Fundusze Europejskie dla Rozwoju Społecznego | Rzeczpospolita Polska | Dofinansowane przez Unię Europejską

Politechnika Świętokrzyska
Wydział Elektrotechniki, Automatyki i Informatyki

Kierunek studiów:
Elektrotechnika

Agata Kaźmierczyk

Materiały dydaktyczne do przedmiotu

Metody optymalizacji w elektroenergetyce przemysłowej

opracowane w ramach realizacji Projektu
„**Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki**”
FERS.01.05-IP.08-0234/23

Kielce, 2025

Spis treści

1. Wprowadzenie do algorytmów heurystycznych

Algorytmy heurystyczne stanowią grupę metod rozwiązywania problemów optymalizacyjnych, które nie gwarantują znalezienia rozwiązania optymalnego, ale pozwalają na szybkie uzyskanie rozwiązań dobrych lub zadowalających. Są szczególnie przydatne w sytuacjach, gdy klasyczne metody numeryczne lub dokładne są niewykonalne z powodu złożoności problemu lub wymaganego czasu obliczeń.
Heurystyki bazują na regułach, które często wynikają z intuicji, doświadczenia lub analizy problemu, i umożliwiają efektywne przeszukiwanie dużych przestrzeni rozwiązań. Dzięki temu stosowane są w wielu dziedzinach inżynierii, informatyki oraz nauk stosowanych, na przykład w optymalizacji tras, planowaniu produkcji, a także w elektroenergetyce.
Zaletą algorytmów heurystycznych jest możliwość szybkiego znalezienia rozwiązań przybliżonych, co jest często wystarczające w praktyce. Wadą jest brak gwarancji, że wynik będzie najlepiej możliwy (optymalny) dla danego problemu.

1.1. Charakterystyka algorytmów heurystycznych

Algorytmy heurystyczne charakteryzują się kilkoma właściwościami, które decydują o ich użyteczności i ograniczeniach:

- Efektywność obliczeniowa: Heurystyki zwykle działają szybciej niż metody dokładne, ponieważ unikają wyczerpującego przeszukiwania całej przestrzeni rozwiązań.
- Rozwiązania przybliżone: Zapewniają odpowiednio dobre, ale niekoniecznie optymalne rozwiązania. W praktyce takie rozwiązania często są wystarczające, zwłaszcza w skomplikowanych problemach.
- Brak gwarancji optymalności: Nie istnieje oferta formalna, że algorytm zawsze odnajdzie globalne optimum. Ponadto, różne uruchomienia mogą zwracać różne wyniki.
- Oparty na intuicji i heurystykach: Algorytmy wykorzystują reguły decyzyjne lub przeszukiwania, które są zaprojektowane na podstawie praktycznego rozumienia problemu.
- Elastyczność: Można je dostosować i modyfikować do bardzo różnorodnych problemów czyniąc je narzędziem uniwersalnym.

1.2. Najpopularniejsze metody heurystyczne

Algorytm zachłanny

Politechnika Świętokrzyska Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki” nr FERS.01.05-IP.08-0234/23

To najprostsza heurystyka, która w każdej iteracji wybiera lokalnie najlepszą opcję, mając nadzieję, że ciąg tych decyzji doprowadzi do rozwiązania bliskiego optymalnemu. Na przykład, przy podejmowaniu decyzji dotyczącej wyłączenia łącznika minimalizującego straty, algorytm wybiera taki, który w tym kroku da największą korzyść.
Choć algorytm zachłanny jest szybki i prosty, często zatrzymuje się w lokalnym optimum, które może być bardzo dalekie od optymalnego na globalnej skali.

Symulowane wyżarzanie (Simulated Annealing)

Inspiracją jest proces wyżarzania w metalurgii, gdzie kontrolowane schładzanie metalu powoduje, że struktura osiąga stan o minimalnej energii.
Algorytm zaczyna od losowego rozwiązania i wprowadza losowe zmiany (mutacje). Zmiany poprawiające wynik są zawsze akceptowane, a zmiany pogarszające — z pewnym malejącym z czasem prawdopodobieństwem. Dzięki temu algorytm może uciec z lokalnych minimów, zwiększając szanse znalezienia rozwiązania bliskiego optymalnemu.
Parametr temperatury kontroluje stopniowo obniżane prawdopodobieństwo akceptacji rozwiązań gorszych.

Algorytmy genetyczne

Algorytmy genetyczne wykorzystują mechanizmy inspirowane biologicznym procesem ewolucji. Tworzą populację możliwych rozwiązań, które są poddawane krzyżowaniu i mutacji, a następnie selekcji najlepszych osobników.
W kolejnych pokoleniach populacja zbliża się do lepszych rozwiązań, dzięki czemu możliwe jest efektywne przeszukanie przestrzeni rozwiązań. Algorytmy genetyczne dobrze nadają się do skomplikowanych problemów z wieloma lokalnymi minimami.

Przeszukiwanie z tabu (Tabu Search)

Ta metoda rozszerza klasyczne przeszukiwanie lokalne poprzez utrzymywanie listy "tabu" — zestawu niedozwolonych ruchów, co zapobiega powrotom do już odwiedzonych rozwiązań i wpadaniu w pętle.
Dzięki temu metoda może efektywniej eksplorować przestrzeń rozwiązań oraz znajdować lepsze optima lokalne.

1.3. Zastosowania algorytmów heurystycznych

Algorytmy heurystyczne mają zastosowanie w rozwiązywaniu problemów praktycznych, gdzie dokładność ustępuje szybkości i możliwościom poradzenia sobie z problemem złożonym:

- Optymalizacja tras i harmonogramów: zadania takie jak problem komiwojażera, planowanie rozdziału zadań czy optymalizacja tras pojazdów.
- Sterowanie systemami elektroenergetycznymi: optymalizacja pracy łączników, minimalizacja strat sieciowych, lokalizacja rozproszonych źródeł energii.
- Planowanie produkcji i nieliniowe problemy inżynierskie: np. dobór parametrów urządzeń, układów elektronicznych, optymalizacja kosztów.
- Optymalizacja wielokryterialna: pozwalająca na uwzględnienie wielu sprzecznych celów jednocześnie.

