STUDIA INFORMATICA 2007
Volume 28 Number 1 (70)

Dariusz R. AUGUSTYN, Łukasz WYCIŚLIK
Politechnika Śląska, Instytut Informatyki

HEURYSTYCZNY ALGORYTM OPTYMALIZACJI PARAMETRYCZNEJ DEDYKOWANY PROBLEMOM WIELOWYMIAROWYM

Streszczenie. Heurystyczne algorytmy optymalizacyjne znane są od początków rozwoju dziedziny informatyki, jednak stosunkowo niedawno zaczęto proponować algorytmy bazujące na obserwacjach zjawisk w otaczającej nas przyrodzie (ewolucja, poszukiwanie pożywienia przez kolonie wieloagentowe, wyżarzanie w procesach metalurgicznych itp.). Każdy z takich algorytmów charakteryzuje się inną specyfiką przeszukiwania przestrzeni rozważań. Jednym z problemów optymalizacji parametrycznej jest przypadek wielowymiarowej przestrzeni przeszukiwań, gdzie liczba wymiarów osiąga setek, a nawet tysięcy. Autorzy opierając się na najlepszych cechach znanych z literatury heurystycznych algorytmów optymalizacji, zaproponowali własny algorytm przeznaczony do rozwiązywania takich problemów.

Słowa kluczowe: optymalizacja parametryczna, heurystyka, metody ewolucyjne, optymalizacja wielowymiarowa

HEURISTIC PARAMETRIC OPTIMIZATION ALGORITHM FOR MULTIDIMESIONAL PROBLEMS SOLVING

Summary. Heuristic optimization algorithms are known from the beginnings of computer science but ones based on observations of nature phenomenons (evolution, food searching of multiagent colonies, annealing) were introduced relatively late. Each of them have different characteristics of search space exploration. One of known problems of parametric optimization is multidimensional case (hundreds or thousands of dimensions). Authors, inspired by best features of known optimization algorithms, proposed optimization method for such problems solving.

Keywords: parametric optimization, heuristics, evolutionary methods, multidimensional optimization

1. Wstęp

Algorytmy przeszukiwania losowego, stosowane do zadań optymalizacji, to takie, które w sposób iteracyjny przeszukują przestrzeń rozważań w przypadkowych kierunkach, aby znaleźć minimum funkcji celu. Każdy taki algorytm rozpoczyna swoje działanie z punktu początkowego (wylosowanego bądź ustalonego). Kierunek poszukiwania jest realizacją wektora, którego składowymi są zmienne losowe. Krok, który w wyniku daje mniejszą wartość funkcji jakości, jest krokiem pomyślnym, inne kroki są niepomyślne. W najprostszych algorytmach przeszukiwania losowego kroki niepomyślne są odrzucane, zaś pomyślne stają się punktem startowym dla kolejnej iteracji algorytmu. Jeżeli w algorytmie wzdłuż obiecującego kierunku wykonuje się serię kroków, algorytm taki nazywany jest algorytmem z pamięcią. Algorytmy, w których w każdym kroku przyjmowany jest nowy kierunek poszukiwań, nazywane są algorytmami bez pamięci. Należy zauważyć, że w algorytmach z pamięcią rozkład prawdopodobieństwa wartości składowych wektora zależy od numeru kroku. Jeśli dodatkowo rozkład ten jest modyfikowany w trakcie procesu przeszukiwania w celu przyspieszenia zbieżności algorytmu, mówi się o algorytmach przeszukiwania losowego z adaptacją [4]. Wśród podstawowych algorytmów przeszukiwania losowego bez pamięci należy wymienić algorytm z parą kroków próbnych, gdzie po wylosowaniu kierunku przeszukiwań realizowany jest krok w kierunku wylosowanym bądź przeciwnym – tam, gdzie funkcja jakości osiąga mniejszą wartość. Innym przykładem algorytmu bez pamięci może być algorytm najlepszej próby, gdzie na początku iteracji realizowanych jest wiele kroków próbnych, a jako krok roboczy przyjmuje się ten, który prowadzi do najmniejszej wartości funkcji jakości. Dla wybranego kierunku poszukiwań algorytm jest realizowany z wykorzystaniem stałego, przyjętego wcześniej współczynnika. Przykładem algorytmu z pamięcią może być metoda przypadkowego spadku, w której z danego punktu wykonuje się serię kroków w danym kierunku dopóki są to kroki pomyślne. Po wystąpieniu kroku niepomyślnego następuje wybór nowego, bardziej obiecującego kierunku [4].

Opisane wyżej podstawowe metody przeszukiwania funkcji celu są bardzo zawodne w przypadku zadań optymalizacji globalnej, ponieważ dla funkcji o wielu minimach lokalnych znalezione minimum lokalne zależy od punktu startowego algorytmu i nie musi być wcale minimum globalnym. Problem ten rozwiązuje się czasami przez modyfikacje wyżej przedstawionych algorytmów optymalizacji lokalnej, polegające bądź to na wielokrotnym uruchamianiu danego algorytmu z różnymi, losowo wybranymi punktami początkowymi (metody wielostartowe), bądź też na dodaniu do wektora kierunku przeszukiwań składowej prędkości wynikającej z ruchu wykonanego w poprzednim kroku (symulacja ruchu punktu materialnego), mającej zapewnić możliwość opuszczania „płytkich” minimów lokalnych [1].

Inną klasą metod niedeterministycznych są algorytmy cechujące się losowością zarówno w procesie poszukiwań nowych rozwiązań, jak też w procesie przechodzenia między kolejnymi rozwiązaniami roboczymi. Nie operuje się tutaj wektorem kierunku dodawanym do bieżącego rozwiązania, ale przy każdej iteracji generuje się zupełnie nowe rozwiązania kandydujące. Najprostszym z takich algorytmów jest metoda Monte Carlo, gdzie z rozkładem równomiernym dokonuje się wielu losowań punktu z przestrzeni rozwiązań, pamiętając najlepszy, dotychczas znaleziony. Inną metodą może być algorytm błądzenia przypadkowego, gdzie przetwarzany jest jeden punkt roboczy, będący realizacją zmiennej losowej rozkładu prawdopodobieństwa, według którego losuje się następny punkt, który to staje się punktem roboczym bez względu na wartość funkcji jakości. Podobnie jak w metodzie Monte Carlo, tak i tutaj pamiętane jest rozwiązanie najlepsze z dotychczas znalezionych [1]. Oprócz algorytmów, które przetwarzają dane rozwiązanie krok po kroku, istnieją również metody populacyjne, gdzie przetwarza się równolegle (w znaczeniu niezależności rozwiązań) wiele kandydujących rozwiązań. Po ocenie ich wartością funkcji jakości realizuje się grupowanie rozwiązań w celu eliminacji tych, które zbiegają się do jednego minimum lokalnego. Pozwala to na zmniejszenie nakładu obliczeń.

