Wrocław 2021.04.21

Autor: *Jakub Belter*
Nr indeksu: ***<263277>***

Raport z prac przeprowadzonych w ramach laboratoriów 1-3

1 Przeszukiwanie losowe z modyfikacją

Metodę Przeszukiwania Losowego (ang. *Random Search*, RS), rozwinąłem w sposób opisany poniżej.

1.1 Przesłanki

RS jest metodą, która nie posiada pamięci i w każdej iteracji losuje nowe rozwiązanie. Stąd wynika, że możliwości eksploracji są duże, natomiast nie uwzględnia ona eksploatacji.

W związku z powyższym, została wykonana hybryda RS oraz algorytmu zachłannego (*RS with Greedy*, RSG). Dzięki temu, w każdej iteracji RS będzie następować eksploatacja zapewniana przez algorytm zachłanny.

1.2 Zaproponowana modyfikacja

Zaproponowana modyfikacja RS polega na uruchomieniu algorytmu zachłannego za każdym razem, kiedy RS losuje nowe rozwiązanie. W badaniach rozważane są następujące sposoby użycia algorytmu zachłannego.

- Pojedyncze uruchomienie dla całego genotypu.
- Czterokrotne uruchomienie dla całego genotypu.
- Uruchomianie algorytmu zachłannego tak długo, jak wartość dopasowania poprawiła się o ϵ. W trakcie badań ten hiperparametr został ustawiony na wartość 0.1.

Treść zaproponowanego algorytmu zachłannego przedstawia Pseudokod 1.

Pseudokod 1 Wykorzystywany w RSG algorytm zachłanny

```
1:  prevBest = Double.MinValue
2:  function OPTIMIZEGREEDY(OptSolution)
3:      curBest ← OptSolution;
4:      while fitness(prevBest) - fitness(curBest) > ϵ do;
5:          Order ← GetRandomOrder(size(OptSolution));
6:          for each gene ∈ Order do
7:              curTest ← curBest;
8:              curTest[gene] ← ¬curTest[gene];
9:              if fitness(curTest) > fitness(curBest) then
10:                 curBest ← curTest;
11:                 prevBest ← curTest;
12:             end if
13:         end for
14:     end while
15:     return OptSolution;
16: end function
```

W zaproponowanym RSG zachowano główną ideę działania RS – co iterację losowane jest nowe rozwiązanie, które zastępuje najlepsze rozwiązanie jeśli ma od niego wyższą jakość. Wtedy zostaje

uruchomiony algorytm zachłanny na najlepszym rozwiązaniu. W zastosowanym algorytmie zachłannym najpierw losujemy kolejność przetwarzania poszczególnych genów (linia 5). Jeśli zmiana wartości rozważanego genu doprowadzi do znalezienia rozwiązania lepszego niż $curBest$, to nowe rozwiązanie zastępuje $curBest$ (linia 10) oraz $prevBest$ (linia 11). Natomiast sama idea poprawiania tj. eksploracji danego rozwiązania zawarta jest w pętli while (linia 4)

1.3 Przeprowadzone badania

W badaniach wykorzystano nastęujące metody:

- RS – standardowa wersja metody RS.
- RSG-1 – RS uruchamiający algorytm zachłanny jeden raz.
- RSG-4 – RS uruchamiający algorytm zachłanny cztery razy.
- RSG-SigImpr – RS uruchomiający algorytm zachłanny tak długo, jak wartość dopasowania w kolejnych iteracjach poprawia się minimalnie o ϵ.

Kryterium zatrzymania: 1000 iteracji.
Każdy eksperyment (para problem-metoda) powtórzono: 25 razy.
Rozważane problemy:

- *OneMax* (długości: 5, 10, 20, 30, 40 50 genów).
- *Order-5 standard deceptive concatenation* (StandardDC) (długości: 5, 10, 20, 30, 40 50 genów).
- *Bimodal-10 deceptive deceptive concatenation* (BimodalDC) (długości: 10, 20, 30, 40 50 genów).
- *Ising Spin Glass* (ISG) (długości: 25, 49, 100, 484 geny).
- *NK fitness landscapes* (długości: 10, 50, 100, 200 genów).

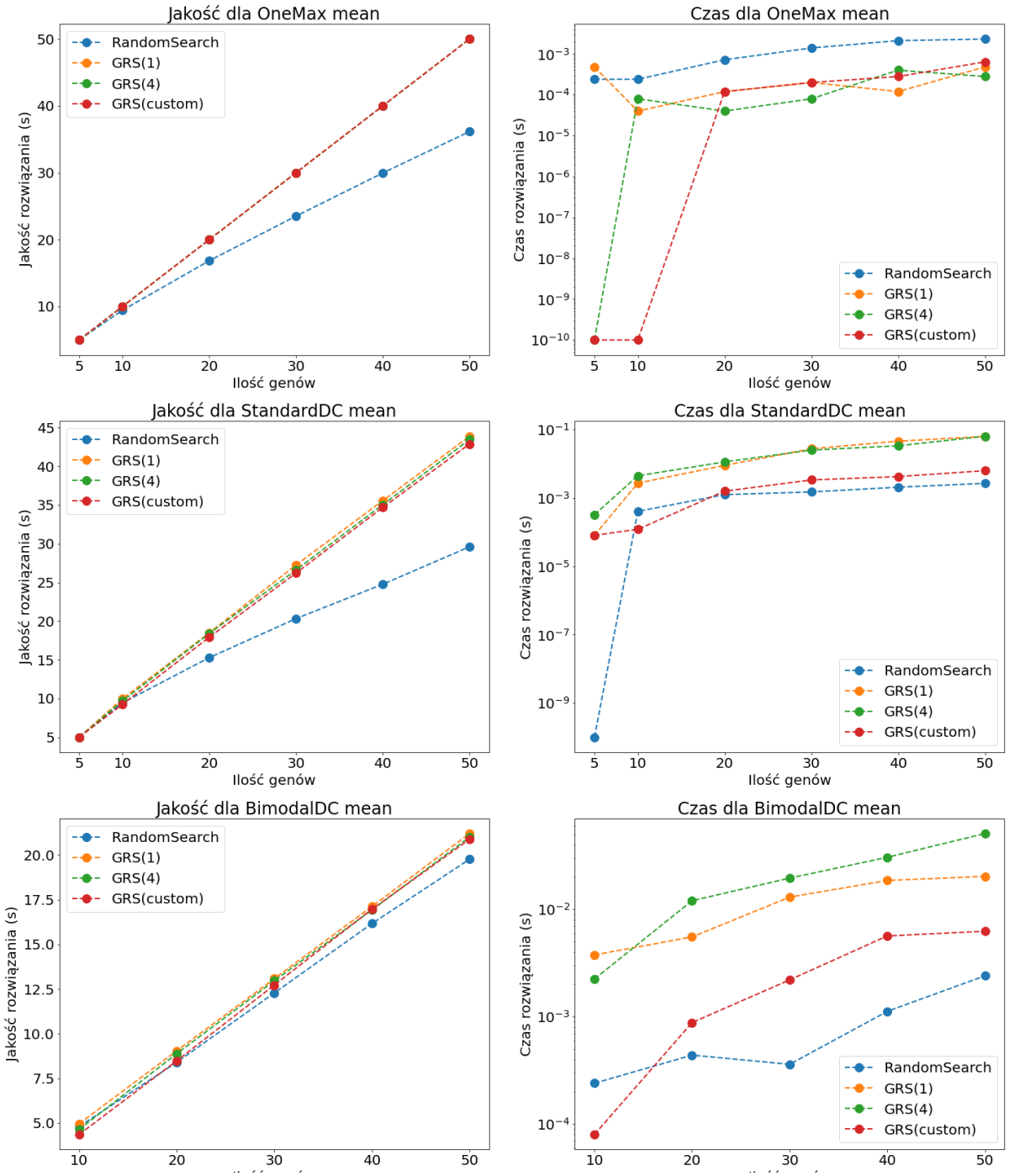
Uzyskane wyniki przedstawiono na Rys 1 i Rys 2.