1.4. Wskazówki praktyczne przy stosowaniu heurystyk

- Dostosowanie do problemu: algorytm powinien być modyfikowany i dobierany do charakterystyki konkretnego problemu, aby osiągnąć dobre wyniki.
- Testowanie i walidacja: ważne jest sprawdzanie zachowania algorytmu na różnych zestawach danych i wariantach problemu, by ocenić jego stabilność i jakość rozwiązań.
- Hybrydyzacja metod: często skuteczne są połączenia heurystyk z metodami dokładnymi lub innymi heurystykami, co pozwala łączyć zalety różnych podejść.
- Parametryzacja: odpowiedni dobór parametrów (np. wielkość populacji, temperatura w symulowanym wyżarzaniu) może znacząco wpłynąć na efektywność algorytmu.
- Balans pomiędzy jakością a czasem: wybór metody i jej parametrów powinien uwzględniać wymagania dotyczące czasu działania i jakości rozwiązania.

Algorytmy heurystyczne to wszechstronne i potężne narzędzia do rozwiązywania szerokiego zakresu problemów optymalizacyjnych. Choć nie zapewniają gwarancji znalezienia rozwiązania optymalnego, to ich szybkość działania oraz elastyczność czynią je niezbędnymi w praktyce inżynierskiej i naukowej. W elektroenergetyce pomagają skutecznie zarządzać złożonymi systemami, pozwalając na uzyskanie wygodnych kompromisów między efektywnością, kosztami i bezpieczeństwem/

2. Algorytmy heurystyczne – symulowane wyżarzanie oraz algorytm cząstek roju

Oprócz tzw. metod klasycznych, istnieje grupa metod zwanych heurystycznymi, iteracyjnymi metodami przeszukiwania. Metody te znalazły zastosowanie z następujących przyczyn:

- metody te nie wymagają szczególnych założeń dotyczących rozwiązywanego problemu (np. różniczkowalność czy wypukłość optymalizowanej funkcji);

- większość praktycznych zadań jest NP-trudna i klasyczne algorytmy nie nadają się do ich rozwiązania;
- nie przetwarzają bezpośrednio zmiennych decyzyjnych, lecz ich zakodowaną postać;
- są to metody bezgradientowe, nie korzysta się z wartości pochodnej funkcji celu lecz z informacji o wartości funkcji celu;
- funkcja celu w zadaniu jest miarą przystosowania bieżącego rozwiązania do warunków zadania i punkty lepiej przystosowane podlegają dalszemu przetwarzaniu do momentu uzyskania końcowego rozwiązania;
- przetwarzanie kodowanych rozwiązań odbywa się z wykorzystaniem procedur losowych, chociaż cały proces pozostaje procesem deterministycznym;
- podstawowy cel algorytmu polega na poprawianiu bieżącego rozwiązania, a rozwiązanie optymalne jest konsekwencją takiego poprawiania.

Metody heurystyczne pozwalają rozwiązywać różnego rodzaju zadania, również w przypadku gdy zastosowanie metod klasycznych jest zbyt pracochłonne. Metody heurystyczne charakteryzują się tym, że nie wymagają znajomości postaci pochodnej funkcji celu oraz są odporne na nieciągłości funkcji oraz na lokalne minima.

Do metod heurystycznych zalicza się wiele innych metod:

- Symulowane wyżarzanie (ang. simulated annealing);
- Algorytmy świetlikowe (ang. firefly algorithm);
- Algorytm optymalizacji cząstek gazu (ang. *gases Brownian motion optimization algorithm*);
- Inteligencja sztucznego roju (ang. *artificial swarm intelligence*).

Wymienione powyżej algorytmy, jest to klasa metod przeznaczonych do rozwiązywania problemów nietypowych, złożonych, wielowymiarowych, dyskretnych lub nie do końca precyzyjnie określonych. Algorytmy te charakteryzują się cechami takimi jak: losowość, obliczenia na n-wektorach zmiennych jednocześnie, możliwość modyfikacji w trakcie trwania obliczeń. Można je stosować do szybkiego rozpoznania problemu. Niestety nie umożliwiają one zawsze znalezienia optimum globalnego. Umożliwiają jednak określenie obszaru, w którym ono się znajduje.
Algorytm ewolucyjny, służący do rozwiązania wybranego problemu zawiera:

- genetyczną reprezentację możliwych rozwiązań problemu;
- sposób na utworzenie populacji początkowej możliwych rozwiązań;
- funkcję ewaluacji szacującą rozwiązania w kontekście ich przystosowania;
- operatory genetyczne zmieniające skład populacji;
- wartość parametrów pracy algorytmu ewolucyjnego.

Problemem przy stosowaniu algorytmów ewolucyjnych jest ich przedwczesna zbieżność lub też brak zbieżności do rozwiązań optymalnych. Metodą likwidacji tych zjawisk jest tzw. skalowanie przystosowania. Rozróżnia się następujące metody skalowania przystosowania: skalowanie liniowe, skalowanie potęgowe, skalowanie σ – odcięcia, skalowanie logarytmiczne, skalowanie „metodą okna", skalowanie metodą Boltzmanna, skalowanie rankingowe liniowe, skalowanie rankingowe wykładnicze. Bardzo istotnym problemem jest także wybór metody kodowania. Przy wyborze metod kodowania należy uwzględniać zasadę znaczących bloków budujących i zasadę minimalizacji alfabetu.

Algorytm optymalizacji cząstek (PSO, ang. Particle Swarm Optimization) jest jedną z technik obliczeniowych opracowanych na podstawie zachowania stad ptaków i ławic ryb. Zaobserwowano, że osobniki w stadzie mają tendencję do utrzymywania określonych odległości od swoich sąsiadów, dzięki odpowiedniemu dostosowaniu swojej prędkości. Ten sposób poruszania umożliwia im synchroniczny i bezkolizyjny ruch, któremu często towarzyszą nagłe zmiany kierunków i towarzyszące im przegrupowania.

Algorytm roju cząstek rozpoczyna się od utworzenia początkowych cząstek i przypisania im początkowych prędkości. Oceniana jest funkcja celu w każdej lokalizacji cząstki i określana jest najlepszą wartość funkcji i najlepsza lokalizacja. Nowe prędkości określane są w oparciu o aktualną prędkość, indywidualne najlepsze położenie cząstek i najlepsze lokalizacje ich sąsiadów. Następnie iteracyjnie aktualizuje się położenie cząstek, nowa lokalizacja ustalana jest na podstawie poprzednich położenia i prędkość, i jest modyfikowane, aby utrzymać cząstki w dopuszczalnych granicach.