W ostatnich latach zaproponowano algorytmy służące m.in. do optymalizacji globalnej, które swoją jakością znacznie przewyższyły dotychczas stosowane metody. Należy wśród nich wymienić algorytmy symulowanego wyżarzania, algorytmy genetyczne, metody ewolucyjne czy algorytmy mrówkowe. Wykorzystując w swoim działaniu czynnik niedeterministyczny, kwalifikują się one do grupy algorytmów przeszukiwania losowego (najczęściej adaptacyjnych i populacyjnych), jednak ze względu na swoją genezę, specyfikę i podstawy teoretyczne, zostały wydzielone jako osobne grupy, zwane czasem algorytmami metaheurystycznymi.

2. Propozycja algorytmu optymalizacji heurystycznej dedykowanego problemom wielowymiarowym

Pomimo zupełnie różnej genezy algorytmów ewolucyjnych i algorytmów mrówkowych można w nich zauważyć duże podobieństwa, zwłaszcza jeśli bierze się pod uwagę wersje poszczególnych heurystyk stworzone do rozwiązywania podobnych zadań. Każde z podejść wykorzystuje populację osobników reprezentujących rozwiązanie danego zadania i w każdym z podejść wiedza o rozwiązywanym problemie, zgromadzona przez poszczególne osobniki, jest wykorzystywana do stochastycznego wygenerowania nowej populacji osobników. Podstawową różnicą jest fakt, że w przypadku algorytmów ewolucyjnych cała wiedza o rozwiązaniu problemu zawarta jest w samych osobnikach danej populacji, zaś w

przypadku algorytmów mrówkowych wiedza ta zawarta jest w śladzie feromonowym, na tworzenie którego ma wpływ każdy z osobników. Mechanizmy te służą jednak jednemu celowi, jakim jest konstrukcja stale zmieniającej się w kolejnych iteracjach algorytmu wielowymiarowej funkcji rozkładu gęstości prawdopodobieństwa (w przypadku optymalizacji kombinatorycznej funkcja ta jest określona na wartościach dyskretnych), na podstawie której generowane są kolejne osobniki (o lepszej wartości wskaźnika jakości). Algorytmy mrówkowe znajdują szczególne zastosowanie w rozwiązywaniu problemów kombinatorycznych, gdzie liczba wszystkich dopuszczalnych rozwiązań jest ograniczona i możliwe jest przedstawienie tych rozwiązań za pomocą grafu. Taka specyfika umożliwia zakodowanie zadania w postaci danej implementacji grafu. Cała wiedza o rozwiązywanym problemie wynika z wag poszczególnych krawędzi grafu, których wartości mogą być na bieżąco zmieniane przez kolejne osobniki (lepsza wartość funkcji przystosowania to większa ilość pozostawionego feromonu). Na postać funkcji gęstości prawdopodobieństwa, reprezentowanej przez ślad feromonowy, na podstawie której generowane będą następne rozwiązania, mają więc wpływ wszystkie osobniki z wagą wyznaczaną przez zastosowaną funkcję jakości oraz funkcję modelującą parowanie feromonu (starsze rozwiązania są mniej prawdopodobne).

Podobna koncepcja generowania nowych osobników przyświeca ewolucyjnemu systemowi PBIL (ang. *Population Based Incremental Learning*), który został zaproponowany przez S. Baluja w 1994 r. [3]. W koncepcji tej wiedza o rozwiązywanym problemie zawarta jest w tzw. bazowym wektorze prawdopodobieństwa, który stanowi jedyną podstawę do generowania nowych osobników, a który uaktualniany jest na podstawie oceny jakości rozwiązań reprezentowanych przez generowane kolejno osobniki. Prowadzone badania wykazały [3], że algorytm ten daje obiecujące wyniki w zastosowaniu do optymalizacji kombinatorycznej. Natomiast w przypadku optymalizacji parametrycznej uzyskiwane wyniki są gorsze niż w przypadku stosowania klasycznych algorytmów ewolucyjnych. Spowodowane jest to zapewne faktem, że w przypadku optymalizowania funkcji wielomodalnych nie ma możliwości zakodowania w wektorze bazowym funkcji gęstości rozkładu prawdopodobieństwa o bardziej złożonym kształcie, która umożliwiałaby eksplorację rozwiązań wszystkich obiecujących obszarów optymalizowanej funkcji.

Propozycja algorytmu optymalizacji parametrycznej polega na ewolucji osobników będących kodowanymi zmiennoprzecinkowo wektorami punktów przestrzeni, na której określona jest funkcja $f(x): \mathbb{R}^n \rightarrow \mathbb{R}$, dla której poszukujemy ekstremum. W każdej iteracji algorytmu nowy osobnik generowany jest na podstawie stałej liczby poprzedników. Prawdopodobieństwo selekcji danego osobnika wynika z wartości obliczonej dla niego funkcji przystosowania oraz czasu, kiedy został wygenerowany, przy czym wpływ czasu, podobnie jak w algorytmach mrówkowych odparowywanie śladu feromonowego, może mieć charakter

liniowy, wykładniczy bądź inny. Badania własne wykazały, że w przypadku złożonych funkcji wielowymiarowych operatory krzyżowania (zarówno, jeśli chodzi o krzyżowanie pozycyjne, jak i krzyżowanie arytmetyczne) nie przyczyniają się do wzrostu skuteczności metody optymalizacji, często stanowiąc wręcz zawadę (we wszystkich testowanych funkcjach nie uzyskano poprawy, a w niektórych nastąpiło wręcz pogorszenie), dlatego nowy osobnik tworzony jest jedynie w oparciu o mutację, która jest realizowana zgodnie ze schematem mutacji nierównomiernej [2]. Na potrzeby zobrazowania sposobu funkcjonowania zaproponowanego algorytmu zastosowano następujące oznaczenia:

$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ – wektor, który jest punktem przeszukiwanej dziedziny $\Re^n$ funkcji *f,* zwany również osobnikiem,

$\underline{x}_m = \begin{bmatrix} x_{m1} \\ x_{m2} \\ \vdots \\ x_{mn} \end{bmatrix}$ – osobnik wygenerowany w *m*-tej iteracji,

gdzie: *k* jest liczbą osobników biorących udział w każdej iteracji algorytmu optymalizacji, *M* założoną liczbą wszystkich iteracji algorytmu optymalizacji, Q_m wartością wskaźnika jakości dla $\underline{x}_m$, czyli wartością $f(\underline{x}_m)$, a $T(m)$ funkcją określającą wpływ numeru iteracji, w której został wygenerowany dany osobnik na jego prawdopodobieństwo selekcji (wybór wcześniej wygenerowanych osobników jest mniej prawdopodobny).