Jeżeli chodzi o jakość zastosowanych metod dla problemu *OneMax* można uznać, że każdy z algorytmów zachłannych wpada w globalne maksimum dla dowolnej liczby genów. Czyli można stwierdzić, że algorytm zachłanny działa poprawnie jako, że jest to problem testujący czy algorytm jest w stanie eksploatować rozwiązania. Niestety nie można można powiedzieć to samo, jeżeli chodzi o metodę losowego przeszukiwania rozwiązań. Jest to jednak naturalne dla tego algorytmu. Co więcej czas wykonywania jest bardzo podobny do każdego rozmiaru problemu. Co ciekawe czas wykonywania GreedyRS są średnio dziesięciokrotnie szybsze niż w przypadku losowego przeszukiwania.

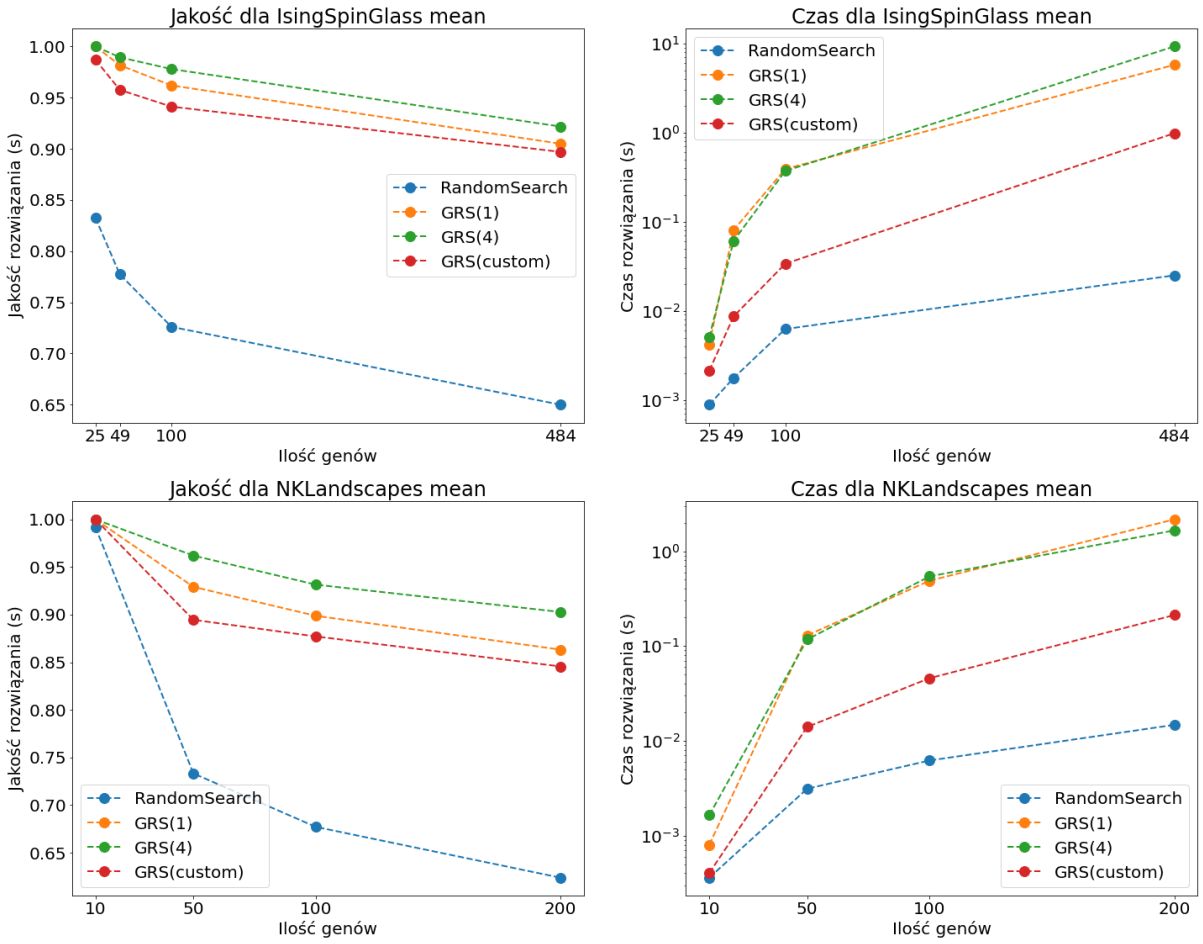
Zanim przejdziemy do kolejnych problemów jest jedna bardzo istotna sprawa. Ze względu na bardzo szybkie wykonywanie algorytmów i problemów z zapisem danych i wyświetlaniem wyników, wszystkie czasy wykonywania, których wartości wynosiły 0 zostały poprawione na 10^{-10}. Również odnośnie czasu wykonania - ilość iteracji wykoniania wszystkich RSG została tak skonfigurowana, aby w kazdym przypadku mieć 1000 iteracji czyli jeżeli używamy RSG-4 to losowanie rozwiązań odbywa się tylko 250 razy, gdyż na każde wylosowanie przypada 4 zachłanne sprawdzenia.

W kolejnym problemie *StandardDC* można zauważyć, iż tym razem algorytm był podatny na zwodzenie, gdyż średni wynik algorytmu jest mniejszy niż optimum globalne. Jednakowoż dalej wynik jest lepszy niżeli RS. Co ciekawe zaproponowana technika greediego jest minimalnie gorsza od reszty wariantów. Najlepszym wariantem jest oczywiście RSG-4, ale można uznać, że różnice są na tyle minimalne że są nieznaczące. Jednakże jeżeli chodzi o czas wykonywania RSG-SigImpr jest podobnie szybki co RS z wynikami zbliżonymi do wyników RSG-1 i RSG-4, co można uznać za duży sukces tej modyfikacji.

Podobną tendencja można zauważyć w kolejnym probleme *BimodalDC*. Z tą różnicą, że RS radzi sobie porównywalnie do RSG w każdym wariancie. Stąd można przypuszczać, że greedy mocno wpada



Rysunek 1: Pierwsze trzy problemy binarne z średnią oceną jakości i czasem wykonania



Rysunek 2: Kolejne dwa problemy binarne z średnią oceną jakości i czasem wykonania

w lokalne optima. Jeżeli chodzi o czas wykonania, to RSG-SigImpr znów zbiega conajmniej dziesięciokrotnie szybciej w porównaniu do reszty algorytmów zachłannych z bardzo podobnym wynikiem.

Wnioski dla kolejnych problemów są analogiczne do poprzednio rozważanych problemów.

Podsumowując, Random Search jest najszybszą metodą i najgorszą jakościowo metodą. Nastepnie pod względem szybkości i jakości mamy RSG-SigImpr, którego wyniki są gorsze niż reszta RSG, ale znacznie lepsze od RS. Następnie mamy RSG-1 i RSG-4 które osiągają najlepsze wyniki (dokładniej RSG-4 osiąga najlepsze wyniki). Czyli najlepsze jakościowo wyniki zwraca RSG-4, a zauważalnie szybciej wyniki zwraca RSG-SigImpr i RS.

