„Algorytm optymalizacji rojem cząsteczek”

Opracował:

Oskar Furmańczuk

Warszawa 2020

1. Wstęp

Optymalizacja jest to jedna z dziedzin nauki, która towarzyszy nam od wieków. W latach 90 ubiegłego wieku obchodzono 300 lat nowożytnej teorii optymalizacji. Do rozwoju optymalizacji przyczyniły się nowatorskie prace fizyków i matematyków XVII wieku np. rachunek wariacyjny Lagrange za pomocą którego szukano ekstremum funkcji na określnych przestrzeniach. Zagadnienie optymalizacji występuje w różnych dziedzinach życia: w przemyśle samochodowym, chemicznym, finansach, podczas projektowania produktów i procesów produkcyjnych, wszędzie tam gdzie z kilku dostępnych wariantów musimy wybrać jeden i podjąć najlepszą (optymalną) decyzję.

Metody optymalizacji służą do wyznaczania najlepszego rozwiązania (poszukiwania ekstremum funkcji) z punktu widzenia określonego kryterium jakości (np. kosztów, wydajności, jakości). Możemy wyróżnić optymalizację lokalną oraz optymalizację globalną. Metody lokalne są stworzone do zadań optymalizacji w których celem jest znalezienie najlepszego stanu (z przestrzeni wielu rozwiązań) według funkcji celu, która jest funkcją heurystyczną. Wadą tej metody jest na pewno to, że znajdując jedno ekstremum w przestrzeni wielu rozwiązań nie możemy już sprawdzić, czy inne ekstrema są rozwiązaniami lepszymi. Mamy tutaj do czynienia z local minimum trap. W wielu problemach rzeczywistych funkcje wielu zmiennych mają dużą liczbą minimów i maksimów lokalnych (wielomodalność), dlatego do rozwiązania takich problemów najlepszym rozwiązaniem będą globalne metody optymalizacji.

Globalne metody optymalizacji służą do znajdowania „globalnego” ekstremum przeszukując globalną przestrzeń rozwiązań. Warto w tym miejscu wspomnieć o metaheurystykach, które służą do rozwiązywania globalnych problemów optymalizacji. Określenie to z greckiego oznacza „Szukanie wyższego poziomu”, co wynika z faktu, że algorytmy tego typu nie rozwiązują bezpośrednio problemu, ale podają sposób na utworzenie odpowiedniego algorytmu. Metody te są często inspirowane różnymi mechanizmami biologicznymi oraz fizycznymi. W niniejszej pracy przedstawiony zostanie algorytm optymalizacji rojem cząsteczek, który jest inspirowany obserwacją funkcjonowania organizmów żywych.

2. Wprowadzenie do PSO

Algorytm optymalizacji rojem cząsteczek (ang. PSO – particle swarm optimization) jest algorytmem powstałym na bazie obserwacji natury. Zauważono, iż zwierzęta żyjące w stadach wybierają dla siebie najkorzystniejsze czynniki środowiska. Na podstawie tych obserwacji stworzono algorytmy, które dobrze sprawdzały się w praktyce, gdyż potrafiły rozwiązać niemal każdy problem niezależnie od postaci funkcji celu, czy nałożonych ograniczeń – zupełnie jak zwierzęta, które zawsze dążą do zapewnienia sobie przetrwania. Algorytmy te wykorzystują reguły rachunku prawdopodobieństwa, a najbardziej popularne z nich to algorytm pszczeli, ławicy ryb, mrówkowy, czy omawiany w naszej pracy – rojem cząstek, którego zastosowanie pomaga znaleźć ekstrema globalne dla funkcji jedno i wielomodalnych.