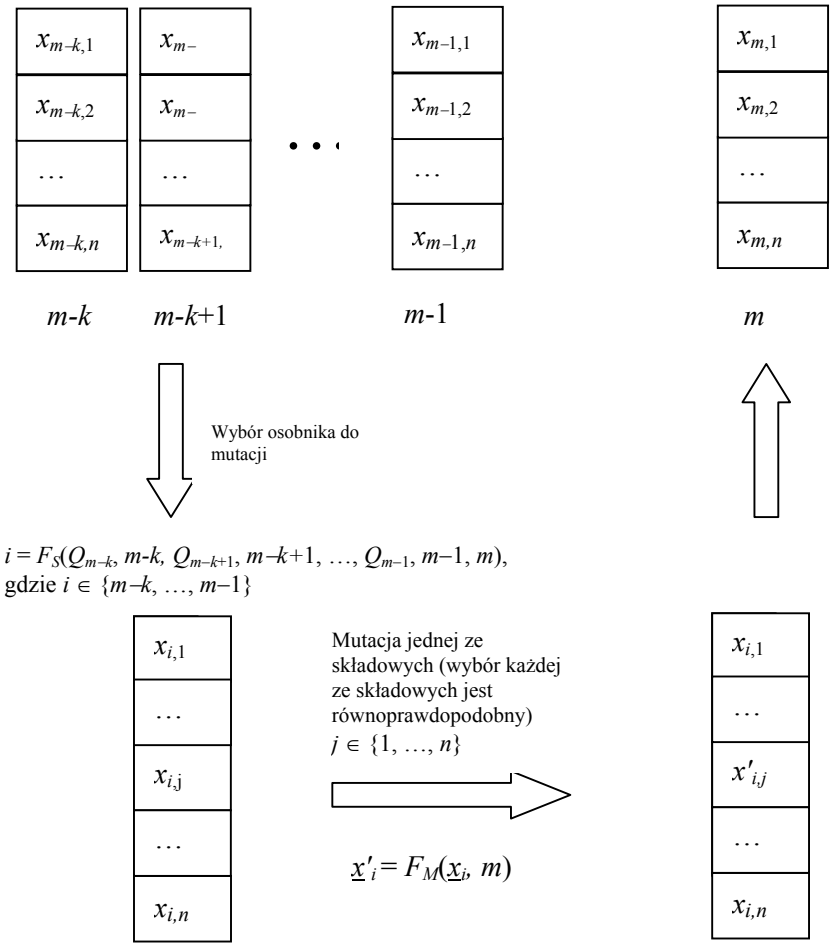
Podstawowe zasady funkcjonowania opisanego algorytmu obrazuje rys. 1 (założono, że algorytm realizuje właśnie *m*-ty krok iteracji).

Selekcji osobnika z populacji bazowej dokonuje się na podstawie wartości funkcji przystosowania Q_m oraz wartości funkcji opisującej wpływ numeru iteracji wygenerowania danego osobnika $T(m)$. Proponuje się, aby prawdopodobieństwo wyboru *m*-tego osobnika opisać następującą zależnością:

$$P_m = \frac{Q_m T(m)}{\sum_{j=1}^{k} Q_j T(j)}, \tag{1}$$

gdzie Q_m jest wskaźnikiem jakości (funkcją przystosowania) *m*-tego osobnika, uprawdopodobniającym wybór osobnika o większej wartości *Q*, zaś *T* funkcją określającą wpływ numeru iteracji, w której został wygenerowany dany osobnik, na jego zdolność do reprodukcji (podobnie jak w algorytmach mrówkowych może to być rosnąca funkcja liniowa, funkcja wykładnicza bądź inna funkcja niemalejąca), uprawdopodobniającą wybór „młodszego" osobnika.

W ramach badań realizowanych w niniejszej pracy wykorzystano funkcję liniową.



Rys. 1. Zobrazowanie funkcjonowania algorytmu optymalizacji parametrycznej
Fig. 1. Visualization of parametric optimization algorithm

Podobnie jak w algorytmach symulowanego wyżarzania, tak i tutaj wprowadzić można czynnik akceptacji (a w przypadku proponowanego algorytmu – selekcji) rozwiązania gorszego z prawdopodobieństwem malejącym wraz z postępem procesu optymalizacji. Prawdopodobieństwo wyboru m-tego osobnika może być przedstawione jako:

$$P_m(m) = \frac{F_A(Q_m, m)T(m)}{\sum_{j=1}^{k} F_A(Q_j, j)T(j)}, \tag{2}$$

gdzie: m jest numerem iteracji algorytmu, a F_A funkcją uprawdopodobniającą rozwiązania o lepszej jakości w ramach postępu procesu optymalizacji np.:

$$F_A(q,m) = a(q+b)^{\left(\frac{A-1}{M}m+1\right)}, \tag{3}$$

gdzie: M jest założoną liczbą iteracji algorytmu, zaś A ($A > 1$) współczynnikiem, z którego wzrostem będzie malało prawdopodobieństwo akceptacji gorszych rozwiązań wraz z postę-

pem procesu optymalizacji. *a* i *b* to współczynniki liniowego przekształcenia przestrzeni funkcji jakości, dobierane przy każdej iteracji algorytmu tak, aby jej wartości dla najgorszego i najlepszego rozwiązania ze zbioru roboczego wynosiły odpowiednio 1 i *MAX* (gdzie $MAX > 1$). Dla zagadnień minimalizacji współczynnik *a* będzie przyjmować wartości ujemne. W ramach badań realizowanych w niniejszej rozprawie ustalono wartości współczynników $A = 2$ oraz $MAX = 10$.

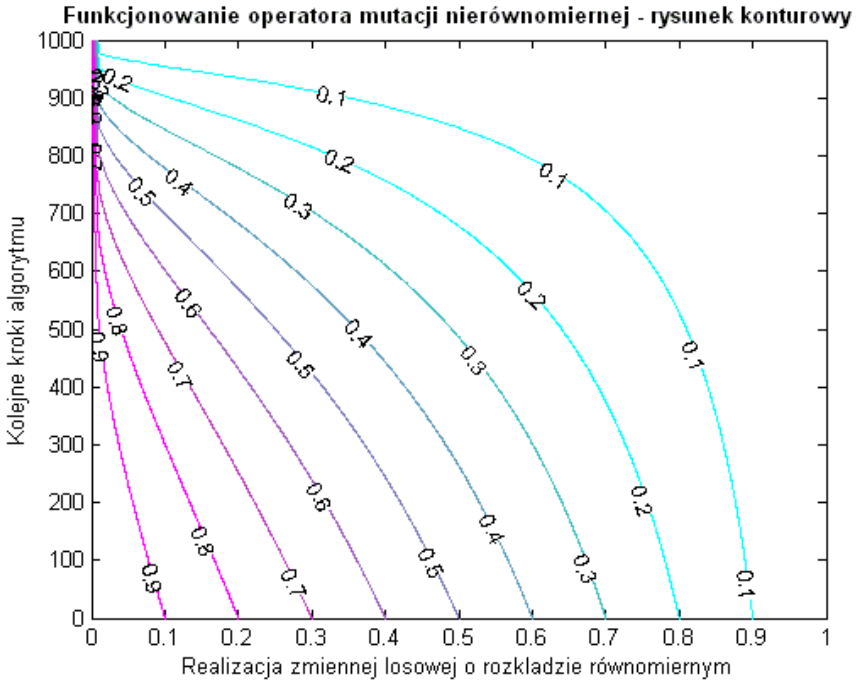
Mutacja realizowana jest w ten sposób, że zmianie podlega każdorazowo tylko jedna składowa wektora rozwiązania, wybrana losowo zgodnie z równomiernym rozkładem prawdopodobieństwa. Zmiana polega na zastąpieniu danej wartości inną, wygenerowaną losowo zgodnie z poniższą formułą [2]:

