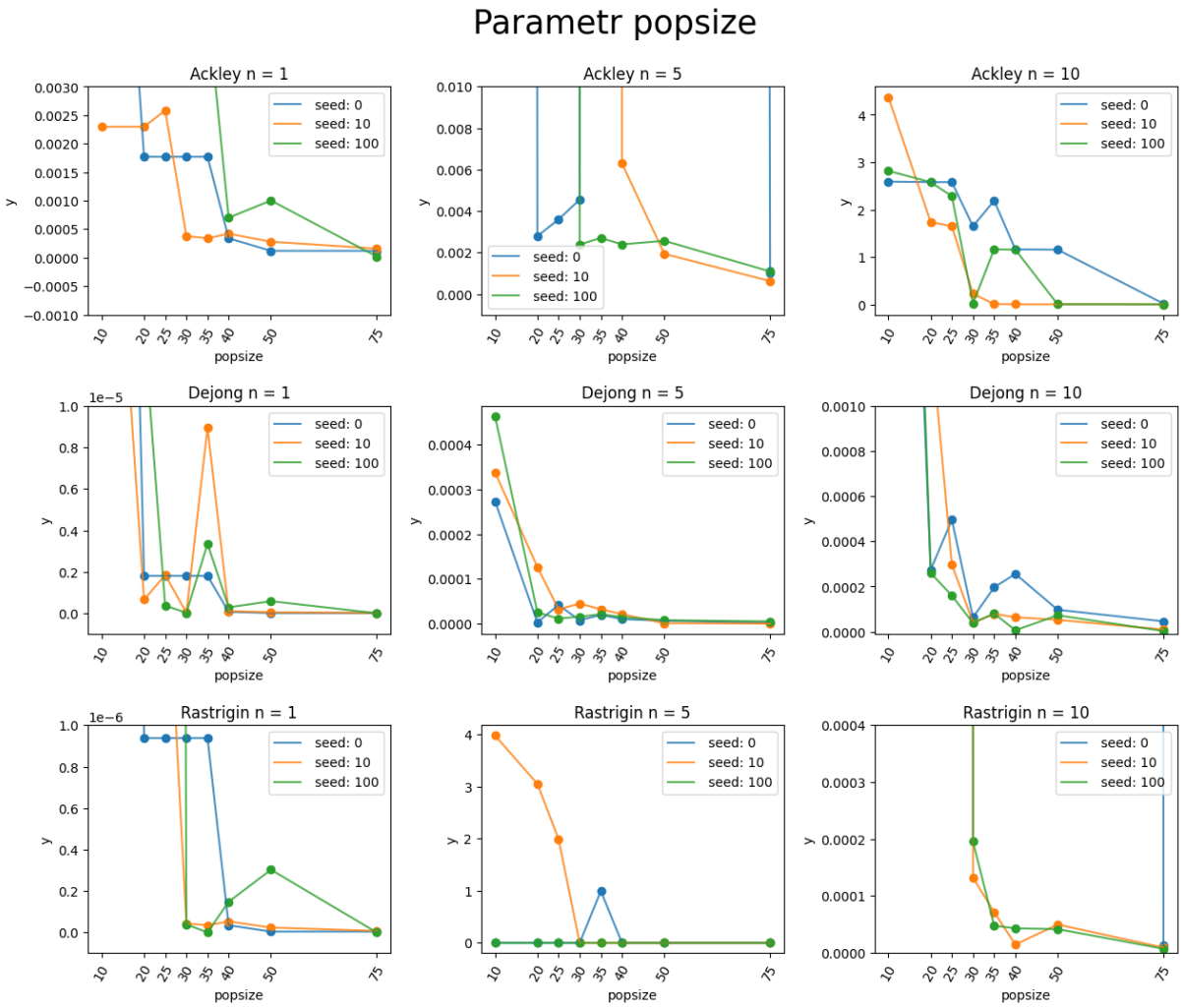


Zależność wyniku y od parametru pa_band_coef dla różnych zadań

W przypadku tego parametru większość zadań przejawia zależność typu „im więcej, tym lepiej”. Istnieją jednak przypadki, gdzie zwiększanie pogarsza Y (np. DeJong n=10). Jeśli jednak występują takie przypadki to tylko dla pojedynczych ziaren losowych. Jest to logiczne, ponieważ wyższy pa_band_coef można utożsamiać z mniejszych krokiem zmian, a więc niefortunny strzał na początku (zależny od ziarna) sprawi, że dla mniejszych kroków populacja nie zdąży się zbiec do dobrego wyniku w czasie ograniczonym iteracjami.