Początkowe prace teoretyczne nad algorytmem PSO skupiały się na ustaleniu optymalnej wielkości poszczególnych parametrów wielkości oraz stworzeniu takich modyfikacji, które będą przeważały nad poprzednimi modelami. Do porównań wybierano wersje metody kanonicznej ustalane na podstawie wartości teoretycznych. Rozważania teoretyczne zostały pozbawione czynnika losowego, a przyjęcie wartości skrajnych, czy ograniczenie liczby badanych parametrów umożliwiły zastosowanie metod analitycznych. W 1995 roku przez Kennede'go oraz Engelbrechta, został zaproponowany algorytm optymalizacji rojem cząstek w postaci stochastycznej z wprowadzonym do modelu czynnikiem zdolności eksploracyjnej stada.

3. Działanie algorytmu

Jak już wspomniano w poprzednim rozdziale idea metaheurytycznego algorytmu wywodzi się z naśladowania zachowania populacji żywych istot i symuluje adaptacje stada do środowiska. Rój stanowi grupę osobników – cząstek danego gatunku, które mają ograniczoną możliwość podejmowania decyzji oraz wzajemnej komunikacji. Cząstki przemieszczają się do nowych położeń, czyli szukają optimum, a w przełożeniu na model numeryczny oznacza to znalezienie takich wartości parametru, które zapewniają największą skuteczność powodzenia.

Każdy osobnik (cząstka) przemieszcza się z prędkością określoną na podstawie doświadczeń własnych oraz innych osobników – iteracyjnie przeszukuje przestrzeń rozwiązań problemu przy pomocy roju cząstek. Prędkości te dobierane są tak by był zachowany odpowiedni dystans między nimi. Prędkość ruchu poszczególnych cząstek zależy od najlepszego położenia globalnego i lokalnego rozwiązania oraz od prędkości zastosowanej w poprzednich krokach. Cząstki zbierają i zapamiętują wszelkie informacje, tak by z uzyskanego zbioru informacji zmieniać swoje położenie, dążąc do punktu o najlepszych własnościach - znaleźć najlepszą wartość funkcji celu. Inteligencję – czyli uzyskany ów zbiór opisuje pięć najważniejszych zasad inteligencji zdefiniowany przez Eberharta (1996) oraz Millanisa (1994):

- Bliskość - zdolność do obliczania czasu i odległości – stado działa na prostej strukturze przestrzeni
- Jakość - zdolność oceny oddziaływania z otoczeniem – stado powinno mieć możliwość reagowania na jakościowe zmiany współczynników środowiska
- zróżnicowanie reakcji - zdolność tworzenia różnych reakcji - stado nie powinno dokonywać pewnych niebezpiecznych ruchów
- Stabilność - zdolność do tworzenia zachowania o umiarkowanej reakcji na zmiany w otoczeniu - stado nie powinno zmieniać swego zachowania, gdy zmienia się środowisko
- Adaptacja - zdolność do zmian w zachowaniu, jeśli wymusza to otoczenie – stado musi mieć możliwość zmiany zachowania, gdy poniesiony koszt tego nie zabrania.

Poniższy wzór pozwala na obliczenie prędkości danej cząstki oraz na aktualizacje prędkości wszystkich cząstek na podstawie uzyskanej wcześniej wiedzy:

$$V_{ij}^{t} = wv_{ij}^{(t-1)} + c_1 r_1 \left(p_{ij} - x_{ij}^{(t-1)}\right) + c_2 r_2 (G_j - x_{ij}^{(t-1)})$$

V – prędkość cząstki

w – współczynnik bezwładności, określa wpływ prędkości w poprzednim kroku w

c_1 – współczynnik dążenia do najlepszego lokalnego rozwiązania (kognitywny) c1

c_2– współczynnik dążenia do najlepszego globalnego rozwiązania(socjalny) c2

p– położenie najlepszego lokalnego rozwiązania p

G – położenie najlepszego globalnego rozwiązania G

x – położenie cząsteczki x

$r_1 r_2$ – losowe wartości z przedziału [0,1]

Nową pozycję cząstki i wyznaczana jest za pomocą wzoru:

$$x_{ij}^{t} = x_{ij}^{t-1} + v_{ij}^{t}$$

5. Schemat działania algorytmu

Algorytm optymalizacji rojem cząstek działa w następujący sposób:

1. Ustaw wartości początkowe parametrów: w, c_1, c_2
2. Ustaw $t = 0$. Wygeneruj początkowy rój $R(t)$.
3. Dla każdej cząstki i w $R(t)$ wyznacz wartości funkcji celu $f(Xi)$.
4. Zapamiętaj najlepszą cząstkę G w $R(t)$. Dla każdej cząstki i w $R(t)$ zapamiętaj jej aktualną pozycję jako najlepszą p_i .
5. Dopóki kryterium stopu nie jest spełnione, wykonuj:
    - Ustaw $t = 1 + 1$.
    - Wygeneruj nowy rój $R(t)$ na podstawie $R(t-1)$.
    - Dla każdej cząstki i w $R(t)$ wyznacz wartość funkcji celu $f(Xi)$.
    - Dla każdej cząstki i w $R(t)$ zaktualizuj jej najlepszą pozycję p_i .
    - Zaktualizuj najlepszą pozycję G w całym roju $R(t)$.
6. Zwróć najlepsze znalezione rozwiązanie G

6. Praktyczny przykład

W pewnym przedsiębiorstwie produkującym napoje słodzone przeprowadzono badania nad związkiem pomiędzy składem produkowanego soku, a zyskiem z jego sprzedaży. Kluczowym składnikiem okazał się cukier i koncentrat z pomarańczy. Funkcją, która najlepiej odzwierciedlała związek ilość cukru oraz koncentratu z pomarańczy od uzyskanego zysku okazała się być:

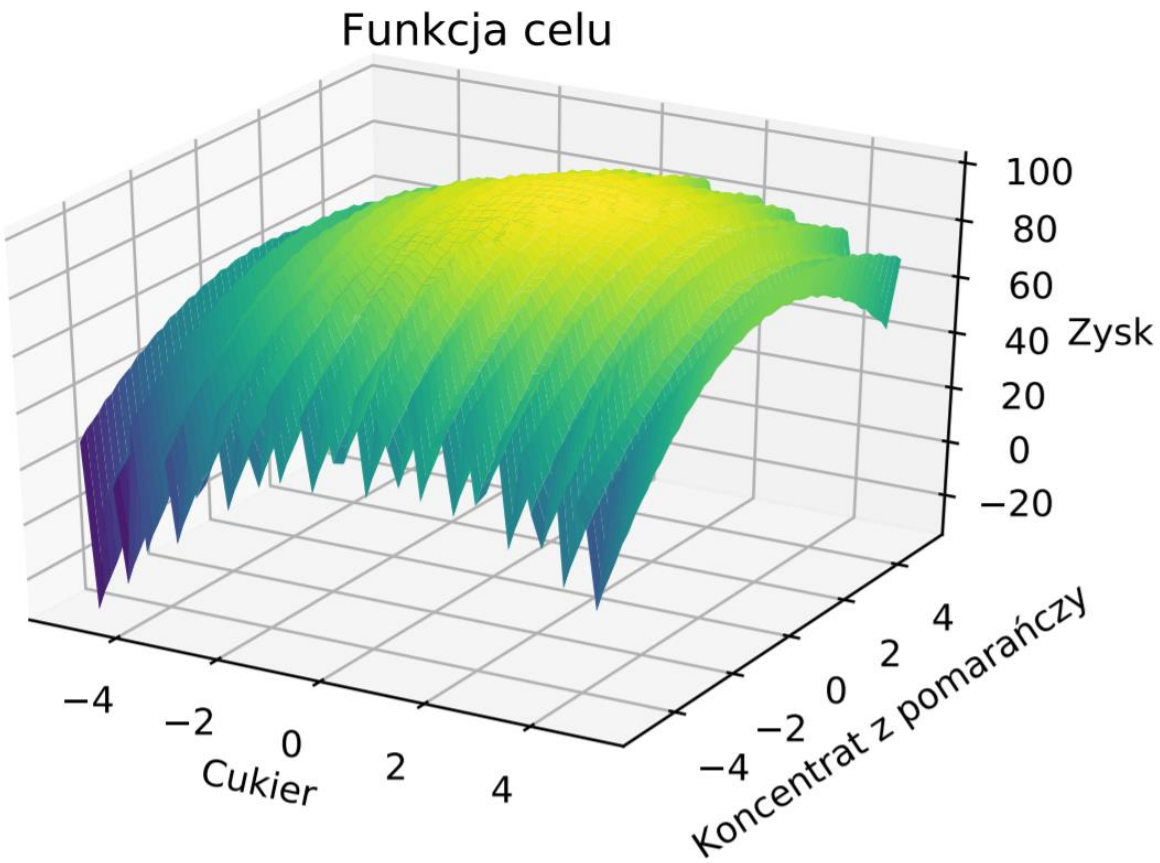
$$f(x) = y = -sin^2(3\pi x_1) - (x_1 - 1)^2[1 + sin^2(3\pi x_2)] - (x_2 - 1)^2[1 + sin^2(2\pi x_2)] + 100$$