$$x'_{i,j} = \begin{cases} x_{i,j} + \Delta(m, u_j - x_{i,j}), & \text{prawdopodobieństwem } 1/2 \\ x_{i,j} - \Delta(m, x_{i,j} - l_j), & \text{prawdopodobieństwem } 1/2 \end{cases} \tag{4}$$

gdzie: u_j jest górnym krańcem przedziału zmienności *j*-tej składowej, zaś l_j jest jej dolnym krańcem przedziału zmienności. Funkcja $\Delta(m, y)$ przyjmuje wartości w zakresie $[0,y]$, a prawdopodobieństwo, że $\Delta(m, y)$ jest bliskie 0, wzrasta ze wzrostem *m*. Właściwość ta zapewnia, że w początkowej fazie optymalizacji algorytm eksploruje przestrzeń rozwiązań w szerokim zakresie, by w swojej końcowej fazie eksploatować „wytypowane" wcześniej obiecujące obszary. Zastosowano następującą funkcję $\Delta(m, y)$ [2]:

$$\Delta(m, y) = y(1 - r^{(1-m/M)b}), \tag{5}$$

gdzie: *r* jest liczbą z przedziału [0, 1], losowaną zgodnie z rozkładem równomiernym, każdorazowo przy obliczaniu wartości funkcji Δ, *M* założoną liczbą iteracji algorytmu, zaś *b* współczynnikiem dobranym eksperymentalnie (decydującym o szybkości wymuszania zbieżności) w zależności od „niejednorodności kształtu" optymalizowanej funkcji. Na rys. 2 zobrazowano odwzorowanie wartości zmiennej losowej na wartość modyfikacji danej składowej, realizowane przez omawiany operator mutacji w zależności od kroku algorytmu. Liczby opisane na konturach to wartości funkcji $\Delta(m,1)$.



Rys. 2. Wizualizacja funkcjonowania operatora mutacji nierównomiernej ($M = 1000$, $b = 1$)
Fig. 2. Visualization of nonuniform mutation operator ($M = 1000$, $b = 1$)

Poniżej przedstawiono uproszczony, formalny opis zaproponowanego algorytmu optymalizacji w pseudokodzie. Przez $\mathbf{P}^t$ oznaczono uporządkowany zbiór osobników przetwarzanych w iteracji numer t. Na zbiorze tym operuje funkcja `selekcja`, która wybiera osobnika do mutacji na podstawie formuły (2) oraz (3). Funkcja `mutacja` realizuje operację mutacji na pojedynczej, losowo wybranej składowej, zgodnie ze schematem mutacji nierównomiernej (4) oraz (5). Przez $\mathbf{O}$ oraz $\mathbf{T}$ oznaczono tymczasowe zmienne, pozwalające operować na pojedynczym osobniku.

```
procedure AlgorytmOptymalizacji
begin
    t := 0
    inicjacja P^t
    while ( t < założona liczba iteracji ) do
    begin
        T := selekcja P^t
        O := mutacja T
        ocena O
        P^(t+1) := P^t ↓ najstarszy osobnik ∪ {O}
        t := t + 1
    end
end
```

Inicjalizacja początkowego zbioru osobników polega na wygenerowaniu losowych wartości dla każdej ze składowych, w granicach określonych przedziałów zmienności, zgodnie z rozkładem równomiernym.

Zastosowanie modelu selekcji elitarnej implementowanego w algorytmach ewolucyjnych w ten sposób, że do następnego pokolenia przechodzi zawsze η najlepszych osobników, realizowane jest tutaj przez określenie prawdopodobieństwa selekcji (a następnie mutacji) któregoś z η najlepszych osobników, wynoszącego $\eta/(k+\eta)$ dla każdego z tych osobników. Realizacja modelu selekcji elitarnej wymaga oczywiście dodatkowej struktury danych, umożliwiającej przechowywanie informacji o η najlepszych, znalezionych do bieżącej iteracji osobnikach.

Podsumowując sposób funkcjonowania zaproponowanego algorytmu, można wyliczyć następujące jego cechy:

- sposób utrzymywania populacji osobniczej podobny jest do strategii ewolucyjnej $(\mu+1)$,
- wpływ numeru iteracji, w której został wygenerowany dany osobnik (a więc jego wieku) na prawdopodobieństwo jego selekcji został zaczerpnięty z algorytmów mrówkowych,
- zmniejszanie prawdopodobieństwa akceptacji rozwiązań gorszych wraz z przebiegiem procesu optymalizacji realizowany jest podobnie jak w algorytmach symulowanego wyżarzania,
- sposób generowania nowego osobnika na podstawie osobnika wybranego w procesie selekcji realizowany jest za pomocą operatora mutacji nierównomiernej.

Badania zaproponowanego algorytmu optymalizacji prowadzono w pierwszej kolejności na wielu popularnych funkcjach testowych, w tym funkcji Michalewicza. Przytoczono tylko niektóre wyniki z całości badań nad tymi funkcjami, gdyż prawdziwą miarą skuteczności metody optymalizacji będzie realizacja badań na rzeczywistym systemie rozmytym. Z drugiej jednak strony weryfikację algorytmu rozpoczęto od funkcji testowych, gdyż narzut potrzebny na obliczenie odpowiedzi systemu rozmytego jest dużo większy niż na obliczenie wartości użytych funkcji testowych.

3. Testy zaproponowanej metody na zestawie funkcji testowych De Jonga

Do testów wybrano popularny zestaw [2] testowych funkcji De Jonga składający się z pięciu funkcji zdefiniowanych w tabeli 1. Testy przeprowadzono na podstawie porównania funkcjonowania zaproponowanego algorytmu z klasycznym algorytmem ewolucyjnym.

Podobnie jak w przypadku funkcji Michalewicza, tak i tutaj maksymalizowano funkcje postaci $-F(x)$. Ponieważ dla pierwszych czterech funkcji liczba wymiarów, na których są one określone, jest parametryzowana, więc testy przeprowadzono na funkcjach określonych na 100 wymiarach, co może pozwolić na lepszą weryfikację zaproponowanego algorytmu w za-

stosowaniu do optymalizacji problemów wielowymiarowych. Ostatnią, piątą funkcję optymalizowano w jej pierwotnej postaci.