Podsumowując, wyższe wartości parametru dadzą lepsze wyniki, ale trzeba zachować umiar by nie spowodować zbyt wolnej zbieżności. Można by się było zastanowić nad opłacalnością podejścia hybrydowego, gdzie współczynnik zmienia się z kolejnymi iteracjami.

1.4 Popsize



Zależność wyniku y od parametru popsize dla różnych zadań

W tym przypadku liczebność populacji zdaje się mieć jednoznacznie pozytywny wpływ na wyniki. Dla pewnych wartości widać jednak znaczny spadek opłacalności dalszego zwiększania (mała poprawa i wydłużony czas wykonywania), dlatego nie powinno się na ślepo ustawiać wartości maksymalnej.

Ponadto, można podejrzewać, że dalsze zwiększanie eksploracji zbyt dużą populacją może w pewnym momencie doprowadzić do przybliżenia algorytmu do losowego przeszukiwania. Badanie nie pokazało jednoznacznie takiej zależności, ale powodem może być zbyt mały przeszukany zakres.

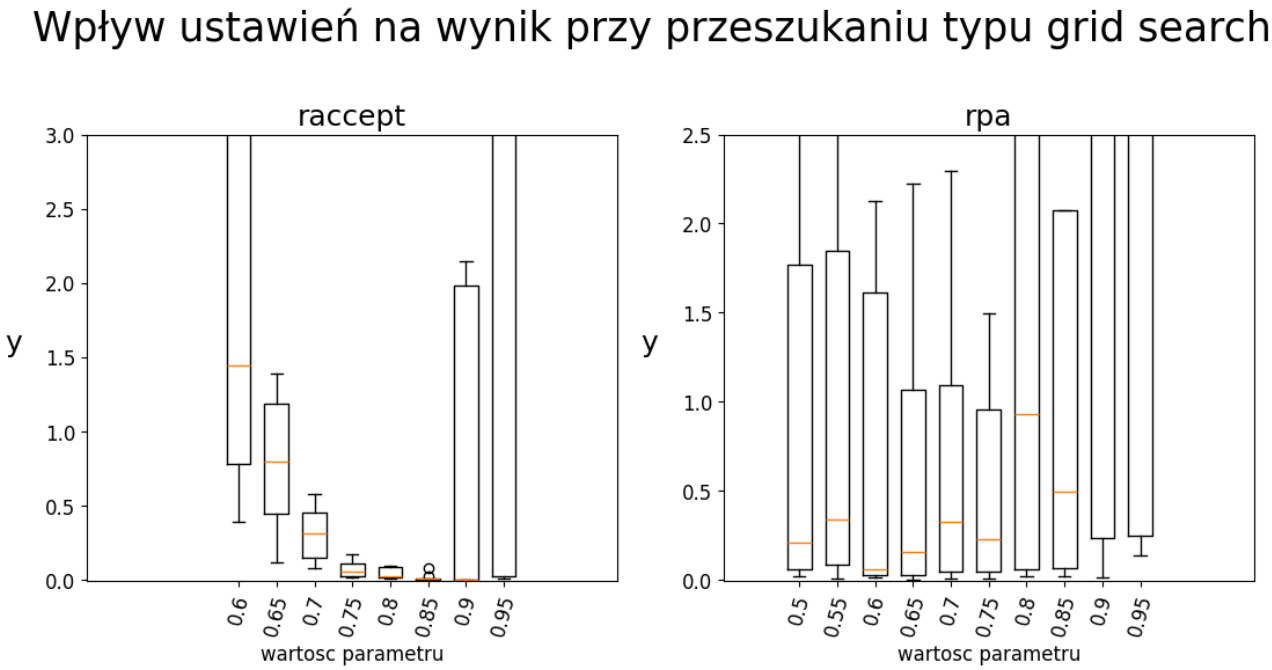
1.5 Niter

Podobnie jak w każdym z poprzednio badanych algorytmów większa liczba iteracji pomagała osiągnąć lepsze wyniki, ale w pewnym momencie poprawa w stosunku do wydłużonego czasu obliczeń stawała się nieopłacalna. Z powodu intuicyjności tego parametru wykresy zostały pominięte.

To co należy podkreślić to, że bardzo dobre wyniki były osiągane już przy 500 iteracjach, czyli dla stosunkowo niewielkiej ich liczby.

2 Podstawowa analiza wieloczynnikowa

Ta analiza obejmuje prosty grid search (przeszukanie automatyczne) parametrów accept_rate (raccept) i pa_rate (rpa) w celu sprawdzenia, czy oddziałują na siebie wzajemnie.