Poniżej zamieszczono przykładowe skrypty z funkcjami do optymalizacji z algorytmem symulowanego wyżarzania.

% skrypt 1

```
%     Minimization of Dejong's fifth function:
%      x0 = [0 0];
%      fun = @dejong5fcn;
%     [x,fval] = simulannealbnd(fun,x0)
% Minimization of Dejong's fifth function subject to lower and upper
bounds:
      x0 = [0 0];
      fun = @dejong5fcn;
      lb = [-64 -64];
```

```
      ub = [64 64];
      [x,fval] = simulannealbnd(fun,x0,lb,ub)
```

```
Optimization terminated: change in best function value less than
options.FunctionTolerance.
x =
  -15.9790  -31.9593
fval =
    1.9920
```

% skrypt 2

```
%FUN can also be an anonymous function:
% fun =  @(x) 3*sin(x(1))+exp(x(2));
% [x,fval] = simulannealbnd(fun,[1;1],[0 0])
% Minimization of Dejong's fifth function while displaying plots:
   x0 = [0 0];
   fun = @dejong5fcn;
   options = optimoptions('simulannealbnd', 'PlotFcn', {@saplotbestx,
@saplotbestf, @saplotx, @saplotf});
   simulannealbnd(fun,x0,[],[],options)
```

Poniżej zamieszczono opis i przykłady skryptów z algorytmem algorytmu cząstek roju.

Poszukuje się takich wartości tych zmiennych, które zapewniają minimalną wartość funkcji celu, tj.

$$F_c(\mathbf{x}) = F_c(x_1^*, x_2^*, \ldots, x_n^*) \rightarrow \min \tag{1}$$

Nawet wobec niezwykle dużej liczby metod, które zapewniają rozwiązanie tego zadania, można stwierdzić, że proces iteracyjny określania optymalnych wartości składowych wektora $[\mathbf{x}]$ jest podobny, albowiem najczęściej w kolejnym kroku iteracyjnym wykonywane jest działanie:

$$x_l^{(i+1)} = x_l^{(i)} + \Delta x_l^{(i)} \tag{2}$$

zaś korekta składowej l - tej wektora, w i - tym kroku iteracyjnym, wyznaczana jest jako wartość funkcji:

$$\Delta x_l^{(i)} = f_l(\xi) \tag{3}$$

W oparciu o powyższą zależność, obliczenia wykonywane są np. w metodach:
– roju cząstek (PSO),

$$x_{k+1}^i = x_k^i + v_{k+1}^i \tag{4}$$

$$v_{k+1}^i = w \cdot v_k^i + c_1 \cdot r_1 \cdot (p^i - x_k^i) + c_2 \cdot r_2 \cdot (p^s - x_k^i) \tag{5}$$

gdzie:

- x_k^i – jest wektorem położenia i-tej cząstki w k-tej iteracji;
- v_k^i – jest wektorem prędkości i-tej cząstki w k-tej iteracji;
- p^i – jest najlepszym, dotychczas znalezionym położeniem i-tej cząstki;
- p^s – jest najlepszym położeniem znalezionym przez lidera roju;
- w – jest współczynnikiem bezwładności ruchu cząstki;
- c_1, c_2 – są to ustalone współczynniki przyspieszenia lub uczenia się;
- r_1, r_2 – są liczbami uzyskiwanymi z generatora liczb losowych o rozkładzie równomiernym w przedziale

```
x = particleswarm(fun,nvars)
x = particleswarm(fun,nvars,lb,ub)
x = particleswarm(fun,nvars,lb,ub,options)
x = particleswarm(problem)
[x,fval,exitflag,output] = particleswarm(___)
```

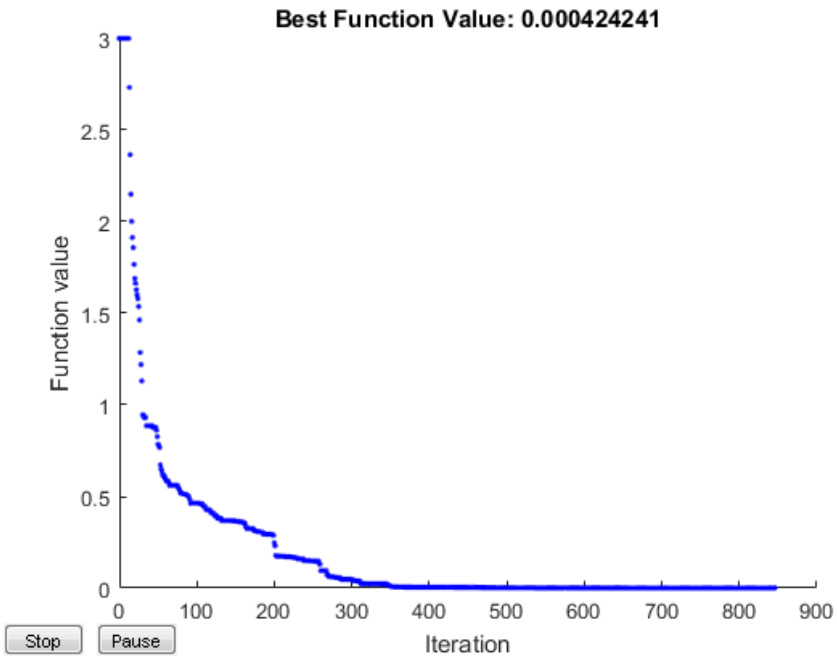
```
Minimize a simple function of two variables.
fun = @(x)x(1)*exp(-norm(x)^2);
rng default
nvars = 2;
x = particleswarm(fun,nvars)
x = 629.4474 311.4814
This solution is far from the true minimum, as you see in a function plot.
fsurf(@(x,y)x.*exp(-(x.^2+y.^2)))
Use a larger population and a hybrid function to try to get a better solution.
Specify the objective function and bounds.
fun = @(x)x(1)*exp(-norm(x)^2); lb = [-10,-15]; ub = [15,20];
```

```
options                                                                    =
optimoptions('particleswarm','SwarmSize',100,'HybridFcn',@fmincon);
rng default % For reproducibility
nvars = 2;
 x = particleswarm(fun,nvars,lb,ub,options)
x = -0.7071 -0.0000

Return the optional output arguments to examine the solution process in more
detail.
fun = @(x)x(1)*exp(-norm(x)^2);
lb = [-10,-15];
ub = [15,20];
options = optimoptions('particleswarm','SwarmSize',50,'HybridFcn',@fmincon);
rng default % For reproducibility
nvars = 2;
[x,fval,exitflag,output] = particleswarm(fun,nvars,lb,ub,options)
x = -0.7071 -0.0000
fval = -0.4289
exitflag = 1
output =
rngstate: [1x1 struct]
iterations: 43
funccount:  2203  message:  'Optimization  ended:  relative  change  in  the
objective valu...'
```