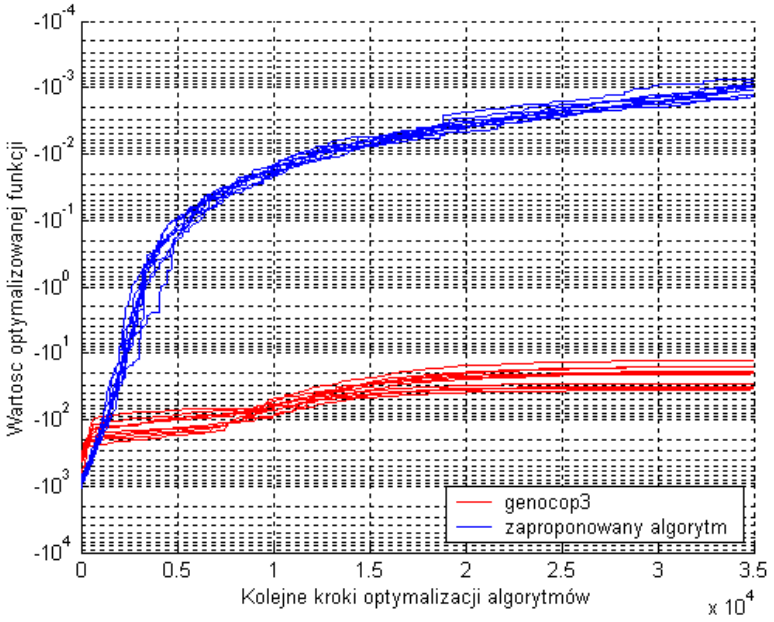
Tabela 1

Definicje funkcji testowych De Jonga

Nazwa	Definicja funkcji
1. Funkcja De Jonga (ang. *Sphere Model*)	$F_1(\underline{x}) = \sum_{i=1}^{Dim} x_i^2$, dla $-5.12 \le x_i \le 5.12;\ i = 1, \ldots, Dim$
2. Funkcja De Jonga (ang. *Rosenbrock's Valley*)	$F_2(\underline{x}) = \sum_{i=1}^{Dim-1} \left(100\left(x_i^2 - x_{i+1}\right)^2 + \left(1 - x_i\right)^2\right)$, dla $-2.048 \le x_i \le 2.048;\ i = 1, \ldots, Dim-1$
3. Funkcja De Jonga (ang. *Step Function*)	$F_3(\underline{x}) = \sum_{i=1}^{Dim} \mathrm{int}\left(x_i\right)$, dla $-5.12 \le x_i \le 5.12;\ i = 1, \ldots, Dim$
4. Funkcja De Jonga (ang. *Quartic Gaussian Function*)	$F_4(\underline{x}) = \sum_{i=1}^{Dim} \left(ix_i^4 + Gauss(0,1)\right)$, dla $-1.28 \le x_i \le 1.28;\ i = 1, \ldots, Dim$
5. Funkcja De Jonga (ang. *Shekel's Foxholes Function*)	$F_5(\underline{x}) = 0.002 + \sum_{j=1}^{25} \frac{1}{f_j}$, $f_j = j + \sum_{i=1}^{2} \left(x_i - a_{i,j}\right)^6$, dla $-65.536 \le x_i \le 65.536;\ i=1,2$ macierz współczynników $a_{i,j}$ przedstawiono poniżej: $a_{ij} = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & -16 & -16 & -16 & 0 & 0 & 0 & 0 & 0 & 16 & 16 & 16 & 16 & 16 & 32 & 32 & 32 & 32 & 32 \end{bmatrix}$

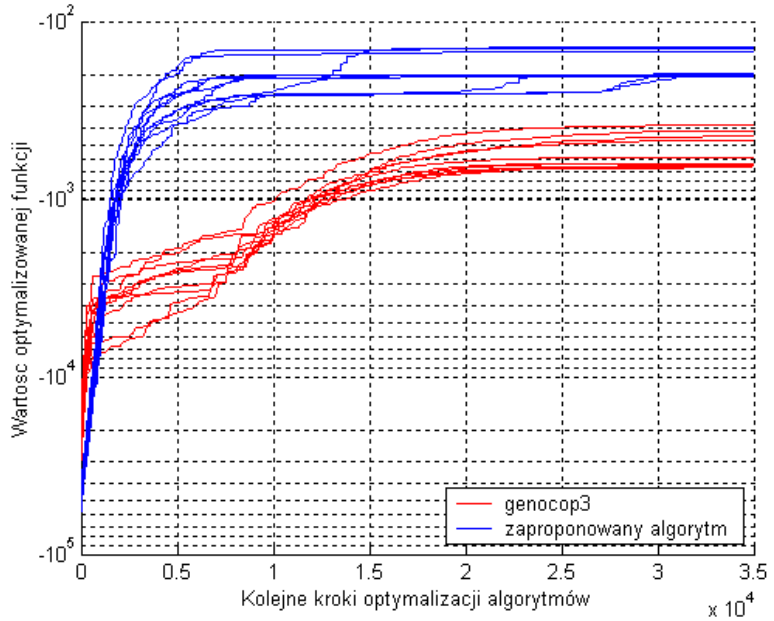
przez *Dim* oznaczono liczbę wymiarów, na których określona jest funkcja.

Wyniki porównań przedstawiono na poniższych rysunkach, gdzie dla każdego z algorytmów wykreślono po 10 krzywych obrazujących przebieg algorytmu optymalizacji, odpowiadających 10 kolejnym uruchomieniom każdego z programów. Dla niektórych funkcji, w celu uzyskania lepszej przejrzystości, wykreślono rysunki z logarytmiczną skalą dla osi odciętych.



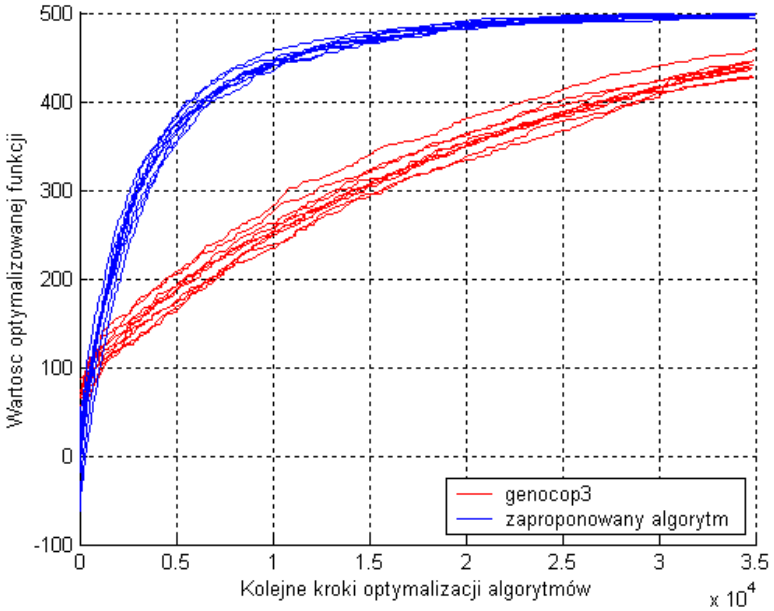
Rys. 3. Porównanie algorytmu ewolucyjnego z zaproponowaną metodą heurystyczną dla pierwszej funkcji De Jonga – skala logarytmiczna