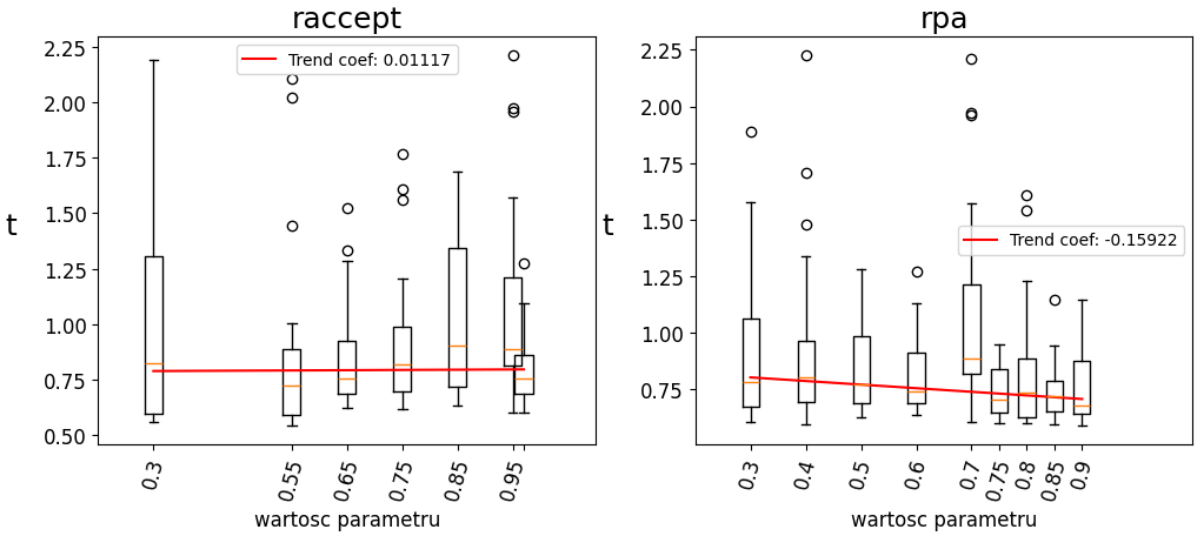
Spośród dwóch zbadanych parametrów zdecydowanie ważniejszy okazuje się być raccept. Łatwo zauważyć, że kiedy jest w swoim optymalnym przedziale (mniej więcej 0.75 – 0.85) to niezależnie od ustawienia rpa wyniki są świetne. Dla parametru rpa po medianach i wartościach minimalnych można zaobserwować, że najlepszy jest szeroki przedział niższych wartości, który jak się okazuje jest bardzo podobny do tego uzyskanego w analizie jednoczynnikowej, jednak nie w pełni – w tej wersji zakres zwracający dobre wyniki jest szerszy.

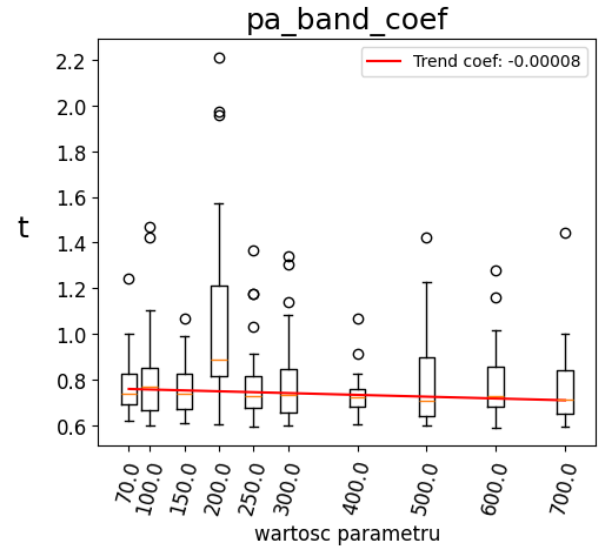
Podsumowując, najlepiej skupić się na badaniu raccept, ponieważ optymalizacja rpa sprowadza się bardziej do sprawdzenia paru ustawień tylko po to, żeby uniknąć kompletnie błędnej wartości jak np. 0.95 – dokładniejsza optymalizacja raczej nie jest opłacalna z powodu małych zysków.