Przykładowy wykres pokazujący proces minimalizacji optymalizowanej funkcji uzyskany z wykorzystaniem funkcji Matlbab particleswarm.

[Tekst alternatywny. Wykres przedstawia przebieg procesu optymalizacji, prezentując wartość funkcji celu w kolejnych iteracjach algorytmu. Oś pozioma obrazuje numer iteracji, natomiast oś pionowa pokazuje wartość funkcji celu, która systematycznie maleje wraz z liczbą iteracji aż do uzyskania najlepszej wartości optymalnej, wynoszącej 0.000424241.]

*Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23*



Rys. 1. Rozwiązanie dla optymalizowanej funkcji testowej

3. Zastosowanie algorytmów heurystycznych w środowisku programu Matlab

3.1. Algorytmy heurystyczne

Algorytmy heurystyczne można określić jako algorytmy bezgradientowe, które czerpią inspiracje m.in. z procesów fizycznych, biologicznych oraz innych obserwacji świata przyrody. Algorytmy heurystyczne to alternatywny sposób wyszukiwania rozwiązań, polegający na regularnym przybliżaniu się do najbardziej optymalnego rozwiązania przy wykorzystaniu różnych heurystycznych reguł. Zaletą stosowania algorytmów opartych o heurystyki jest ich duża umiejętność adaptacyjna do problemów. Znane są m.in. następujące algorytmy optymalizacji heurystycznej takie jak:

- algorytmy genetyczne (GA);
- algorytmy ewolucyjne (EA);
- optymalizacja cząstek roju (PSO);
- symulowane wyżarzanie (SA);
- algorytmy wody deszczowej (RWA);
- algorytmy pszczele (BA);

- algorytmy świetlikowe (FA);
- algorytmy żabich skoków (SFLA);
- algorytmy orki zabójcy (KWA);
- algorytmy mrówkowe (AA).

Algorytmy heurystyczne stosowane są również do rozwiązywania złożonych obliczeniowo optymalizacyjnych problemów w elektroenergetyce. Zastosowania te dotyczą zagadnień projektowania, funkcjonowania oraz sterowania procesami w obiektach energetycznych.
Algorytmy optymalizacyjne znajdują n.in. zastosowania w zakresie optymalizacji projektowania i funkcjonowania sieci elektroenergetycznych, które są podatne na przykład na problemy dotyczące nowych źródeł energii przyłączanych do sieci.
Metody optymalizacji stosowane są m.in. do rozwiązania następujących problemów w elektroenergetyce:

- podłączania nowych źródeł do istniejącego już systemu elektroenergetycznego;
- doboru dla farmy wiatrowej układ kompensacji z dławikiem;
- dostosowania poziomu generowanej mocy do limitów transformatorów;
- ograniczania kosztów zapotrzebowania na moc;
- doboru falowników dla instalacji fotowoltaicznych;
- kontroli przepływu mocy czynnej w systemach elektroenergetycznych;
- optymalizacji w systemie elektroenergetycznym rozpływów mocy biernej.

Jednym z problemów optymalizacyjnych opisywanym w literaturze jest zadanie optymalizacji dotyczące doboru urządzeń kompensacyjnych do układów przyłączenia farm wiatrowych przyłączanych liniami kablowymi do sieci elektroenergetycznej w celu kompensacji mocy biernej. W takiej sytuacji instaluje się m.in. dławiki kompensujące. Innym przykładem zadania optymalizacyjnego jest optymalizacja lokalizacji tzw. „punktów rozcięć" w sieciach średniego napięcia przy wykorzystaniu algorytmów heurystycznych. Ten problem optymalizacyjny dotyczy minimalizacji straty technicznych w sieciach. W celu realizacji obliczeń optymalizacyjnych można wykorzystać program Matlab, który zawiera bibliotekę z wbudowanymi funkcjami opartymi o wybrane algorytm optymalizacyjne.

Do realizacji obliczeń wykorzystano następujące metody:

- funkcja „runopf" pakietu Matpower;
- algorytm cząstek roju (PSO);
- algorytm symulowanego wyżarzania (SA).

3.2. Przykłady obliczeń optymalizacyjnych:

Jednym z algorytmów optymalizacyjnych jest metoda algorytmu cząstek roju (Particle Swarm Optimalization). Metoda cząstek roju opiera swoje działanie o matematyczny opis zachowani roju cząstek, dla algorytmu ważnymi parametrami są m.in.: położenie i prędkość lidera roju. Poniżej zamieszczono przykładowy skrypt w którym wykorzystano algorytm PSO do optymalizacji testowej funkcji (dla analizowanego zadania optymalne rozwiązanie znajduje się blisko punktu x=1000 i y=1000). Skrypt utworzono w programie Matlab. Uzyskany wynik zaprezentowano na utworzonym wykresie na rysunku 1.

Przykład 1. Zastosowanie algorytmów heurystycznych w środowisku Matlab.

Zapoznanie się z zastosowaniem algorytmów heurystycznych przez wykorzystanie programu Matlab, jako idealnego środowiska optymalizacyjnego.

I. Zagadnienia:

- cząstek roju (PSO);
- symulowane wyżarzanie (SA);
- wykorzystanie programu Matlab do rozwiązywania problemów optymalizacyjnych;
- analiza zastosowań algorytmów heurystycznych w elektroenergetyce.

II. Przykłady:

Jednym z algorytmów optymalizacyjnym jest metoda cząstek roju (Particle Swarm Optimalization). Poniższy przykład umożliwia nam zrozumienie jak omawiany sposób może być wykorzystywany do znalezienia optimum.