Fig. 3. Comparison of the evolutionary algorithm and the proposed method for the first De Jong's function – logarithmic scale



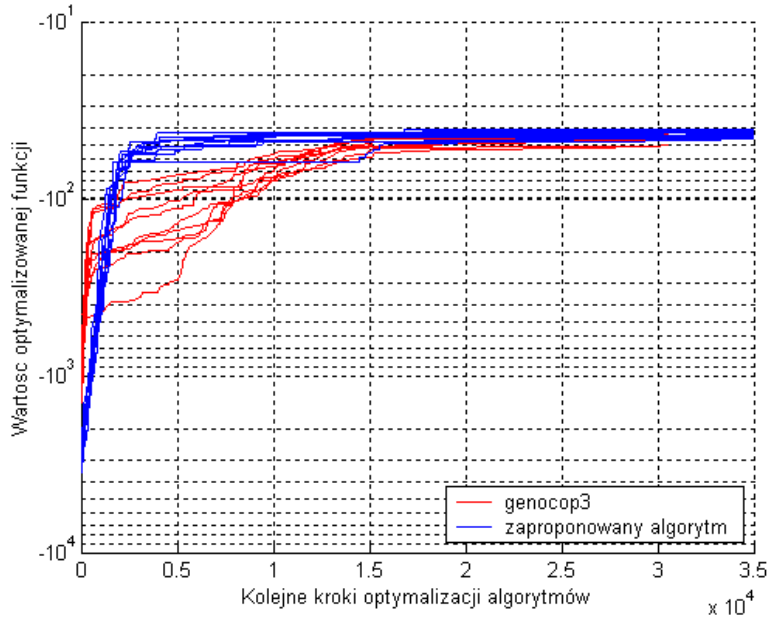
Rys. 4. Porównanie algorytmu ewolucyjnego z zaproponowaną metodą heurystyczną dla drugiej funkcji De Jonga – skala logarytmiczna

Fig. 4. Comparison of the evolutionary algorithm and the proposed method for the second De Jong's function - logarithmic scale



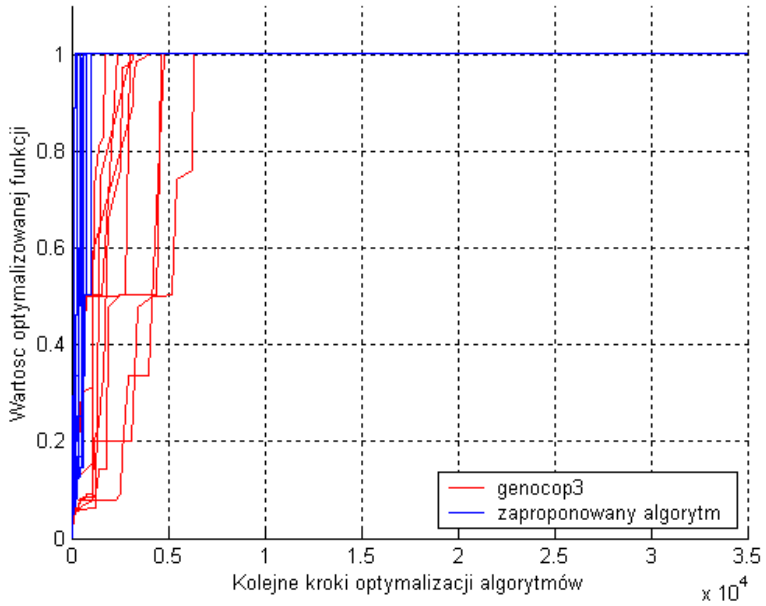
Rys. 5. Porównanie algorytmu ewolucyjnego z zaproponowaną metodą heurystyczną dla trzeciej funkcji De Jonga – skala liniowa

Fig. 5. Comparison of the evolutionary algorithm and the proposed method for the third De Jong's function – linear scale



Rys. 6. Porównanie algorytmu ewolucyjnego z zaproponowaną metodą heurystyczną dla czwartej funkcji De Jonga – skala logarytmiczna

Fig. 6. Comparison of the evolutionary algorithm and the proposed method for the fourth De Jong's function - logarithmic scale



Rys. 7. Porównanie algorytmu ewolucyjnego z zaproponowaną metodą heurystyczną dla piątej funkcji De Jonga – skala liniowa

Fig. 7. Comparison of the evolutionary algorithm and the proposed method for the fifth De Jong's function - linear scale

4. Podsumowanie

Dla wszystkich przetestowanych powyżej funkcji zaproponowana metoda znajduje lepsze przybliżenie wartości optymalnej. Należy również zwrócić uwagę na to, że zaproponowany algorytm cechuje o wiele mniejszy rozrzut zarówno końcowego wyniku, jak i wyników pośrednich, uzyskanych w trakcie przebiegu procesu optymalizacji. Istotną zaletą algorytmu jest również duża szybkość zbieżności, polegająca na wypracowaniu dobrych wyników już we wczesnych iteracjach procesu optymalizacji. Dla drugiej funkcji testowej zaobserwować można pewien rozrzut w znalezionych optimach dla poszczególnych powtórzeń algorytmu, ale uzyskane wyniki są i tak znacznie lepsze od wyników wypracowanych przez system *genocop3*. Rozrzut ten spowodowany jest przez fakt, iż badana funkcja jest niewspółmiernie bardziej „niejednorodna" w porównaniu do pozostałych, a należy przypomnieć, że obydwa algorytmy na potrzeby wszystkich testów miały ustalone i niezmienne parametry funkcjonowania.

Piąta funkcja De Jonga również charakteryzuje się dużym stopniem „niejednorodności", ale ponieważ zdefiniowana jest ona tylko dla dwóch wymiarów, to obydwa algorytmy przy założonej liczbie iteracji uzyskały podobne wyniki, chociaż zaproponowana metoda znacznie przewyższyła szybkością zbieżności.

Skomentować należy również to, iż w początkowej fazie procesu optymalizacji system *genocop3* „wyprzedza" w większości przykładów zaproponowany algorytm. Spowodowane jest to faktem, że system *genocop3* dla zalecanych wartości parametrów rozpoczyna proces optymalizacji z pokoleniem losowo dobranych osobników. Choć zaproponowana metoda przewiduje oczywiście taką możliwość, to przetestowana została wersja z jednym punktem startowym. Miało to na celu zbadanie wpływu losowości punktu startowego na zbieżność wyników dla poszczególnych uruchomień programu. Jak można zobaczyć na wykresach, wpływ ten dla testowanych funkcji nie jest duży, co stanowi niewątpliwą zaletę. Wszystkie z wyżej przeprowadzonych badań zrealizowano z wartościami współczynników zgodnymi z tabelą 2.