1.4 Przedstawione badania a zadania laboratoryjne

Powyższe badania stanowią realizację zadań 1, 2, 3 i 4 z laboratorium 1.

2 Strategie ewolucyjne w przestrzeniach ciągłych

W ramach przeprowadzonych badań zaproponowano modyfikację metody ES(1+1), w której należało dodać zmiane kroku.

2.1 Przesłanki

Metoda ES(1+1) przy stałym kroku mogłaby albo za mocno eksplorować przestrzeń rozwiązań tj. nie mogłaby zejść do odpowiedniego rozwiązania z powodu zbyt dużego kroku i zbyt dużej zmiany. W odwrotnym scenariuszu, gdy krok jest zbyt mały zejście do lokalnego optimum może zająć zbyt dużo czasu.

W związku z powyższym zostały zaproponowane następujące modyfikacje do ES(1+1): modyfikacja kroku na podstawie malejącej temperatury z możliwością jej ponownego zwiekszenia (wzorowane na symulowanym wyżarzaniu z tzw. żarzeniem), która dalej będzie nazywana *Igniting Simulated Annealing* w skrócie *ISA* jak i bez tej możliwości (wzorowane na symulowanym wyżarzaniu bez żarzenia), dalej zwana *Simulated Annealing* w skrócie *SA* oraz obie metody z użyciem wiedzy domenowej nazwane odpowiednio *Domain Knowledge Igniting Simulated Annealing* (DKISA) oraz *Domain Knowledge Simulated Annealing*, analiza historii *n* ostatnich kroków dalej zwana *OneFifth* oraz mechanizm wykrywający utknięcie metody i jej wybicie na podstawie wysyłania zwiadowców, którzy eksplorują przestrzeń rozwiązań, która będzie dalej nazwana *Scouting ES(1+1)* czy też *SES*.

W przypadku pierwszej propozycji tj. *SA* przesłanka jest taka, że z początku posiadamy bardzo duży krok określony przez hiperparametr *startTemp*, dzięki któremu nie wpadniemy w lokalne minimum i wraz z zwiększającą się liczbą iteracji krok będzie się multiplikatywnie zmniejszać o *tempDropCoef*, co umożliwi po wstępnej eksploracji, eskploatacje. Jednakże, może pojawić się problem wybicia z globalnego optimum na rzecz lokalnego optimum, co jest ryzykiem tego podejścia. Drugim natomiast problemem może być zbyt szybkie zbiegnięcie do lokalnego optimum, w którym znalazła się metoda po pierwotnej eksploracji. Na ten problem odpowiada *ISA*. W przypadku kiedy znajdzie metoda ES(1+1) lepsze rozwiązanie, to może sugerować, że gdzieś indziej w przestrzeni rozwiązań są jeszcze lepsze rozwiązania. Stąd krok metody jest zwiększany multiplikatywnie o *ignitingCoef*. Jeżeli jednak nie znajdzie lepszego to tak samo jak w przypadku *SA* zmniejsza temperaturę czyli długość kroku multiplikatywnie. Również jak *SA*, *ISA* krok początkowy jest określany przez *startTemp*.

Ponadto może się pojawić problem ustawienia wartości początkowej wszystkich hiperparametrów i będzie wymagane dostrajanie tych metod adaptacji kroku dla metody ES(1+1). Jednakże zostanie również wzięta pod uwagę modyfikacja, ustawiająca wartości tych parametrów na podstawie wiedzy o domenie problemu. Po krótkich badaniach zostały przyjęte nastepujące wartości parametrów:

- dla *DKISA* : *startTemp* = 5% rozmiaru dziedziny problemu, *temperatureDropCoef* = 0.05, *igniting-Coefficient* = 1.1;
- dla *DKSA* : *startTemp* = 25% rozmiaru dziedziny problemu, *temperatureDropCoef* = 0.05.

Jeżeli chodzi o analizę historii *n* ostatnich kroków (*OneFifth*), została wykorzystana metoda tzw. jednej piątej. Chodzi w niej o to, że jeżeli 1/5 * n ostatnich kroków zawierało poprawienie się metody,

to zwiększamy krok. Jeżeli mniej niż 1/5 to zmniejszamy krok. Jej działanie można uzasadnić intuicją, że jeżeli metoda zmierza w ogólnym kierunku lepszego optimum, niż te które jest to może ona przyspieszyć dochodzenie do niego np. gdyby był ustawiony zbyt mały krok początkowy. Natomiast w przypadku kiedy jest mniej niż 1/5 to jest wiadome, ze nie ma lepszych rozwiązań (wg tej metody) więc można starać się eksploatować znalezione rozwiązanie poprzez zmniejszenie kroku. Zmiana kroku jest multiplikatywnie zmieniana przez użycie parametru *modifier*, który w tym przypadku wynosi 1,5.

Takie rozwiązanie sprawi to, że szybciej podejdzie do maksimum i może wyskoczyć z optimum lokalnych. Jednakże może miec problem zbyt dużego żozpędzeniaśię tj. tak zwiekszenonego kroku optymalizacji, że wartość optymalna nie zostanie osiągnieta, gdyż algorytm będzie zbyt mocno zmieniał rozwiązania. Ale również może sie pojawić problem z brakiem zbyt wolnego i nieoptymalnego zbiegania - jezeli utknelo na samym poczatku w lokalnym optimum to moze miec problem z niego wyjscia.

Natomiast w przypadku ostatniej wspomnianej modyfikacji, czyli *SES* przesłanka jest taka, że chcemy rozdzielić naszą metodę na dwie części. Pierwsza część jest odwzorowana na bazowej metodzie ES(1+1) tj. wykonuje te same operacje co ta metoda. Natomiast druga część tworzy *nScouts* osobnikow zwiadowców, którzy mutują najlepsze rozwiązanie z bardzo dużą sigmą czyli posiadają bardzo duży krok, przez co eksplorują przestrzeń wokół nalepszego rozwiązania. Jeżeli choć jeden zwiadowca znalazł lepsze rozwiązanie niżeli aktualnie najlepsze zwiększa ona krok całej metody. Wysyłanie zwiadowców ma zapobiec zatrzymywaniu się metody w lokalnym optimum, z którego bardzo ciężko się wydostać.

Niestety w przypadku dwóch ostatnich zaproponowanych modyfikacji (*OneFifth* i *SES*) wymagane jest podanie hiperparametrów, które należałoby dopasować. W następnej sekcji zostaną opisane badania wartości tych hiperparametrów.

2.2 Zaproponowane modyfikacje

Zaproponowane modyfikacje ES(1+1) polega na trzech osobnych mechanizmach:

- Modyfikowanie sigmy przy pomocy zmiany temperatury bez żarzenia – *SA*
- Modyfikowanie sigmy przy pomocy zmiany temperatury z żarzeniem – *ISA*
- Inicjalizacja sigmy przy pomocy wiedzy o problemie – *DKSA* oraz *DKISA*
- Analiza ostatnich *n* kroków – *OneFifth*
- Wykrywanie i wybijanie z lokalnego optimum na podstawie zwiadowców – *SES*

2.2.1 Modyfikacje metody ES(1+1) na laboratorium 2

Pseudokod do pierwszych czterech zaproponowanych metod adaptacji kroku (*ISA*, *SA*, *DKISA*, *DKSA*), zostały przedstawiony w pseudokodzie 2 i metodzie **RUN**, której definicja została podana w linijce 1. Cała metoda to ES(1+1) z tym, że zmiany są głównie w funkcji **ADAPT** podana w linijce 20 oraz **INITIALIZE** podana w linijce 12

Wartości hiperparametrów dla poszczególnych adaptacji kroku, które zostały przyjęte to:

- *ISA* : *startTemp* = 1000, *temperatureDropCoef* = 0.05, *ignitingCoefficient* = 1.1;
- *SA* : *startTemp* = 1000, *temperatureDropCoef* = 0.02.
- *DKISA* : *startTemp* = 5% rozmiaru dziedziny problemu, *temperatureDropCoef* = 0.05, *ignitingCoefficient* = 1.1;
- *DKSA* : *startTemp* = 25% rozmiaru dziedziny problemu, *temperatureDropCoef* = 0.05.