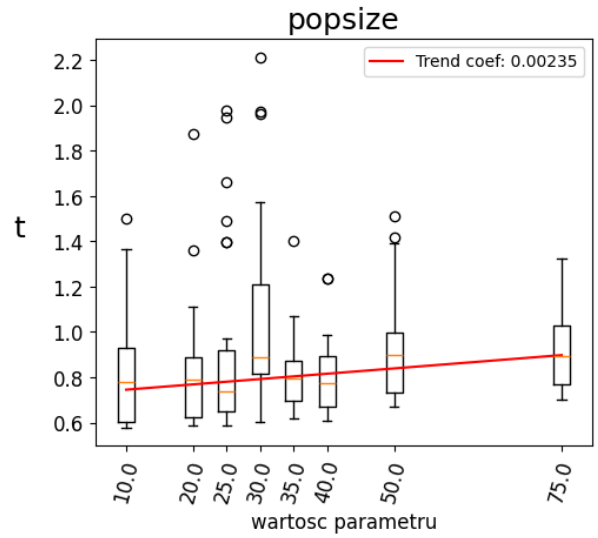
3 Wpływ ustawień na czas wykonania

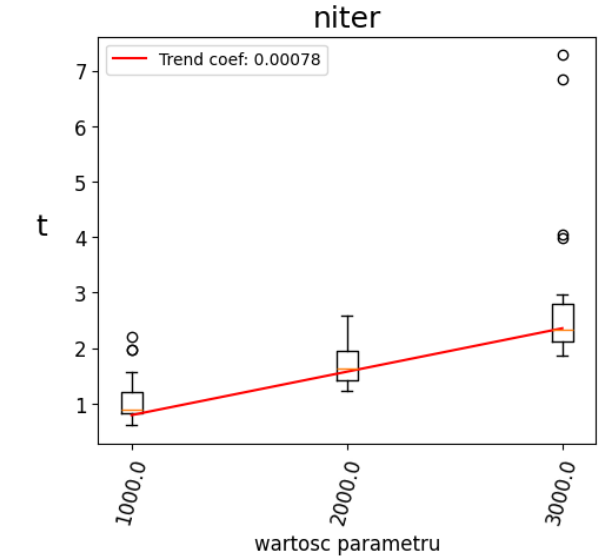
Badanie czasu trwania zostało wykonane na zwiększonej liczbie iteracji równej 1000 (500 więcej), ponieważ przy 500 czasy były na tyle krótkie, że nie dało się odróżnić wpływu parametrów od losowych odchyleń wynikających z fizycznej budowy komputera i współdziałających programów.