Gdzie:

x_1 – ilość cukru dodanego do tradycyjnej receptury (na litr soku) [g/l]

x_2 – ilość koncentratu z pomarańczy dodanego do tradycyjnej receptury (na litr soku) [ml/l]

y – zysk ze sprzedaży jednego hektolitra soku [PLN]



Do wyznaczenia ilość składników odpowiedzialnych za najwyższy zysk niezbędne jest znalezienie globalnego maksimum funkcji f(x). W tym celu posłużymy się algorytmem optymalizacji rojem cząsteczek zaimplementowanym przy użyciu języka Python (skrypt poniżej).

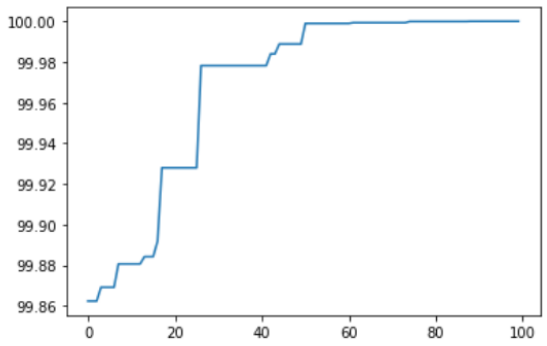
```python
import matplotlib.pyplot as plt
import random
import math

# nadanie parametrów dla PSO
dziedzina = [(-5, 5), (-5, 5)]
ilosc_zmiennych = 2
ilosc_czasteczek = 100
ilosc_iteracji = 100
w = 0.85
c1 = 1
c2 = 2
wart_inicjacyjna = -float("inf")


# implementacja funkcji celu
def funkcja_celu(x):
    y = -(math.sin(3 * math.pi * x[0])) ** 2 - ((x[0] - 1) ** 2) * (1 + (math.sin(3 * math.pi * x[1])) ** 2) - (
                (x[1] - 1) ** 2) * (1 + (math.sin(2 * math.pi * x[1])) ** 2) + 100
    return y


# klasa przechowująca własności pojedynczej cząsteczki wchodzącej w skład roju
class Czasteczka:
    def __init__(self, dziedzina):
        self.pozycja_czasteczki = []
        self.predkosc_czasteczki = []
        self.lokalna_najlepsza_pozycja_czasteczki = []
        self.wartosc_funkcji_dla_najlepszego_lokalnego_polozenia = wart_inicjacyjna
        self.wartosc_funcji_dla_polozenia_czasteczki = wart_inicjacyjna

        for i in range(ilosc_zmiennych):
            self.pozycja_czasteczki.append(
                random.uniform(dziedzina[i][0], dziedzina[i][1]))
            self.predkosc_czasteczki.append(random.uniform(-1, 1))

    def porownaj(self, funkcja_celu):
        self.wartosc_funcji_dla_polozenia_czasteczki = funkcja_celu(self.pozycja_czasteczki)
        if self.wartosc_funcji_dla_polozenia_czasteczki > self.wartosc_funkcji_dla_najlepszego_lokalnego_polozenia:
            self.lokalna_najlepsza_pozycja_czasteczki = self.pozycja_czasteczki
            self.wartosc_funkcji_dla_najlepszego_lokalnego_polozenia = self.wartosc_funcji_dla_polozenia_czasteczki

    def przelicz_predkosc(self, najlepsza_globalna_pozycja):
        for i in range(ilosc_zmiennych):
            r1 = random.random()
            r2 = random.random()

            predkosc_kongnitywna = c1 * r1 * (self.lokalna_najlepsza_pozycja_czasteczki[i] -
self.pozycja_czasteczki[i])
            predkosc_socjalna = c2 * r2 * (najlepsza_globalna_pozycja[i] - self.pozycja_czasteczki[i])