W podanym kodzie, można prześledzić działanie całego algorytmu wraz z adapatacją *ISA*. Na początku inicjalizujemy wszystkie wartości parametrów. Następnie wykonujemy główną pętlę, przedstawioną w linii 3. Istotne jest to, że znamy globalne optimum dla tych problemów, gdyz badane problemy są do tego specjalnie dostosowane. Również jako ograniczenie zostało uznana ilość iteracji, aby metoda

Pseudokod 2 Pełny pseudokod wykorzystywanej adaptacji kroku ISA

```
1: function RUN
2:     Initialize();
3:     while fitness(globalOptimum) - fitness(curBest) > ϵ || iterationCount > maxIter do;
4:         candSol ← bestSol;
5:         mutatedSol ← Mutate(candSol);
6:         Adapt(fitness(candSol), fitness(mutatedSol))
7:         if fitness(mutatedSol) > fitness(bestSol) then
8:             bestSol ← mutatedSol;
9:         end if
10:    end while
11: end function
12: function INITIALIZE(size)
13:    Sigmas ← ones(size)
14:    startTemp ← getStartTemp, where startTemp ∈ (1, inf);
15:    tempDropCoef ← getTempDropCoef, where tempDropCoef ∈ (0, 1);
16:    ignitingCoef ← getIgnitingCoef, where ignitingCoef ∈ (1, inf);
17:    MultiplySigmas(startTemp);
18:    bestSol ← generateRandomSolution(size);
19: end function
20: function ADAPT(beforeVal , afterVal)
21:    if beforeVal < afterVal then
22:        MultiplySigmas(ignitingCoef);
23:    end if
24:    if else then
25:        MultiplySigmas(1 - tempDropCoef);
26:    end if
27: end function
28: function MUTATE(sol)
29:    for each solutionGene in sol do
30:        step ∼ N(μ, sigma^2)
31:        solutionGene ← solutionGene + step
32:    end for
33: end function
34: function MULTIPLYSIGMAS(coef)
35:    for each sigma in Sigmas do
36:        sigma ← sigma ∗ coef
37:    end for
38: end function
```

nie działała zbyt długo. W każdym przebiegu pętli algorytmu ES(1+1), kopiujemy najlepsze rozwiązanie, mutujemy je biorąc pod uwagę wewnętrzną zmienną $sigma$ i sprawdzamy wartości dostosowana w adaptacji kroku. Jeżeli wartość się poprawiła, to adaptujemy krok przez przemnożenie $sigm$ przez *ignitingCoef*. W odwrotnym scenariuszu, rozmiar kroku jest zmniejszany multiplikatywnie o *tempDropCoef*. Tutaj warto zwrócić uwagę, że na początku zaczynamy z krokiem - sigmą *startTemp*, który jest parametrem do losowania z rozkładu normalnego wartości zmiany kroku zwanej w kodzie *step*.

Jeżeli chodzi o pozostałe trzy adaptacje, różnice w ich działaniu są następujące:

- *SA* : brakuje inicjalizacji *ignitingCoef* gdyż nie jest on używany linia 16, sigma nie są mnożone przez *ignitingCoef* linia 22;
- *DKISA* : *startTemp* jest inicjalizowane jako 5% zakresu z problemu, linia 14;
- *DKSA* : brakuje inicjalizacji *ignitingCoef* gdyż nie jest on używany, linia 16, sigma nie są mnożone przez *ignitingCoef*, linia 22, *startTemp* jest inicjalizowane jako 2% zakresu z problemu, linia 14;

2.2.2 Modyfikacje metody ES(1+1) na laboratorium 3

Nastepną modyfikacją braną pod uwagę byłaby analiza historii kroków. Jej przebieg można prześledzić w pseudokodzie 3. Jest on analogiczny do metody *ISA* ze zmienioną funkcją adaptacji. To jest jeżeli 1/5 kroków, których liczba została podana w parametrze *archiveSize*, poprawia wynik, to wtedy zwiększamy sigme o określony współczynnik podany w parametrze *multiplier*.

Kolejną zaproponowaną modyfikacją, która miala na celu badanie przestrzeni wokół kandydata rozwiązania było tzw. wysyłanie zwiadowców tj. stworzenie populacji osobników, którzy będą bardzo szeroko przeszukiwać przestrzeń wokół osobnika, poprzez zwiększenie ich wartości kroku czy też ustawienie ich kroku jako osobnego. Ich rozpiętość sprawdzania będzie zwiększać się wraz z ilością iteracji. Jeżeli znajdą oni lepsze lepsze rozwiązanie, to w tym przypadku zwiększą krok podstawowego algorytmu o *scoutingMultiplier*, linia 19. Jeżeli natomiast zwiadowcy nie znajda lepszego rozwiazania to krok głównego algorytmu zostanie zmniejszony o *1 / scoutingMultiplier*, linia 23. Ponadto w kodzie została podana zmienna $maxScoutingWalking$, linia 13, której celem jest ograniczenie zwiadowców do 25% rozmiaru problemu, aby sama metoda nie utknęła poza zakresem problemu. Jest ona ustawiana przez zwracaną wartość logiczną z mutacji, jeżeli ona powiodła się lub nie.

Wszystkie hiperparametry zostały wybrane na podstawie badania podanego w kolejnej sekcji.

2.2.3 GridSearch wartości hiperparametrów dla modyfikacji z 3 laboratorium

W przypadku użycia analizy ostatnich kroków oraz wykrywania lokalnego optimum, podczas badań metody, które zostały zaproponowane, wystąpił taki problem, że z parametrami ustawionymi ź góry"; bez sprawdzenia działały gorzej od NullMutationAdaptation, stąd pojawił się pomysł zrobienia małego GridSearchu parametrów, aby znaleźć te najlepsze do metody. Niestety sprawia to, że zaproponowana metoda wymagała strojenia, co jest generalnie rzecz ujmując minusem.

Na Rys 3 oraz na Rys 4 zostały przedstawione wyniki badań hiperparametrów. Na ich podstawie zostały dobrane parametry na Scouting $nScouts = 1$ oraz $multiplier = 4$. Natomiast dla one-fifth były do parametry $history = 5$ oraz $multiplier = 20$. Widać, że onefifth ma problem z rozpędzeniem się.