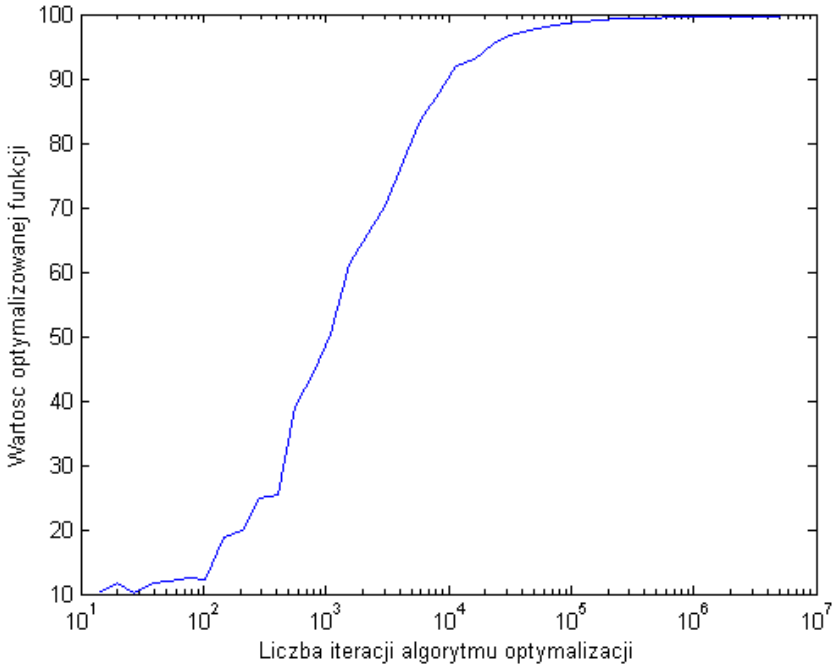
Tabela 2

Wartości współczynników w funkcjach testowych

Współczynnik	Wartość
M (liczba iteracji)	3.5e4
k (liczebność populacji)	15
b (współczynnik niejednorodności)	1

Liczba iteracji została narzucona na potrzeby porównania z systemem Michalewicza (operacją dominującą w przypadku znajdowania ekstremum funkcji jest obliczenie wartości tej funkcji dla zadanych argumentów), zaś pozostałe współczynniki zostały dobrane eksperymentalnie. Podobnie jak w przypadku innych metod heurystycznych, wyliczenie wartości współczynników (zapewniających osiągnięcie najlepszych rezultatów w najkrótszym czasie) sterujących pracą algorytmu na drodze analitycznej nie jest możliwe. Dlatego, aby obrazować ich możliwy wpływ na osiągane wyniki, przeprowadzono poniższe badania z wykorzystaniem funkcji Michalewicza ($k = 2$, $n = 100$).

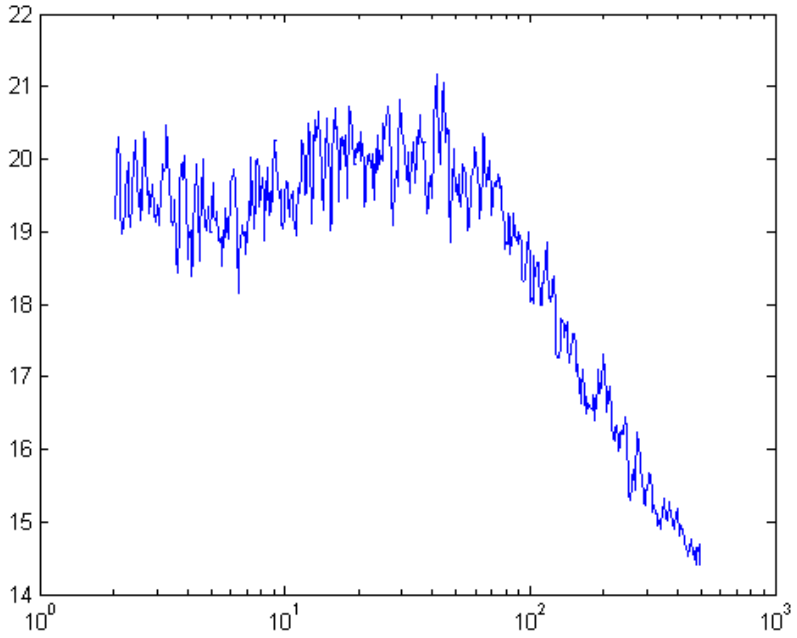
Rysunek 8 obrazuje wpływ liczby iteracji realizowanych przez algorytm w procesie optymalizacji na jakość osiąganych wyników.



Rys. 8. Zobrazowanie wpływu liczby iteracji algorytmu na jakość wyników optymalizacji
Fig. 8. Visualization of number of iterations influence on optimization results quality

Jak widać na rysunku, dla niewielkiej liczby iteracji ($10 - 10^3$) uzyskiwane są słabe wyniki, a ich zmienność jest duża. Dla liczby iteracji większej niż 10^5 uzyskuje się bardzo dobre wyniki, przy czym zwiększanie liczby iteracji ponad 10^6 jest już rozrzutnością w gospodarowaniu mocą obliczeniową, gdyż nie uzyskuje się już dalszej poprawy jakości wyników. W przypadku funkcji Michalewicza z postaci analitycznej szacować można jej górne ograniczenie, co ułatwia podjęcie decyzji o ograniczeniu liczby iteracji do rzędu 10^6, niestety w przypadku przeszukiwania bardziej skomplikowanych funkcji często nie jest to możliwe. Należy zauważyć, że z praktycznego punktu widzenia liczba iteracji algorytmu jest parametrem, od którego zależy czas realizacji procesu optymalizacji, dlatego też, jeśli jest to możliwe, powinno się tak dobierać wartości pozostałych parametrów, aby możliwe było uzyskanie zadowalających wyników optymalizacji dla jak najmniejszej wymaganej liczby iteracji algorytmu.