            self.predkosc_czasteczki[i] = w * self.predkosc_czasteczki[i] + predkosc_kongnitywna + predkosc_socjalna

    def przelicz_pozycje(self, dziedzina):
        for i in range(ilosc_zmiennych):
            self.pozycja_czasteczki[i] = self.pozycja_czasteczki[i] + self.predkosc_czasteczki[i]
            if self.pozycja_czasteczki[i] > dziedzina[i][1]:
                self.pozycja_czasteczki[i] = dziedzina[i][1]
            if self.pozycja_czasteczki[i] < dziedzina[i][0]:
                self.pozycja_czasteczki[i] = dziedzina[i][0]
```

```
# klasa przechowująca własności roju cząsteczek wraz ze zwracaniem finalnego rozwiązania problemu
class Optymalizacja_Rojem_Czasteczek():
    def __init__(self, funkcja_celu, dziedzina, ilosc_czasteczek, ilosc_iteracji):

        wartosc_funckcji_dla_najlepszej_globalnej_pozycji = wart_inicjacyjna
        najlepsza_globalna_pozycja = []
        pozycje_czasteczek = []

        for i in range(ilosc_czasteczek):
            pozycje_czasteczek.append(Czasteczka(dziedzina))
        wektor_najwyzszych_wartosci_funkcji = []

        for i in range(ilosc_iteracji):
            for j in range(ilosc_czasteczek):
                pozycje_czasteczek[j].porownaj(funkcja_celu)
                if pozycje_czasteczek[
                    j].wartosc_funcji_dla_polozenia_czasteczki > wartosc_funckcji_dla_najlepszej_globalnej_pozycji:
                    najlepsza_globalna_pozycja = list(pozycje_czasteczek[j].pozycja_czasteczki)
                    wartosc_funckcji_dla_najlepszej_globalnej_pozycji = float(
                        pozycje_czasteczek[j].wartosc_funcji_dla_polozenia_czasteczki)
            for j in range(ilosc_czasteczek):
                pozycje_czasteczek[j].przelicz_predkosc(najlepsza_globalna_pozycja)
                pozycje_czasteczek[j].przelicz_pozycje(dziedzina)

            wektor_najwyzszych_wartosci_funkcji.append(
                wartosc_funckcji_dla_najlepszej_globalnej_pozycji)

        print('Optymalna wartość dla x1 i x2:', najlepsza_globalna_pozycja)
        print('Wartość maksymalnych zysków:', wartosc_funckcji_dla_najlepszej_globalnej_pozycji)
        print('Wykres maksymalnych wartości funkcji celu osiąganych przez rój cząsteczek w kolejnych iteracjach')
        plt.plot(wektor_najwyzszych_wartosci_funkcji)
        plt.show()


# wywołanie algorytmu z zadanymi parametrami oraz funckcją celu
Optymalizacja_Rojem_Czasteczek(funkcja_celu, dziedzina, ilosc_czasteczek, ilosc_iteracji)
```