2.3 Przeprowadzone badania

W badaniach wykorzystano nastęujące metody:

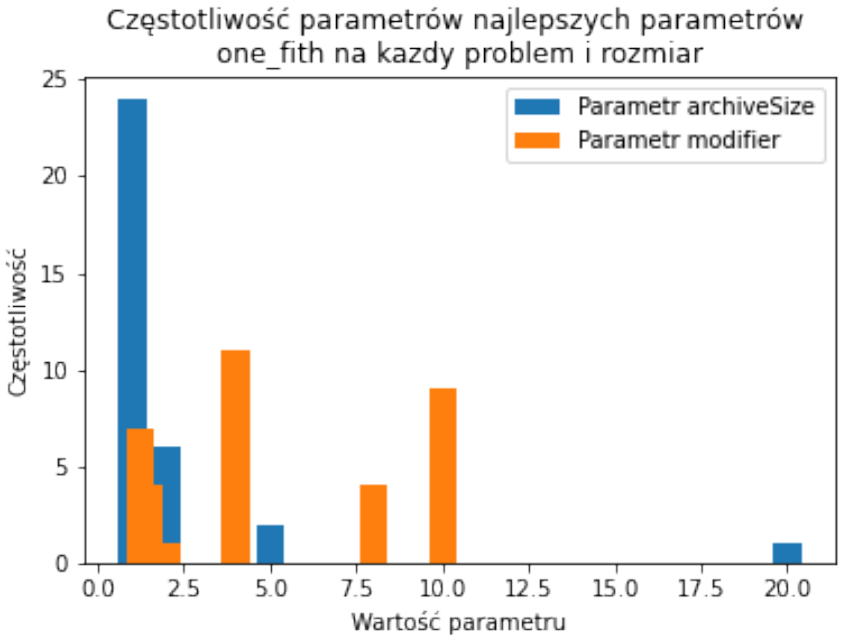
- ES(1+1) – standardowa wersja metody ES(1+1).
- ES(1+1)-DK – ES(1+1) z krokiem ustawionym na podstawie wiedzy o domenie
- SA – ES(1+1) z adaptacją kroku z symulowanym wyżarzaniem bez żarzenia.
- SA-DK – ES(1+1) z adaptacją kroku z symulowanym wyżarzaniem bez żarzenia z wiedzą domenową.

Pseudokod 3 Pełny pseudokod wykorzystywanej adaptacji kroku OneFifth

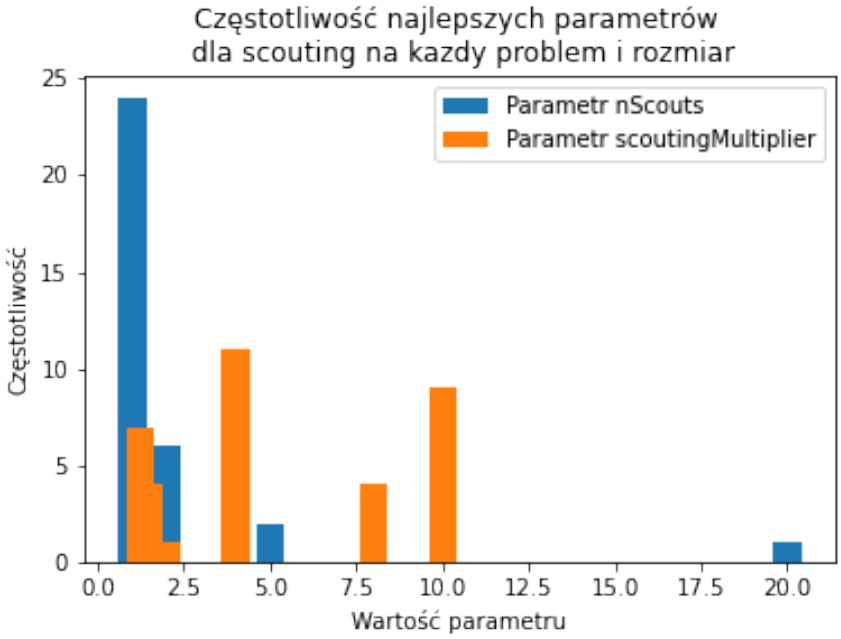
```
1: function RUN
2:     Initialize();
3:     while fitness(globalOptimum) - fitness(curBest) > ϵ || iterationCount > maxIter do;
4:         candSol ← bestSol;
5:         mutatedSol ← Mutate(candSol);
6:         Adapt(candSol, mutatedSol);
7:         if fitness(mutatedSol) > fitness(bestSol) then
8:             bestSol ← mutatedSol;
9:         end if
10:    end while
11: end function
12: function INITIALIZE(size)
13:    Sigmas ← ones(size)
14:    archiveSize ← getArchiveSize();
15:    successes ← new Queue(archiveSize);
16:    modifier ← getModifier();
17:    bestSol ← generateRandomSolution(size);
18: end function
19: function ADAPT(beforeSol , afterSol)
20:    beforeVal ← fitness(beforeSol);
21:    afterVal ← fitness(afterSol);
22:    success ← beforeVal < afterVal;
23:    successes.Enqueue(success)
24:    if successes.count == archiveSize then
25:        ratio ← successes.sum()/archiveSize
26:        if ratio > 1/5 then
27:            MultiplySigmas(modifier);
28:        end if
29:        if else then
30:            MultiplySigmas(1 / modifier);
31:        end if
32:    end if
33: end function
34: function MUTATE(sol)
35:    for each solutionGene in sol do
36:        step ∼ N(μ, sigma^2)
37:        solutionGene ← solutionGene + step
38:    end for
39: end function
40: function MULTIPLYSIGMAS(coef)
41:    for each sigma in Sigmas do
42:        sigma ← sigma ∗ coef
43:    end for
44: end function
```

Pseudokod 4 Istotne częsci pseudokodu wykorzystywanej modyfikacji SES

```
1:  function INITIALIZE(size)
2:      Sigmas ← ones(size)
3:      nScouts ← getNScouts();
4:      scoutingMultiplier ← getScoutingMultiplier();
5:      bestSol ← generateRandomSolution(size);
6:      maxScoutingWalking ← False
7:  end function
8:  function ADAPT(beforeSol , afterSol)
9:      beforeVal ← fitness(beforeSol);
10:     afterVal ← fitness(afterSol);
11:     for i in [1..nScouts] do
12:         scoutSigmas ← iterationNumber * Sigmas;
13:         if ¬maxScoutingWalking then
14:             scoutSigmas ← iterationNumber * Sigmas / 4;
15:         end if
16:         scoutSolution ← bestSol
17:         mutatedScout ← Mutate(scoutSolution)
18:         if fitness(mutatedScout) > fitness(bestSol) then
19:             MultiplySigmas(scoutingMultiplier)
20:             return
21:         end if
22:     end for
23:     MultiplySigmas(1/scoutingMultiplier)
24: end function
```



Rysunek 3: GridSearch dla modelu ES(1+1) z adaptacja 1/5



Rysunek 4: GridSearch dla modelu ES(1+1) z adaptacja typu scouting

- ISA – ES(1+1)z adaptacją kroku z symulowanym wyżarzaniem z żarzeniem/
- ISA-DK – ES(1+1) z adaptacją kroku z symulowanym wyżarzaniem z żarzenia i z wiedzą domenową.
- CMAES – standardowa wersja metoda CMAES.