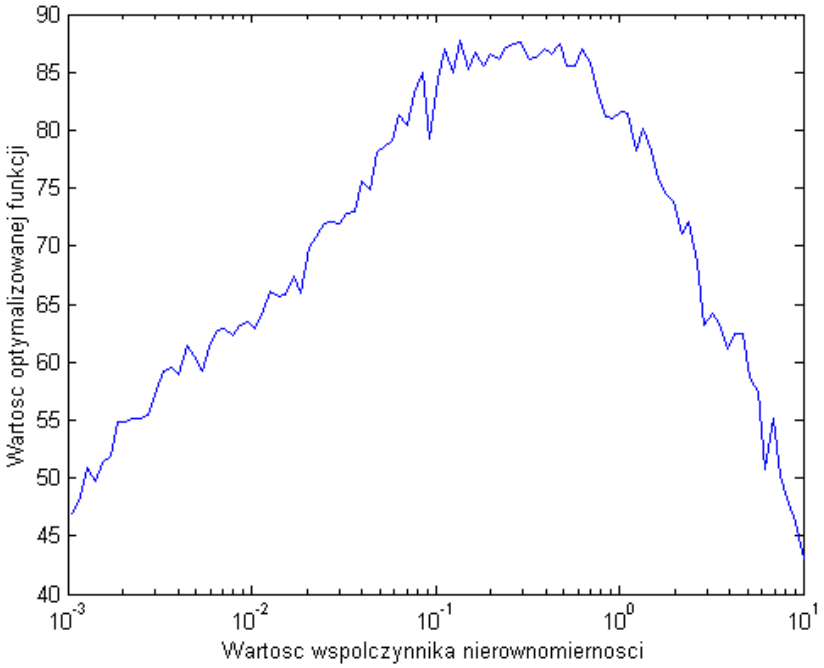
Kolejnym z badanych parametrów zaproponowanego algorytmu optymalizacji jest liczebność roboczego zbioru osobników (liczebność populacji). Przykładowy wpływ zmian wartości tego parametru na jakość osiąganych wyników przedstawiono na rys. 9.



Rys. 9. Zobrazowanie wpływu liczebności roboczego zbioru osobników na jakość wyników optymalizacji
Fig. 9. Visualization of number of agents in working set influence on optimization results quality

Należy zauważyć, że w przypadku poszukiwania minimum funkcji Michalewicza dobre wyniki osiąga się dla wartości tego parametru do 10^2, a dalsze zwiększanie jego wartości przynosi zauważalne pogorszenie osiąganych wyników. Najlepsze rezultaty w badanym przypadku osiągnięto dla liczebności roboczego zbioru osobników ~25, ale mniejsze wartości tego parametru nie wpływają w zdecydowany sposób na znaczne pogorszenie osiąganych wyników. Należy przypuszczać, że w przypadku przeszukiwania bardziej skomplikowanych funkcji, zbyt mała wartość tego parametru może powodować „osiadanie" procesu optymalizacji w lokalnych ekstremach, zaś zbyt duża jego wartość może powodować ekstensywne eksplorowanie całej przestrzeni, na której określona jest przeszukiwana funkcja, zamiast intensywniejszej eksploatacji najbardziej obiecujących obszarów.

Ostatnim z najważniejszych parametrów zaproponowanego algorytmu optymalizacji jest współczynnik niejednorodności przeszukiwanej funkcji. Wpływ zmian wartości tego współczynnika na jakość osiąganych wyników w przypadku przeszukiwania funkcji Michalewicza przedstawiono na rys. 10.



Rys. 10. Zobrazowanie wpływu współczynnika niejednorodności przeszukiwanej funkcji na jakość wyników optymalizacji

Fig. 10. Visualization of nonuniform mutation factor influence on optimization results quality

Współczynnik niejednorodności przeszukiwanej funkcji ma bezpośredni wpływ na funkcjonowanie operacji mutacji nierównomiernej. Mutacja ta zapewniać ma generowanie nowych osobników w taki sposób, że w początkowej fazie mają być tworzone osobniki „odległe" od osobników „rodzicielskich", zaś w końcowej fazie „odległość" ta ma stopniowo maleć. Zastosowanie tego mechanizmu umożliwia znalezienie dokładniejszych wartości dla wyszukanego we wcześniejszej fazie „obiecującego" ekstremum. Należy zauważyć, że zbyt małe wartości tego współczynnika uniemożliwiają skuteczny przegląd całej przeszukiwanej przestrzeni we wczesnej fazie procesu optymalizacji, zaś zbyt duże wartości uniemożliwią „doprecyzowanie" znalezionego rozwiązania w końcowej fazie. Dla stuwymiarowej wersji funkcji Michalewicza i liczbie iteracji ustalonej na 5 tys. wartości prowadzące do dobrych wyników zawierają się w granicach od 0.1 do 1. Istotnym spostrzeżeniem jest ścisły związek tego parametru z doborem liczby iteracji algorytmu. W trakcie realizacji badań zaobserwowano (zgodnie z intuicyjnymi przewidywaniami), że w przypadku dużego rozrzutu wyników dla kolejnych uruchomień programu realizującego proces optymalizacji (przy stałych wartościach współczynników), rozrzut ulegał zmniejszeniu i jakość wyników ulegała poprawie w przypadku zwiększenia liczby iteracji algorytmu bądź też zwiększenia wartości współczynnika niejednorodności (chociaż w tym ostatnim przypadku wyniki były zauważalnie gorsze). Ponieważ zwiększanie liczby iteracji jest możliwe tylko do punktu wyznaczonego przez moc środowiska obliczeniowego, więc w przypadku ustalania wartości

tych dwóch parametrów dla procesu przeszukiwania funkcji, której ekstremum nie da się oszacować, powinno się ustalić maksymalną możliwą liczbę iteracji algorytmu optymalizacji, a współczynnik niejednorodności przeszukiwanej funkcji dobrać eksperymentalnie.

LITERATURA

1. Arabas J.: Wykłady z algorytmów ewolucyjnych. WNT, Warszawa 2001.
2. Michalewicz Z.: Algorytmy genetyczne + struktury danych = programy ewolucyjne. WNT, Warszawa 2003.
3. Baluja S.: Population-based incremental learning: A metod for integrating genetics serach based function optimization and competitive learning. Technical Report CMU-CS-94-163. Carnegie Mellon University. 1994.
4. Seidler J.:, Badach A., Molisz W.: Metody rozwiązywania zadań optymalizacji. WNT, Warszawa 1980.

Recenzent: Dr inż. Arkadiusz Sochan

Wpłynęło do Redakcji 20 listopada 2006 r.

Abstract

Heuristic optimization algorithms are known from the beginnings of computer science but ones based on observations of nature phenomenons (evolution, food searching of multiagent colonies, annealing) were introduced relatively late. Each of them have different characteristics of search space exploration. One of known problems of parametric optimization is the multidimensional case (hundreds or thousands of dimensions). Authors proposed optimization method for such problems solving being inspired by features of well known best optimization algorithms:

- the population of agents is managed similarly to the evolutionary strategy $(\mu + 1)$,
- the age of agent influence on the probability of selection is similar to used in ant systems,
- the decrease of acceptance probability of poor agents in latter iterations is based on the simulated annealing algorithm.

Proposed method was tested on popular benchmark functions suite (De Jong's functions). The results achieved by the proposed method are comparable and even better then ones achieved by the classical evolutionary system especially in a multidimensional case.

Adresy

Dariusz Rafał AUGUSTYN: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, draugustyn@polsl.pl .
Łukasz WYCIŚLIK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, lukasz.wycislik@polsl.pl .