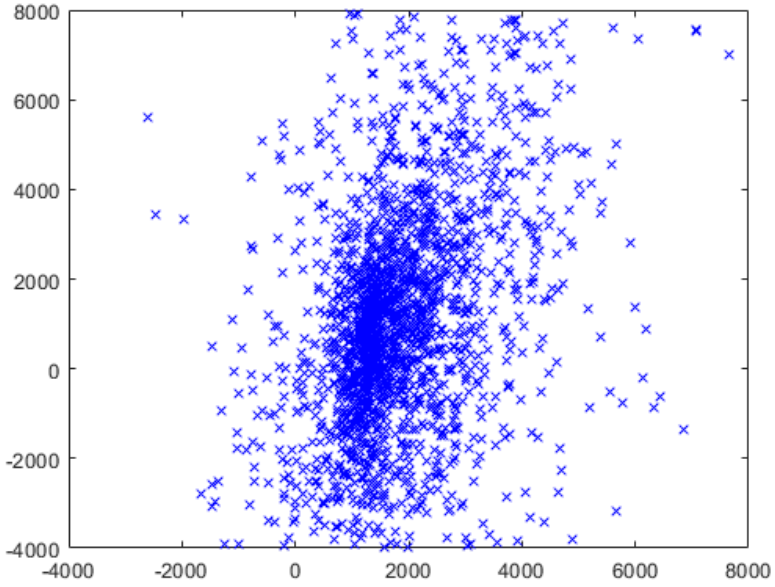
```
clc;
clear;

sw = 2500; % liczba rojów
i = 200; % liczba zastosowanych iteracji

factor = 2; % współczynnik korygujący

s = 1; % krok
psw = zeros (2500 , 8); % ustalenie początkowej pozycji roju

for n = 1 : 2500
```

```
    psw (s, 1 : 5) = n;
    s = s + 1;

end

for iteration = 1 : i

    for n = 1 : sw

        psw (n , 1) = psw (n , 1) + psw (n , 5); % zmiana wartości a
        psw (n , 2) = psw (n , 2) + psw (n , 6); % zmiana wartości b

        a = psw (n , 1);
        b = psw (n , 2);

        fun = (a - 20)^2 + (b - 10)^2; % zastosowanie funkcji celu

        if fun < psw (n , 8)

            psw (n , 4) = psw (n , 1);  % zmiana najlepszej wartości a
            psw (n , 5) = psw (n , 2);  % zmiana najlepszej wartości b
            psw (n , 8) = fun;        % wyznaczenie uzyskanej wartości
minimalnej
         end
    end

    [temp, value] = min (psw (: , 8)); % wyznaczenie najlepszej uzyskanej
wartości
     for n = 1 : sw
         psw (n , 5) = psw (n,5)*rand + factor*(psw (n,4)*rand - psw
(n,1))...
             + factor*(psw (value,3)*rand)+ 2500;

        psw (n , 6) = psw (n,6)*rand + factor*(psw (n,5)*rand - psw
(n,2))...
             + factor*(psw (value,4)*rand)+ 2000;
     end

  plot (psw (: , 1), psw (: , 2) , 'x','Color','b') % uzyskanie wykresu
    axis ([-4000 8000 -4000 8000])
    pause (.1)

end
```

[Tekst alternatywny. Wykres punktowy przedstawia rozrzut danych w dwóch wymiarach, gdzie większość punktów skupia się w centralnej części wykresu,

*Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23*

wskazując na obszar najczęściej osiąganych wartości podczas optymalizacji funkcji testowej.]



Rys. 2. Rozwiązanie dla optymalizowanej funkcji testowej

Poprawne wykorzystanie metody PSO umożliwia uzyskanie wykresu dzięki któremu można zauważyć, że rozwiązanie znajduje się blisko punktu x=1000 i y=1000. Poniżej przedstawiono przykładowy skrypt optymalizacyjny:

```
fun = @dejong5fcn;
x0 = [0 0];
a = [-64 -64];
b = [64 64];
[x , fval] = simulannealbnd (fun , x0 , a , b)
% Wykonanie minimalizacji poprzez użycie piątej funkcji Dejonga
x   = -15.9790   -31.9593
fval = 1.9920
```

Drugi przykład algorytmu optymalizacyjnego:

```
fun = @dejong5fcn;
x0 = [0 0];
w = optimoptions ('simulannealbnd' , 'PlotFcn' , {@saplotbestx ,
@saplotbestf , @saplotx , @saplotf});
simulannealbnd (fun , x0 , [] , [] , w);
```

Po poznaniu algorytmu rojem cząstek, warto zapoznać się symulowanym wyżarzaniem. Algorytmy heurystyczne mogą pomóc również przy problemach nie tylko związanymi z elektroenergetyką. Z ich pomocą jesteśmy w stanie rozwiązać również proste zagadki typu sudoku. Poniższy plik idealnie to pokazuje wykorzystanie symulowanego wyżarzania (Simulated annealing) do rozwiązania prostej łamigłówki.

```
columna = {[] [] [] [] [] [] [] [] []}; % pierwsza kolumna do
przechowywania elementów
columnb = {[] [] [] [] [] [] [] [] []}; % druga kolumna do przechowywania
elementów

sudoku = [ 9 0 4 1 3 0 0 8 6;
           0 6 1 0 0 0 0 0 7;
           8 0 0 7 6 4 9 1 2;
           6 0 0 0 0 9 1 0 8;
           5 0 3 8 7 0 6 0 9;
           1 9 0 0 5 0 3 7 4;
           0 1 9 4 0 7 0 0 5;
           0 0 0 9 0 0 2 4 3;
           0 0 2 6 8 3 7 0 0];
```

Zastosowanie podwójnej pętli w celu znalezienia rozwiązania dla losowej macierzy może być dobrym rozwiązaniem. Na początku należy oprzeć pętle o kod wynajdujący indeksy elementów zerowych macierzy. Po zachowaniu miejsca na przechowanie indeksów zerowych, należy napisać funkcję wypełniającą puste bloki elementami.

```
for m = 0 : 2
    for n = 0 : 2

        [a , b] = find (~sudoku (m*3+1 : m*3+3 , n*3+1 : n*3+3));
        columna{i} = a + m*3;
        columnb{i} = b + n*3;

        sudoku (m*3+1 : m*3+3 , n*3+1 : n*3+3) = fill (sudoku (m*3+1 :
m*3+3 , n*3+1 : n*3+3));
    end
end
```