Kryterium zatrzymania: 1000 iteracji.
Każdy eksperyment (para problem-metoda) powtórzono: 25 razy.
Rozważane problemy:

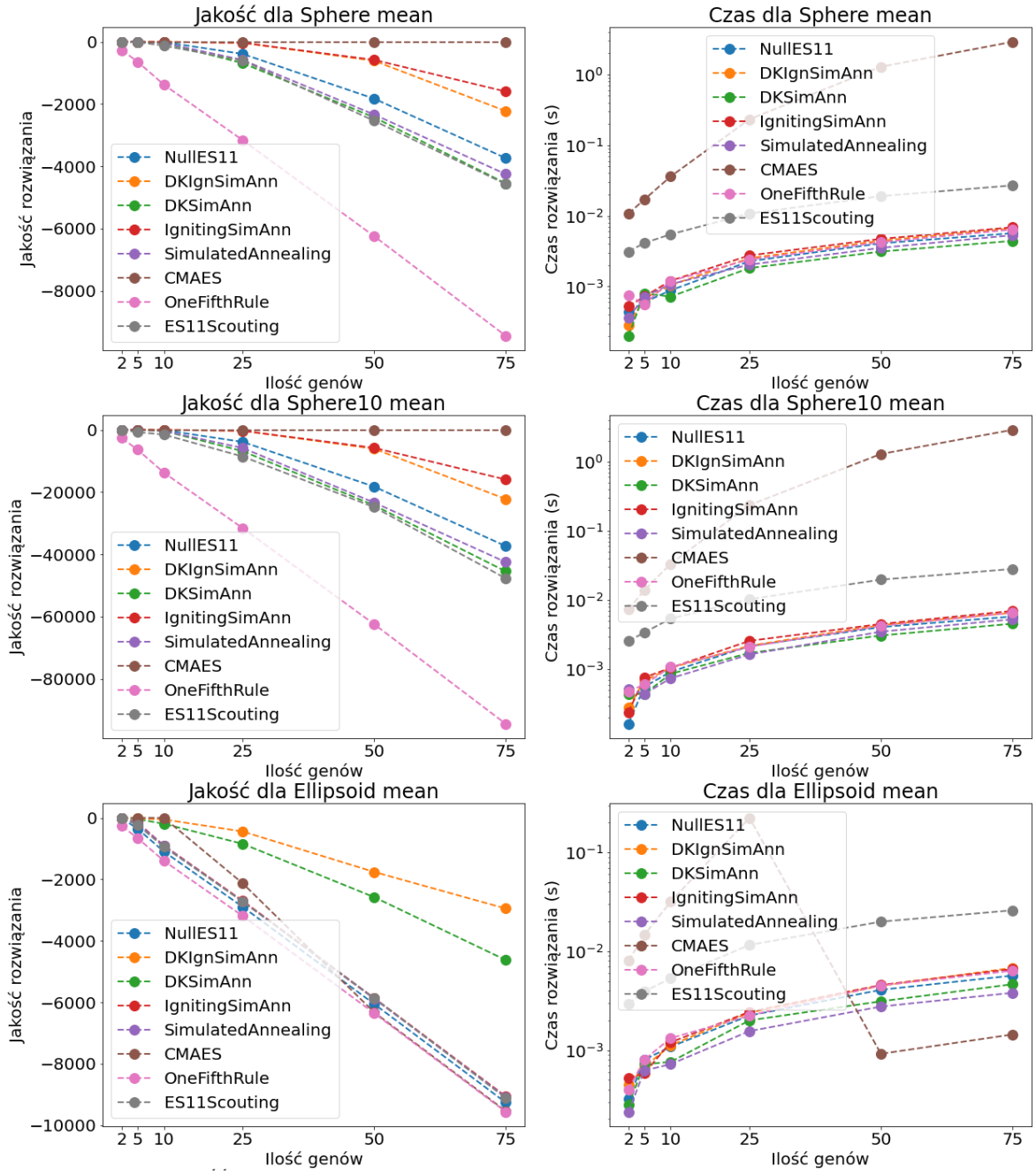
- *Sphere* (długości: 2, 5, 10, 25, 50, 75 genów).
- *Sphere-10* (długości: 2, 5, 10, 25, 50, 75 genów).
- *Ellipsoid* (długości: 2, 5, 10, 25, 50, 75 genów).
- *Step2Sphere* (długości: 2, 5, 10, 25, 50, 75 genów).
- *Rastrigin* (długości: 2, 5, 10, 25, 50, 75 genów).
- *Ackley* (długości: 2, 5, 10, 25, 50, 75 genów).

Uzyskane wyniki przedstawiono na Rys 5 i Rys 6. Istotne w analizie jest to, że CMAES miał nieznany błąd w bilbiotece (ale znany prowadzącym) i nie mógł wykonać się dla dużych problemów ciągłych elipsoidy. Stąd niektóre wartości mogą się wydać dziwne - dalej jednak można zauważyć tendencje złożoności czasowej tego algorytmu w podanych problemach.

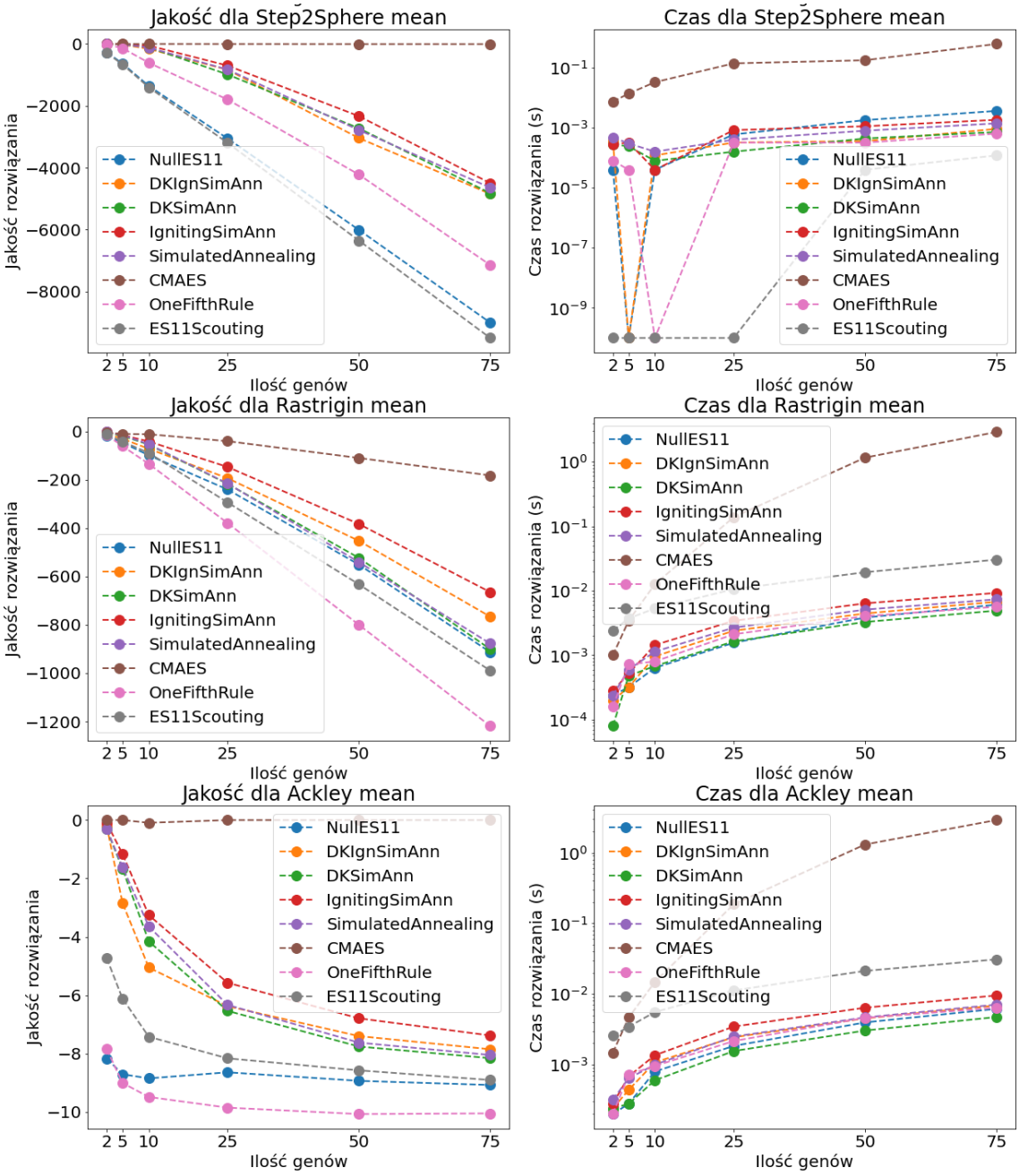
Po poprawieniu parametrów, lepsze wyniki zostały przedstawione na Rys 7 oraz na Rys 8.