Wpływ ustawień na czas wykonywania 1000 iteracji







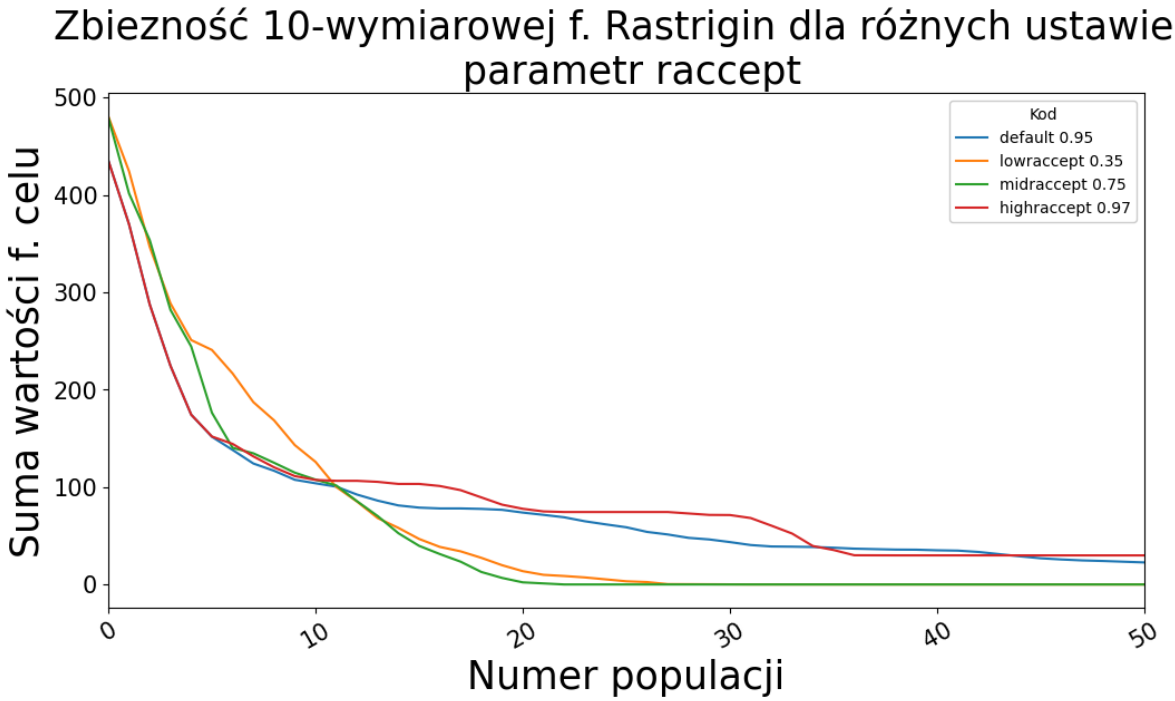


Zdecydowanie największy wpływ na czas wykonanie ma liczba iteracji. Drugi najbardziej wpływowy parametr to zgodnie z oczekiwaniami pop size. Największym zaskoczeniem jest pa_rate, którego wzrost powoduje przyśpieszenie działania. Hipotezą tego zjawiska jest to, że implementacja losowania nowej wartości parametru jest bardziej obciążająca niż funkcja dodająca szum do już istniejącej. Pozostałe parametry można uznać za nie powodujące znaczących zmian w czasie wykonania.

4 Wpływ ustawień na zbieżność populacji

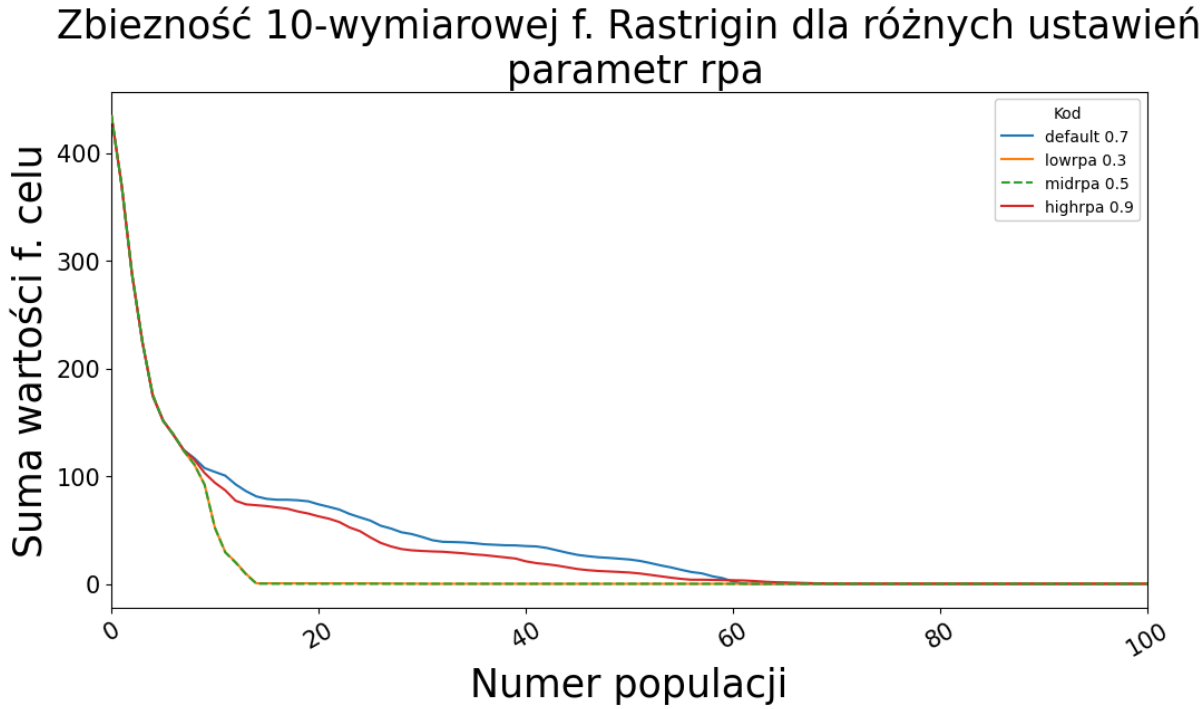
W celu dokładniejszej analizy wykresy zostały zawężone do 50 pierwszych iteracji. Jest wyjątek od tej reguły, ale jest to zaznaczone w opisie.

4.1 Accept_rate (raccept)



Wyższa wartość raccept sprawia, że do kolejnej harmonii przechodzi więcej niezmienionych nut. Naturalnie oznacza to mniejsze zmiany w kolejnych iteracjach, a co za tym idzie wolniejszy zbieg.