Algorytm stosujący symulowane wyżarzanie wykorzystuje losowe dobieranie liczb, a potem sprawdzanie czy nowe rozwiązanie jest lepsze od wcześniej uzyskanego.

```
if   dr >= e (D)
    sudoku = D;
```

```
    else
        if  rand < 0.02
            sudoku = D;
        end
    end
end
```

Do wypełnienia macierzy brakującymi elementami, zastępując elementy zerowe konkretnymi liczbami, można zastosować poniższą funkcję:

```
function sudoku = fill(sudoku) % zapełnia macierz
    m = 1;
    z = setdiff (1 : 9 , sudoku);
    for n = 1 : 9
        if (sudoku(n) == 0)
            sudoku(n)= z (m);
            m = m + 1;
        end
    end
end
```

Przykład 2: Algorytmy genetyczne

Rozwinięcie wiedzy dotyczącej algorytmów heurystycznych i ich zastosowania. Zapoznanie się z algorytmem symulowanego wyżarzania za pośrednictwem programu Matlab i zapoznanie się z algorytmami genetycznymi.

I. Zagadnienia

- algorytmy genetyczne (GA);
- wielokryterialne algorytmy genetyczne (GA multi object);
- wykorzystanie programu Matlab do rozwiązywania problemów optymalizacyjnych;
- analiza algorytmów genetycznych.

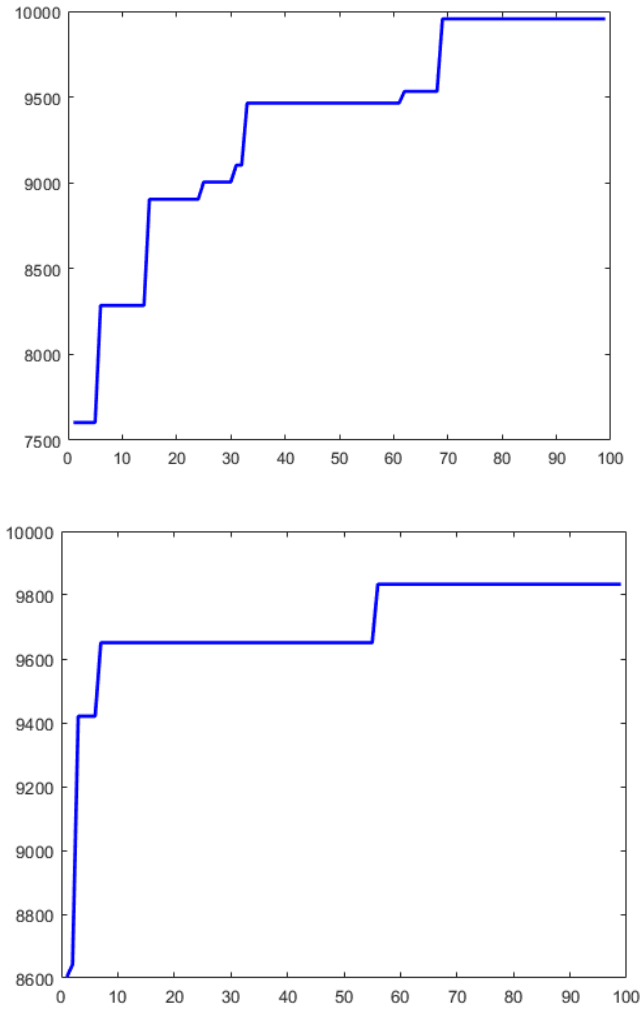
Wykorzystanie algorytmu genetycznego do wyszukania maksimum przy zastosowaniu dwóch zmiennych:

```
clc;
clear;

    max_x = input ('Podaj maksimum x: ');
    min_x = input ('Podaj minimum x: ');
    max_y = input ('Podaj maksimum y: ');
```

```
    min_y = input ('Podaj minimum y: ');
    fun = input ('Podaj funkcję: ','s');

    z = inline (fun ,'x','y');

for n = 1 : 4
    x(n) = (max_x - min_x)*rand(1,1) + min_x;
    y(n) = (max_y - min_y)*rand(1,1) + min_y;
    fit(n) = (feval(z , x(n) , y(n)));
end

i = 1;
while i < 100 %liczba zastosowanych cykli

    column = [ fit(1) fit(2) fit(3) fit(4)]

    [sorted,set] = sort (column , 'descend');

    fitness(i) = fit (set(1));
    iteration(i) = i;

    x (set(3)) = x (set(1));
    y (set(3)) = y (set(2));
    x (set(4)) = x (set(2));
    y (set(4)) = y (set(1));

    random = randi ([1 2] , 1 , 1);
    if random == 1
        x(set(4)) = (max_x - min_x)*rand(1,1) + min_x;
    else
        y(set(4)) = (max_y - min_y)*rand(1,1) + min_y;
    end

    for n = 1 : 4
        fit(n) = (feval(z , x(n) , y(n)));
    end

    i = i + 1;
end

plot(iteration (1 : 99),fitness (1 : 99),'Linewidth',2,'Color','b')

disp ('Uzykana maksymalna wartość x: ')
disp (x (set(1)));
disp ('Uzykana maksymalna wartość y: ')
disp (y (set(1)));
```

Można wykorzystać prostą funkcję $x + y$, gdzie zastosujemy wartości znajdujące się w przedziałach od 5 do 5000. Przy zastosowaniu takiej liczby cykli, odnalezione maksimum powinno znajdować się blisko wartości 5000 zarówno dla x, jak i y. Widzimy, że większa ich ilość, tym większa szansa uzyskania bardziej satysfakcjonującego rezultatu.

[Tekst alternatywny. Rysunek przedstawia wynik optymalizacji uzyskany za pomocą algorytmu genetycznego, pokazujący końcowe rozwiązanie po zakończeniu procesu ewolucyjnego oraz osiągnięcie najlepszego dopasowania spośród populacji rozwiązań.]