Istotnym jest również nakreślenie lini bazowej jakości rozwiązań. Jeżeli coś jest lepsze od *NullES11* to można uznać, że jest do dobra modyfikacja problemu. W innym przypadku jest to raczej rozwiązanie gorsze od pierwotnego.

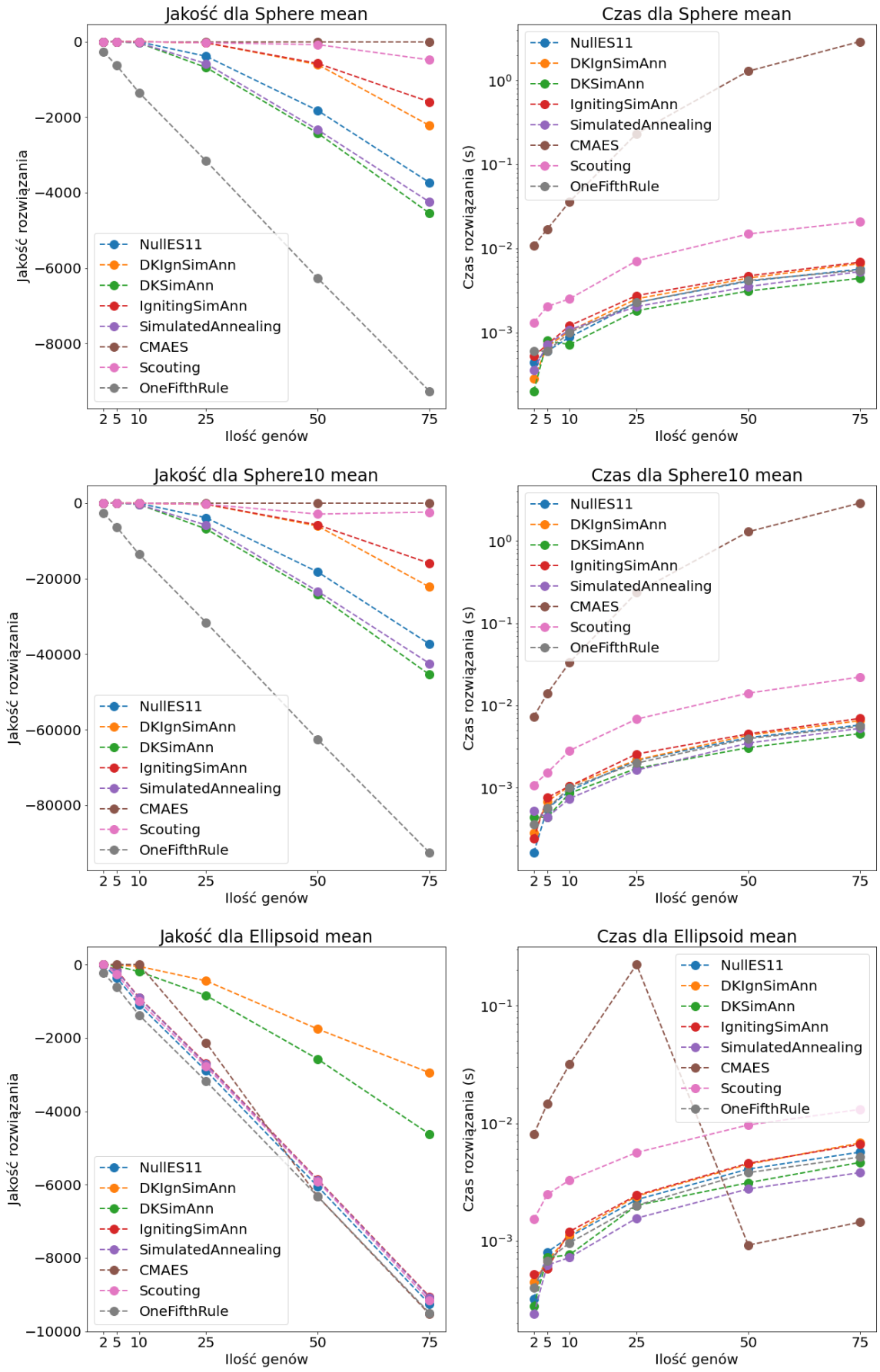
Najlepszym modelem w każdej dziedzinie okazał się CMAES. Niestety jednak w niektorych przypadkach czas jego obliczeń sprawiał, że trzeba się zastanowić czy chce się go wykorzystywać. Jeżeli czas nie jest ważny ten algorytm z pewnością znajdzie dobre wyniki. Jedyny problem jaki napotkał to z wcześniej wspomnianą biblioteką odnośnie liczenia ellipsoid - nie radzi sobie ze zbyt duzymi liczbami.



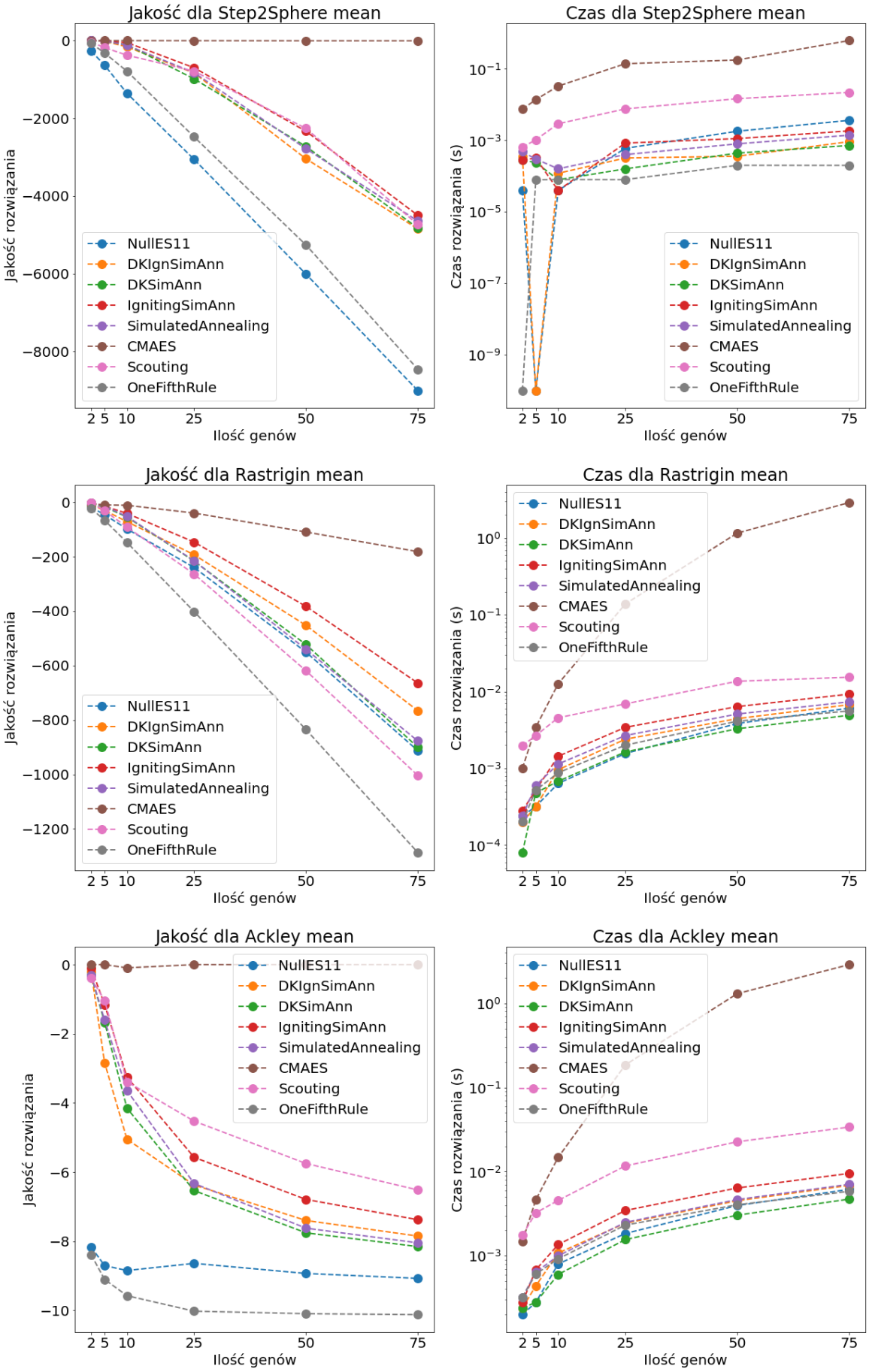
Rysunek 5: Pierwsze trzy problemy ciągłe z średnią oceną jakości i czasem wykonania bez poprawy scoutingu i 1/5