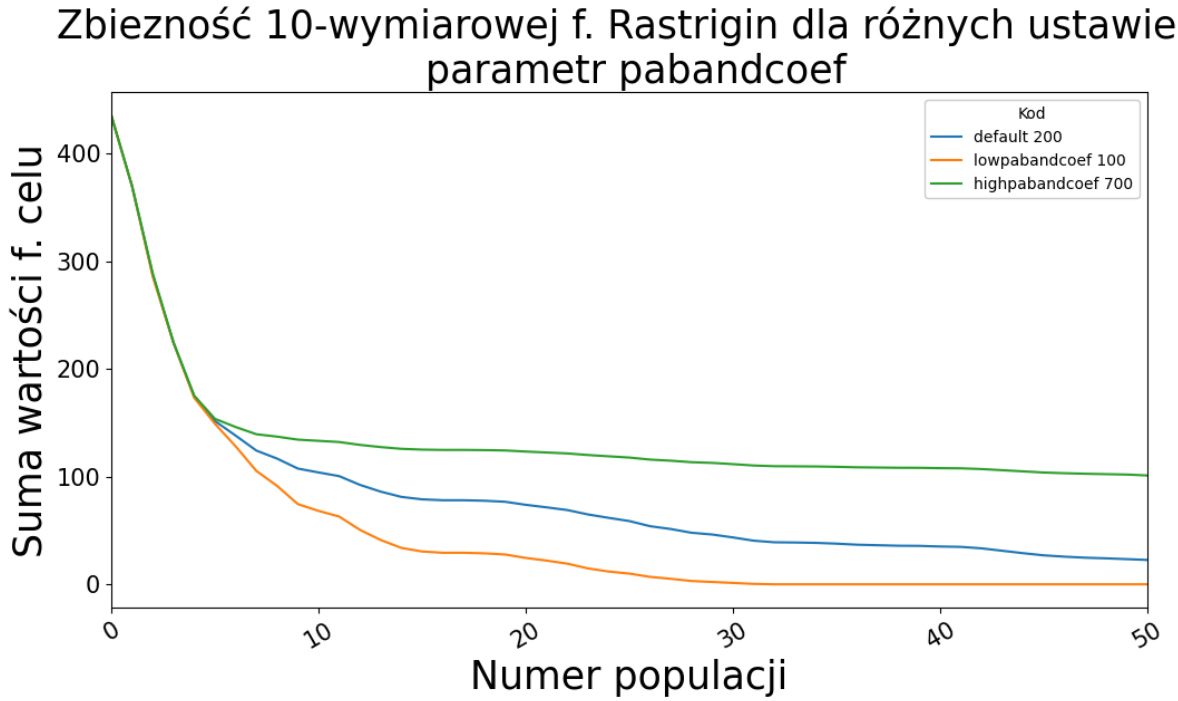
4.2 Pa_rate (rpa)



W tym przypadku zakres zwiększono do 100 iteracji, aby upewnić się czy końcowa zbieżność zmienia się dla ustawień. Ponadto, wykresy niskiego (lowrpa) i pośredniego ustawienia (midrpa) nakładały się na siebie całkowicie, dlatego jeden z nich został zaznaczony linią przerywaną (może być to słabo widoczne na wizualizacji).

Dla pa_rate (rpa) wyższe ustawienie pod kątem zbieżności są lepsze od domyślnego ustawienia 0.7, ale gorsze od niskiego i średniego ustawienia (odp. 0.3 i 0.5). Nie ma więc podstaw do jednoznacznego określenia jakiejkolwiek zależności między rpa, a zbieżnością algorytmu, poza tym że dla pewnego przedziału algorytm działa lepiej (co już wiadomo z poprzednich sekcji).