Rys. 3. Uzyskane rozwiązanie po wykorzystaniu algorytmu genetycznego

Warto również wspomnieć o funkcji Rastrigina. Omawiana funkcja jest typowym przykładem funkcji nieliniowej multimodalnej. Często funkcję Rastrigina stosuje się do testowania algorytmu genetycznego, jednak znalezienie optimum tej funkcji jest trudne ze względu na dużą liczbę minimów lokalnych jak i dużą przestrzeń.
Poniżej został przedstawiony skrypt stosujący funkcję Rastrigina.

Zastosowanie funkcji myfun:

```
function y= myfun(x)
y = 20 + x1^2 + x2^2 - 10 ( cos(6.28x1) + cos(6.28x2) );
clc;
clear;
opcjega = gaoptmiset ('PlotFons',@gaplotbestfun);
[x fval] = ga (myfun , 2 , opcjega)
```

Analizując algorytmy genetyczne warto wspomnieć o wielokryterialnym algorytmie genetycznym (GA multi object).
Genetyczny algorytm wielokryterialny opiera swoją budowę o zastosowanie operatorów, które mają swoje działanie w populacji. Na początek należy wylosować populację, by później ją ewoluować, za pomocą rangi niezdominowanej.
Poniżej przedstawiono plik przedstawiający funkcję wielokryterialną.

```
function y = gamul(x)
%funkcja wielokryterialna
y = zeros (2 , 1);
 for i = 1:2
  y(1) = y(1)  - 2*exp (-0.4*sqrt (x(i)^2 + x (i+1)^2)); %obliczenie celu
pierwszego
end
 for i = 1:3
   y(2) = y(2) +  abs (x(i))^1.5 + 5*sin (x(i)^2); %obliczenie drugiego
celu
end
```

Pplik posiada rzeczywistą funkcję składającą się z dwóch celów, posiadających po trzy zmienne.

```
%użycie funkcji optimoptions
n = 3; %określenie zmiennych
FitnessFunction = @gamul;
a = [10 10 10]; %określenie górnej granicy
b = [0 0 0]; %określenie dolnej granicy
%brak występujących ograniczeń dla nierówności liniowej
c1 = [];
```

```
d1 = [];
%brak występujących ograniczeń dla równości liniowej
c2 = [];
d2 = [];
i = optimoptions(@gamultiobj, 'PlotFcn' , @gaplotpareto);
```

Uruchomienie programu sprawi wykreślenie wykresu przedstawiającego działanie programu.
Zastosowanie metod hybrydowych, łącząc optymalizację genetyczną z innymi wartymi uwagi algorytmami wydaje się również dobrym pomysłem.

Przykład 3

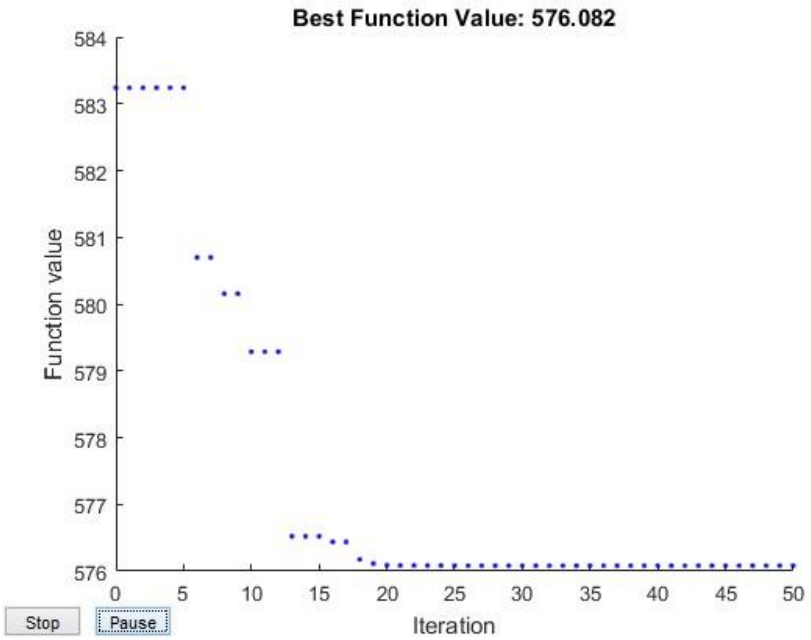
Poniżej zamieszczono skrypt dla obliczeń z wykorzystaniem algorytmu cząstek roju dostępnego w funkcji „particleswarm" programu Matlab. Podczas obliczeń poszukiwana była optymalna wartość funkcji kosztów dla problemu optymalizacji rozpływu mocy w systemie 30 – sto węzłowym, którego struktura w postaci macierzowej zapisana jest w pliku „case30" pakietu Matpower.

```
% skrypt przykład 4 ub = []; lb = []; for j = 1 : 1 : 6      lb(j) = 0.0;
ub(j) = 0.99; end options = optimoptions('particleswarm' , 'SwarmSize' ,
50 , 'MaxIterations'
, 50 , 'PlotFcn' , 'pswplotbestf' , 'StallIterLimit' , 40 );
rng default
nvars = 6; % liczba zmiennych decyzyjnych
[x , fval , exitflag , output] = particleswarm ( @funkcja_celu , nvars ,
lb , ub , options)
```

Wykres 4 prezentuje wyniki obliczeń dla algorytmu cząstek roju dostępny w funkcji particleswarm.

[Tekst alternatywny. Wykres przedstawia zmianę wartości funkcji celu w kolejnych iteracjach algorytmu optymalizacyjnego. Oś pozioma pokazuje numer iteracji, a oś pionowa przedstawia wartość funkcji celu, która maleje z każdym kolejnym krokiem obliczeń do poziomu najlepszego znalezionego rozwiązania wynoszącego 576.082]

*Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki"
nr FERS.01.05-IP.08-0234/23*



Rys. 4. Przebieg optymalizacji rozpływu mocy w sieci 30-węzłowej algorytmem cząstek roju