Jako wynik wywołania powyższego programu w środowisku Jupyter Notebook otrzymujemy:

```
# wywołanie algorytmu z zadanymi parametrami oraz funckcją celu
Optymalizacja_Rojem_Czasteczek(funkcja_celu, dziedzina, ilosc_czasteczek, ilosc_iteracji)
```

```
Optymalna wartość dla x1 i x2: [1.0001863641720476, 1.0035606491015627]
Wartość maksymalnych zysków: 99.99998419558106
Wykres maksymalnych wartości funkcji celu osiąganych przez rój cząsteczek w kolejnych iteracjach
```




```
Out[1]: <__main__.Optymalizacja_Rojem_Czasteczek at 0x2b4a3346988>
```

Algorytm znalazł maksimum funkcji wynoszące 99.99998419558106 dla X_1 = 1.000186364 oraz X_2 = 1.0035606491015627 co w przybliżeniu do części tysięcznych ułamka jest tożsame z maksimum funkcji -g(x) + 100, gdzie g(x) to funkcja Levy'ego nr 13.[1]

[1] Simon Fraser University: https://www.sfu.ca/~ssurjano/levy13.html

Rezultatem analiz prowadzonych dla przedsiębiorstwa produkującego napoje słodzone byłaby rekomendacja zmiany receptury badanego soku. Do każdego litra soku należałoby dodać o 1g więcej cukru oraz o 1 ml koncentratu z soku pomarańczy więcej. Zysk ze sprzedaży hektolitra takiego soku wynosiłby 100 PLN.

7. Inne warianty PSO

Możliwych jest wiele wariantów nawet podstawowego algorytmu PSO. Istnieją różne sposoby inicjalizacji cząstek i prędkości (np. rozpoczynanie od prędkości zerowych), sposób tłumienia prędkości, aktualizacja pi i g dopiero po aktualizacji całego roju, itp.

Wiodący naukowcy stworzyli serię standardowych wdrożeń, "przeznaczonych do wykorzystania zarówno jako punkt odniesienia dla testów wydajnościowych ulepszeń techniki, jak i do reprezentowania PSO w szerszej społeczności optymalizacyjnej. Posiadanie dobrze znanego, ściśle określonego standardowego algorytmu stanowi cenny punkt odniesienia, który może być wykorzystany w całym obszarze badań w celu lepszego testowania nowych osiągnięć". Najnowszym jest Standard PSO 2011 (SPSO-2011).[2]

- Hybrydyzacja

Nowe i bardziej wyrafinowane warianty PSO są również stale wprowadzane w celu poprawy wydajności optymalizacji. Istnieją pewne trendy w tych badaniach; jednym z nich jest stworzenie hybrydowej metody optymalizacji wykorzystującej PSO w połączeniu z innymi optymalizatorami, np. połączenie PSO z optymalizacją opartą na biogeografii, oraz włączenie skutecznej metody uczenia się.[3]

- Złagodzenie przedwczesnej konwergencję

Innym trendem badawczym jest próba złagodzenia przedwczesnej konwergencji (czyli stagnacji optymalizacyjnej), np. poprzez odwrócenie lub zakłócenie ruchu cząstek PSO, innym podejściem do problemu przedwczesnej konwergencji jest zastosowanie wielu rojów (optymalizacja wielorojowa). Podejście wielorojówkowe może być również wykorzystane do wdrożenia optymalizacji wielocelowej. Finalnie są postępy w dostosowywaniu parametrów behawioralnych PSO podczas optymalizacji.[4]

- Uproszczenia

Inną szkołą myślenia jest to, że PSO należy jak najbardziej uprościć bez uszczerbku dla jego wydajności; ogólna koncepcja często określana jako brzytwa Occama. Uproszczenie PSO zostało pierwotnie zasugerowane przez Kennedy'ego i było przedmiotem szerszych badań, gdzie okazało się, że wydajność optymalizacyjna została poprawiona, a parametry były łatwiejsze do dostrojenia i działały bardziej konsekwentnie w różnych problemach optymalizacyjnych.