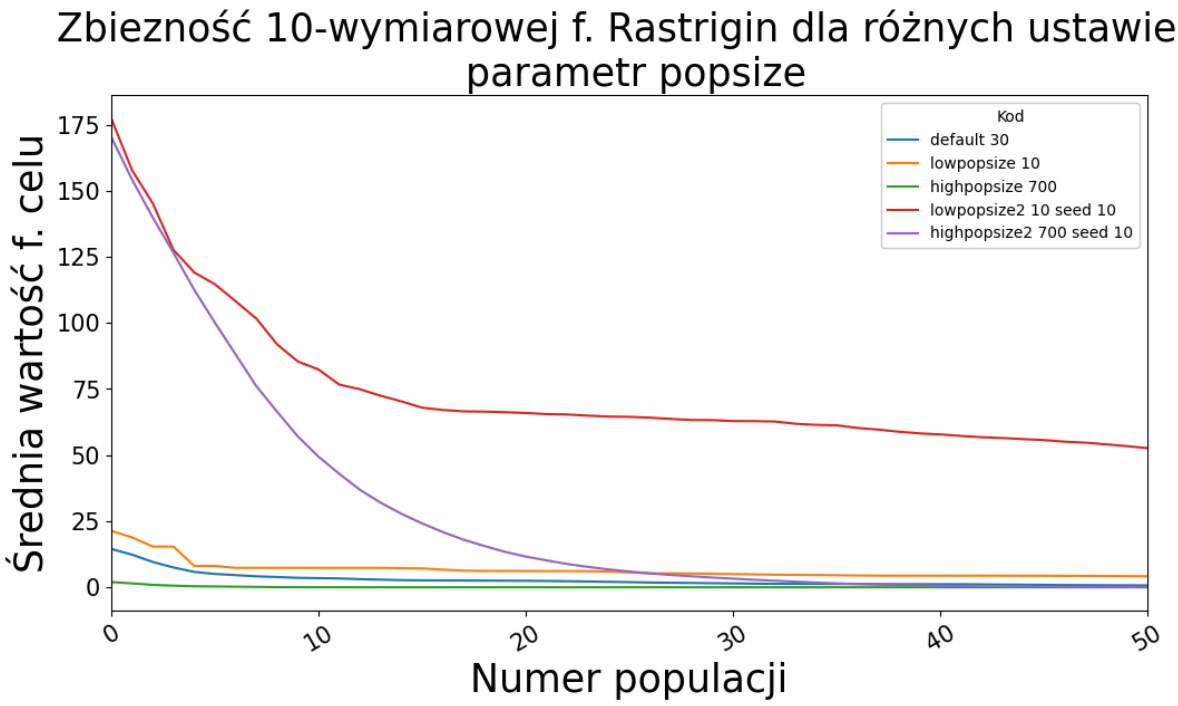
4.3 Pa_band_coef

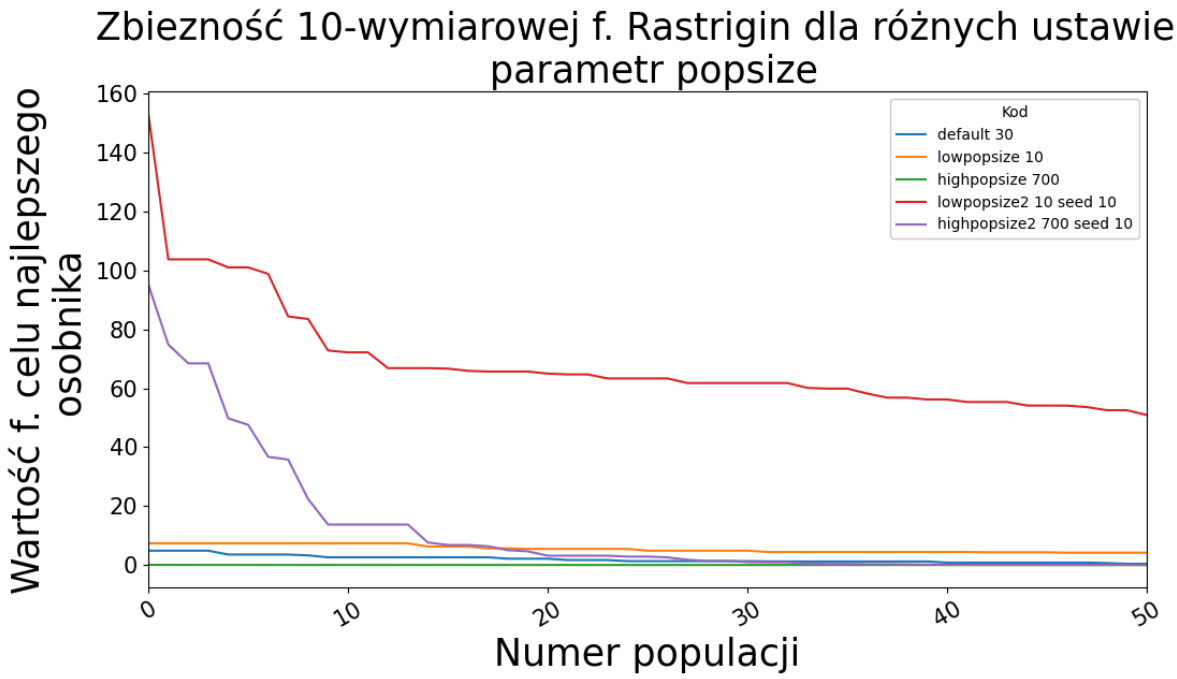


Zgodnie z oczekiwaniami wysokie ustawienie współczynnika, czyli mniejszy krok powoduje wolniejszą zbieżność. Jak wiadomo jednak z wyników i poprzednich badań, populacja przy powolnym kroku jest bardziej konsekwentna w zbieganiu i pozwala na przeszukanie dokładniejszych okolic optimum (zakładając, że algorytm zdąży dotrzeć do optimum w danej liczbie iteracji).

4.4 Popsize

W trakcie badania okazało się, że duży wpływ ma ziarno losowe, dlatego na wykresach zostały zawarte dwie wersje (seed 0 oraz seed 10 zaznaczone w legendzie).