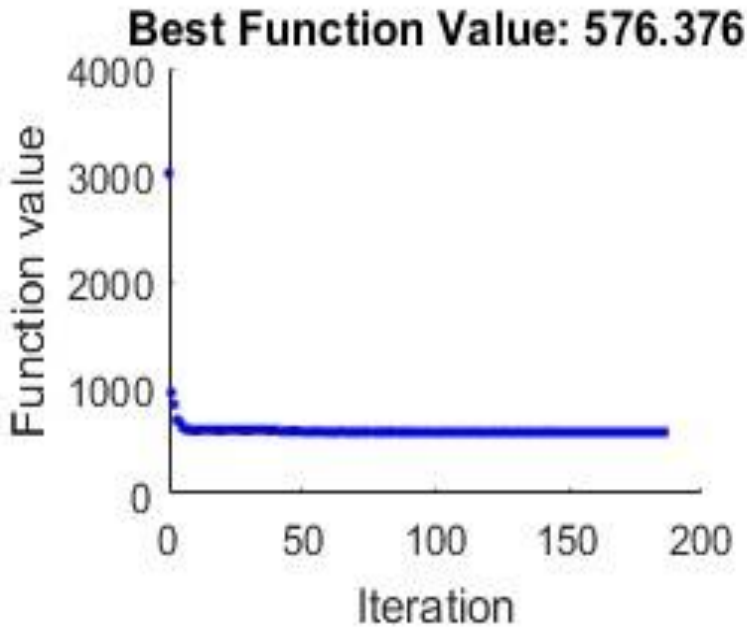
W obliczeniach poszukiwana była optymalna wartość funkcji kosztów dla problemu optymalizacji rozpływu mocy w sieci 30-węzłowej.

Następnie w kolejnym skrypcie przestawiono rozwiązanie problemu obliczania optymalnych rozpływów mocy w sieci 30-węzłowej z zastosowanie algorytmu symulowanego wyrzażania, który zawarty jest w funkcji „simulannealbnd" programu Matlab. W obliczeniach poszukiwana była optymalna wartość funkcji kosztów dla problemu optymalizacji rozpływu mocy w sieci 30-węzłowej.

```
% skrypt przykład
5 ub = []; lb =
[]; for j = 1 : 1
: 6      lb(j) =
0.0;      ub(j) =
0.99; end
options = optimoptions ( 'simulannealbnd' , 'PlotFcn' , '@saplotbestx'
); x0 = [0.1 0.1 0.1 0.1 0.1 0.1]; w = optimoptions ('simulannealbnd' ,
'PlotFcn' , {@saplotbestx ,
@saplotbestf , @saplotx , @saplotf}); x =
simulannealbnd (@funkcja_celu , x0 , lb , ub , w)
```

[Tekst alternatywny. Wykres przedstawia zmianę wartości funkcji celu w kolejnych iteracjach algorytmu optymalizacyjnego. Oś pozioma pokazuje numer iteracji, a oś

pionowa przedstawia wartość funkcji celu, która maleje z każdym kolejnym krokiem obliczeń do poziomu najlepszego znalezionego rozwiązania wynoszącego 576.082]



Rys. 5. Przebieg obliczeń dla problemu optymalizacji rozpływu mocy w sieci 30-węzłowej za pomocą symulowanego wyżarzania

Analizując przedstawione wcześniej rezultaty w zakresie obliczania optymalnych rozpływów mocy dla wybranych struktur systemów elektroenergetycznych, można zauważyć zbieżność w wynikach uzyskanych wybranymi metodami, co świadczy o poprawności przeprowadzonych obliczeń.
Poniżej zmieszczono kolejny przykład wykorzystujący metodę algorytmu symulowanego wyżarzania. W poniższym przykładzie algorytmu ten zastosowano do optymalizacji testowej funkcji, której opis zawarto w poniższym pliku o nazwie „fun”.

```
% skrypt przykład 6

% Wykonanie minimalizacji poprzez użycie piątej funkcji Dejonga
fun = @dejong5fcn; x0 = [0 0]; a = [-64 -64]; b = [64 64];
[x , fval] = simulannealbnd (fun , x0 , a , b)
```

Poniżej opis zawarto w poniższym pliku o nazwie „fun”:

```
fun = @dejong5fcn; x0 = [0 0]; w = optimoptions ('simulannealbnd' ,
'PlotFcn' , {@saplotbestx ,
```

```
@saplotbestf , @saplotx , @saplotf});
simulannealbnd (fun , x0 , [] , [] , w);
```

Wyliczono wektor zmiennych decyzyjnych x, oraz wartość funkcji celu:

```
x   = -15.9790   -31.9593
fval = 1.9920
```

Przykłady do samodzielnego wykonania za użyciem algorytmów genetycznych:

Przykład I

Znajdź minimum funkcji $y = e^{x_1} + e^{x_2}$, wobec następujących równościowych ograniczeń:

$x(1)^2 + x(2)^2 - 9 = 0$ oraz $-x_1 - x_2 + 1 < 0$, $-x_1 < 0$, $x_2 > 0$

Wynik rozwiązania: $x = [2.12 \quad 2.12]$

Funkcja celu = 16.6843

Przykład II

Znajdź minimum funkcji

$$y = (x_1 - 2)^2 + (x_1 - 1)^2 + 0.04 * \left(\frac{x_1^2}{4} - x_2^2 + 1\right) + 5\,(x_1 - 2x_2 + 1)^2 ,$$

przy granicach:

$x_1 = [-2 : 0.1 : 5]$ oraz $x_2 = [-1 : 0.1 : 4]$

Wynik rozwiązania: $x = [1.84 \quad 1.41]$

Funkcja celu = 0.1212

Literatura

1. Amborski, K., 2009. Podstawy metod optymalizacji, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa.
2. Arabas, J., 2004. Wykłady z algorytmów ewolucyjnych, Wydawnictwo NaukowoTechniczne, Warszawa.
3. Baczyński, D., 2013. Metody inteligencji obliczeniowej w elektroenergetyce, WPW, Warszawa.
4. Helt, P., Parol, M., Piotrowski, P., 2000. Metody sztucznej inteligencji w elektroenergetyce, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa.

5. Pijarski, P., 2019. Optymalizacji heurystyczna w ocenie warunków pracy i planowaniu rozwoju systemu elektroenergetycznego, Wydawnictwo Politechniki Lubelskiej, Lublin.
6. Stadnicki, J., 2006. Teoria i Praktyka Rozwiązywania Zadań Optymalizacyjnych, Wydawnictwo Naukowo-Techniczne, Warszawa.
7. Sysło, M., Deo, N., Kowalik, J., 1995. Algorytmy optymalizacji dyskretnej, Warszawa, PWN 1995.