---

[2] Zambrano-Bigiarini, M.; Clerc, M.; Rojas, R. (2013). "Standard Particle Swarm Optimisation 2011 at CEC-2013"
[3] Zhan, Z-H.; Zhang, J.; Li, Y; Shi, Y-H. (2011). "Orthogonal Learning Particle Swarm Optimization"
[4] Zhan, Z-H.; Zhang, J.; Li, Y; Chung, H.S-H. (2009). "Adaptive Particle Swarm Optimization"

Innym argumentem przemawiającym za uproszczeniem PSO jest to, że metaheurystyka może wykazać swoją skuteczność tylko empirycznie, przeprowadzając eksperymenty obliczeniowe na określonej liczbie problemów optymalizacyjnych. Oznacza to, że metaheurystyka taka jak PSO nie może zostać udowodniona jako poprawna, a to zwiększa ryzyko popełnienia błędów w jej opisie i wdrożeniu. Dobrym przykładem jest tu obiecujący wariant algorytmu genetycznego (kolejny popularny metaheurystyczny), który okazał się jednak wadliwy, gdyż w swoich poszukiwaniach optymalizacyjnych był silnie stronniczy w stosunku do podobnych wartości dla różnych wymiarów w przestrzeni badawczej, co okazało się być optymalnym z rozważanych problemów benchmarkowych. To odchylenie wynikało z błędu w programowaniu i zostało naprawione.

Inicjalizacja prędkości może wymagać dodatkowych wejść. Wariant Bare Bones PSO został zaproponowany w 2003 roku przez Jamesa Kennedy'ego i nie musi w ogóle używać prędkości.

Innym prostszym wariantem jest przyspieszona optymalizacja roju cząstek (APSO), która również nie musi używać prędkości i może przyspieszyć konwergencję w wielu zastosowaniach. Dostępny jest prosty kod demonstracyjny APSO.[5]

- Wielocelowa optymalizacja

PSO stosuje się również do problemów wielozadaniowych, w których obiektywne porównanie funkcji uwzględnia dominację pareto przy przemieszczaniu cząstek PSO, a rozwiązania niezdominowane są przechowywane tak, aby zbliżyć się do frontu pareto.[6]

- Binarne, dyskretne i kombinatoryczne

Ponieważ podane powyżej równania PSO działają na liczbach rzeczywistych, powszechnie stosowaną metodą rozwiązywania problemów dyskretnych jest mapowanie dyskretnej przestrzeni wyszukiwania do domeny ciągłej, stosowanie klasycznego PSO, a następnie demapowanie wyniku. Takie mapowanie może być bardzo proste (na przykład poprzez użycie tylko zaokrąglonych wartości) lub bardziej wyrafinowane.

Można jednak zauważyć, że w równaniach ruchu wykorzystuje się operatorów, którzy wykonują cztery czynności:

- ✓ obliczając różnicę dwóch pozycji. Wynikiem tego jest prędkość (a dokładniej przemieszczenie)
- ✓ pomnożenie prędkości przez współczynnik liczbowy
- ✓ dodawanie dwóch prędkości
- ✓ przykładanie prędkości do pozycji

Zazwyczaj pozycja i prędkość są reprezentowane przez n liczb rzeczywistych, a te operatory to po prostu -, *, +, i jeszcze raz +. Ale wszystkie te obiekty matematyczne można zdefiniować w zupełnie inny sposób, aby poradzić sobie z problemami binarnymi (lub bardziej ogólnie dyskretnymi), a nawet kombinatorycznymi. Jednym z podejść jest redefiniowanie operatorów w oparciu o zbiory.[7]

---

[5] "Search Results: APSO - File Exchange - MATLAB Central"

[6] Zhan, Z-H.; Zhang, J.; Li, Y; Shi, Y-H. (2011). "Orthogonal Learning Particle Swarm Optimization"

[7] Chen, Wei-neng; Zhang, Jun (2010). "A novel set-based particle swarm optimization method for discrete optimization problem"