Biorąc pod uwagę, że domyślnie użyte parametry raccept i pa_rate oferowały w miarę zbalansowane podejście do tworzenia kolejnych populacji można zaobserwować, że zwiększenie liczności harmonii pozwala na szybszy zbieg populacji, szczególnie patrząc na przystosowanie najlepszego osobnika. Jest to naturalne zjawisko, ponieważ więcej harmonii to więcej szans na trafienie losu powodującego modyfikację lub dodanie nowej.

5 Porównanie

Funkcja	Wymiary	Accept_rate	Pa_rate	Pa_band_coef	popsize	niter	seed	Y
Ackley	1	0.55	0.7	200	30	500	0	1.465494e-14
Ackley	5	0.95	0.7	200	75	500	10	0.0006212985
Ackley	10	0.95	0.7	200	75	500	100	0.001820866
DeJong	1	0.95	0.8	200	30	500	100	1.232595e-32
DeJong	5	0.95	0.7	700	30	500	0	2.906474e-07
DeJong	10	0.95	0.7	200	75	500	100	3.077976e-06
Rastrigin	1	0.65	0.7	200	30	500	0	0
Rastrigin	5	0.95	0.7	600	30	500	0	2.118647e-07
Rastrigin	10	0.95	0.7	200	75	500	100	6.890985e-06

Tabela zawierające najlepsze wyniki HS pochodzące z analizy jednoczynnikowej dla każdego zadania

Co do samych wyników, zdecydowana większość już jest świetna. Wyjątkiem jest tylko funkcja 5 i 10 wymiarowa Ackley'a, która okazała się być problematyczna dla tego algorytmu. Wyniki dało by się poprawić chociażby stosując większa populację i więcej iteracji, jednak takie poprawy nie zostały wprowadzone, by zachować możliwość porównań do reszty zadań.

W porównaniu do poprzednio badanych algorytmów poszukiwanie harmonii na pewno radzi sobie gorzej od algorytmu rojowego. Sam proces działania HS bardzo przypomina działanie AE, jednak w uproszczonej wersji, co może być powodem trochę mniejszej skuteczności HS.

Podsumowując, algorytm poszukiwania harmonii można ocenić jako nowoczesne połączenie prostoty symulowanego wyżarzania z podejściem algorytmu ewolucyjnego, czyli prosty i bardzo zrozumiały algorytm, który pozwala błyskawicznie osiągnąć całkiem dobre (ale nie najlepsze) wyniki.

6 Podsumowanie wszystkich badań

Zbadane algorytmy były różnorodne i wykazywały się różną skutecznością dla wybranych funkcji testowych, jednak można je krótko podsumować:

- Symulowane wyżarzanie

Przestarzały algorytm, który nie jest już w stanie konkurować z nowszymi i bardziej wykwintnymi rozwiązaniami. Wciąż trzeba docenić jego nie najgorszą skuteczność w stosunku do prostoty działania. Aktualnie jedynym praktycznym zastosowaniem prawdopodobnie są projekty edukacyjne.

- Algorytm ewolucyjny

Bardzo skuteczny i zrozumiały algorytm. Pozwolił osiągnąć prawie idealne rozwiązania w krótkim czasie. Kluczową cechą jest możliwość dostosowywania tego algorytmu do bardzo różnorodnych zadań – różne metody kodowania, skalowanie, selekcja, krzyżowanie, mutacje itd. pozwalają na naturalne dostosowanie algorytmu do działania w dowolnych warunkach. Największą zaletą jest możliwość działania w zadaniach, gdzie rozwiązanie nie jest liczbą – np. kod binarny, czy permutacje.

- Algorytm rojowy

Zdecydowany zwycięzca spośród zbadanych dla tych funkcji testowych. Oferował najszybsze i najdokładniejsze wyniki. Jest prostszy, a co za tym idzie prawdopodobnie możliwości jak chodzi o różnorodne zadania mogą być mniejsze niż AE, ale w funkcjach gdzie poszukiwany jest min/max określony liczbą, wydaje się on być najlepszym wyborem, w szczególności dla ciągłych funkcji celu.

- Poszukiwanie harmonii

Mimo swojej uderzającej prostoty (mało łatwo-dostosowywalnych do zadania parametrów) oferuje bardzo dobre wyniki dla większości zadań. Jak jednak dowiodła funkcja Ackley'a potrafi być zawodny, dlatego należy go umieścić w rankingu niżej niż AE, czy PSO. Mimo tego, łatwość zrozumienia i użycia wciąż sprawiają, że warto rozważyć ten algorytm do zadań optymalizacyjnych, ponieważ szczególnie dla prostszych funkcji może się okazać bardziej opłacalny od reszty.