Rysunek 6: Kolejne trzy problemy ciągłe z średnią oceną jakości i czasem wykonania bez poprawy scoutingu i 1/5



Rysunek 7: Pierwsze trzy problemy ciągłe z średnią oceną jakości i czasem wykonania z poprawa



Rysunek 8: Kolejne trzy problemy ciągłe z średnią oceną jakości i czasem wykonania z poprawa

Modele zaproponowane w ramach drugich laboratorium tj. symulowane wyżarzanie z żarzeniem i bez , radziły sobie nadzwyczaj dobrze w porównaniu z metodami z trzecich laboratorium czyli scouting oraz oneFifth.

W wiekszości przypadków wiedza dziedzinowa pogarsza jakość zaproponowanych wyżarzań, z jednym wyjątkiem, którym jest problem Ellipsoid. W tym przypadku algorytmy radzą sobie o wiele lepiej niż reszta innych nawet CMAES (który w tym przypadku wyrzucał wyjątki). Może to być jednak spowodowane dziedziną problemu, która była kompletnie różna od pozostałych problemów. Domyślne parametry dobrze były dostosowane do reszty problemów, więc możliwe, że użycie wiedzy dziedzinowej eliminuje dostrajanie tych algorytmów.

Co ciekawe *zapalający się* algorytm radzi sobie lepiej, niżeli ten jedynie gasnący (symulowane wyżarzanie). We wszystkich problemach ISA radzi sobie lepiej niż SA. Jednakże czas działania ISA jest średnio dłuższy niż SA.

Również ważnym jest to, że ta zaproponowana modyfikacja adaptacji kroku zawsze sobie radzi lepiej niż bazowy algorytm bez adaptacji co sprawia że jest obiektywnie dobrą modyfikacją. Jest również obarczona niewielkim wydłużeniem czasu działania, ale jest to na tyle małe, że nie jest znaczące w działaniu algorytmu.

Co się rzuca na pierwszy rzut oka w metodzie OneFifth, w podanych ograniczeniach tj. 1000 iteracji algorytm jest o wiele gorszy od bazowego algorytmu. Jednakże daje sobie rade lepiej z jednym problemem, a mianowicie z problemem *Step2Sphere*. Może to wskazywać na właściwości możliwego ominięcia minimów lokalnych. Niestety w innych przypadkach nie raddzi sobie tak dobrze. Oznacza to, że ten algorytm nie radzi sobie dobrze z eksploatacją rozwiązań.

Natomiast Scouting po modyfikacji radzi sobie w większości problemów na poziomie ISA, jednakże pomimo nacechowania tego algorytmu do nieutykania w lokalnych minimach to nie radził on sobie z *Rastriginem* i *Step2Sphere*. Aczkolwiek w przypadku *Ackleya* nie miał takich problemów. Mozna więc rzec, że przez mała ilość skautów nie mógl trafić w te lokalne optima, natomiast w przypadku rastrigina, gdzie sa one wieksza juz sobie lepiej z tym radził. Niestety czasowo on ucierpiał przez to gdyż leży pomiędzy CMAESEM a resztą algorytmów, przez co finalna ocena jego jest średnia. Byc może dobrą modyfikacją byłoby dynamiczne wybieranie ilości skautów, które będą szukały przestrzeń np. wraz z liczbą iteracji zwiększa się ich ilość.

Wracając do czasu, reszta algorytmów, tj. tych zaproponowanych w ramach drugiego laboratorium działają w bardzo zbliżonej skali czasowej z tym, że ISA działa najwolniej, a SA działa najszybciej. Wersje z wiedzą dziedzinową zbiegają znacząco szybciej niż swoje wersje bez tej wiedzy, ale niestety tracą one na jakości.

Finalnie najlepszym modelem jest CMAES, tylko ma ograniczenie czasowe, a następnie mamy ISA, dla większości problemów. Jeżeli wiemy coś na temat problemu i wiemy że jest ogromny lub bardzo mały należy użyć DKISA gdyż zbiega znacznie lepiej od pozostałych algorytmów. Następnie mamy algorytm Scouting, który ma podobną jakość co ISA i SA, natomiast działa znacznie wolniej. Noi najgorszym algorytmem jest one fifth, którą można uznać za niedobrą modyfikacje algorytmu. Ta ostatnia pozycja może być spowodowana zbyt słabym rozpędzeniem się tej metody, przez co nie jest ona w stanie zbiec do odpowiedniego miejsca. Tą teorię potwierdza ISA która jest pewnym wariantem adaptacji kroku tyle że zaczyna ona z odpowiedniej wartości. Być może jezeli dać algorytmowi onefifth minimalną wartość sigmy, lepiej by sobie radziła.

2.4 Przedstawione badania a zadania laboratoryjne

Powyższe badania stanowią realizację zadań 1-3 z laboratorium 2 i zadań 1-3 z laboratorium nr 3.

3 Podsumowanie

- **Laboratorium 1. Zadanie 1.** Realizacja w ramach sekcji 1.
- **Laboratorium 1. Zadanie 2.** Realizacja w ramach sekcji 1 Przeprowadzone badania.
- **Laboratorium 1. Zadanie 3.** Realizacja w ramach sekcji 1 Przeprowadzone badania.

- **Laboratorium 1. Zadanie 4.** Realizacja w ramach sekcji 1 Przeprowadzone badania.
- **Laboratorium 2. Zadanie 1.** Realizacja w ramach sekcji 2 Przeprowadzone badania.
- **Laboratorium 2. Zadanie 2.** Realizacja w ramach sekcji 2 Zaproponowane modyfikacje i Przeprowadzone badania.
- **Laboratorium 2. Zadanie 3.** Realizacja w ramach sekcji 2 Przeprowadzone badania.
- **Laboratorium 3. Zadanie 1.** Realizacja w ramach sekcji 2 Zaproponowane modyfikacje i Przeprowadzone badania.
- **Laboratorium 3. Zadanie 2.** Realizacja w ramach sekcji 2 Zaproponowane modyfikacje.
- **Laboratorium 3. Zadanie 3.** Realizacja w ramach sekcji 2 Przeprowadzone badania.