Algorytm Cuckoo Search (CS)

- CS – metoda optimymalizacji stochastycznej (Xin-she Yang & Suash Deb, 2009)
- CS naśladuje zachowanie niektórych gatunków kukułki, które wykorzystują gniazda innych ptaków do wylęgu jaj i wychowywania piskląt
- CS jest rozszerzony o mechanizm „lotów Levy'ego" czyli wykonywania skoków w oparciu o rozkład Levy'ego

Algorytm CS

- Ostatnie badania sugerują, że CS może przewyższać inne algorytmów, takie jak optymalizacja rojem cząsteczek (PSO)

- W porównaniu do PSO i ABC, algorytm CS zapewnia bardziej odporne przeszukiwania przestrzeni rozwiązań

Tło biologiczne algorytmu CS

- Kukułki:
  - piękne dźwięki
  - agresywna strategia reprodukcji

- Niektóre gatunki kukułek składają jaja we wspólnych gniazdach i mogą usunąć inne jaja w celu zwiększenia prawdopodobieństwa wylęgu własnych jaj

Tło biologiczne algorytmu CS

- Spora liczba gatunków kukułki wykorzystuje pasożytnictwo, polegające na składaniu swoich jaj do gniazd innych ptaków

- Trzy typy pasożytniczego zachowania:
  - wewnątrzgatunkowe
  - hodowlana współpraca
  - przejmowanie gniazd

Tło biologiczne algorytmu CS

- Niektóre ptaki z gniazd gospodarzy mogą zaangażować się bezpośrednio w konflikt z kukułkami

- Ptaki, które odkryją, że jaja nie są ich:
  - wyrzucają z gniazd obce jaja
  - opuszczają swoje gniazdo i budują nowe w innym miejscu

Tło biologiczne algorytmu CS

- Niektóre gatunki z kukułek wyewoluowały w taki sposób, są często wyspecjalizowane w mimikrę czyli naśladownictwo w kolorze i fakturze jaj kilku gatunków

- Zmniejsza to prawdopodobieństwo, że ich jaja będą porzucone, co z kolei zwiększa ich rozrodczość

Tło biologiczne algorytmu CS

- Pasożytnicze kukułki często wybierają gniazda ptaków, które właśnie złożyły niedawno jaja
- Jaja kukułki wylęgają nieco wcześniej niż jaja gospodarza
- Pierwszym działaniem instynktu wyklutych osobników jest eksmisja (wyrzucenie) z gniazda jaj ptaka gospodarza
- Pisklę kukułki może naśladować dźwięki piskląt gospodarza, aby uzyskać dostęp do większej ilości pokarmu

Algorytm lotów Levy’ego

- Zachowanie lotu wielu zwierząt i owadów można opisać za pomocą lotów Lévy’ego

- Lot Lévy’ego ma charakter błądzenia losowego, w którym długość kroku jest wyznaczana w oparciu o rozkład „długiego ogona”

Algorytm lotów Levy’ego

- „Lot Lévy" odnosi się do francuskiego matematyka Paula Pierre Lévy
- Wyrażenie „loty Lévy’ego“ (Mandelbrot B.) uzyte do zdefiniowania rozmiaru kroku wg rozkładu Lévy’ego
- Termin lot Cauchy'ego jest używany do określenia wielkości kroku wg rozkładu Cauchy'ego
- Lot Rayleigh jest opisywany wg rozkładu normalnego

Algorytm lotów Levy’ego

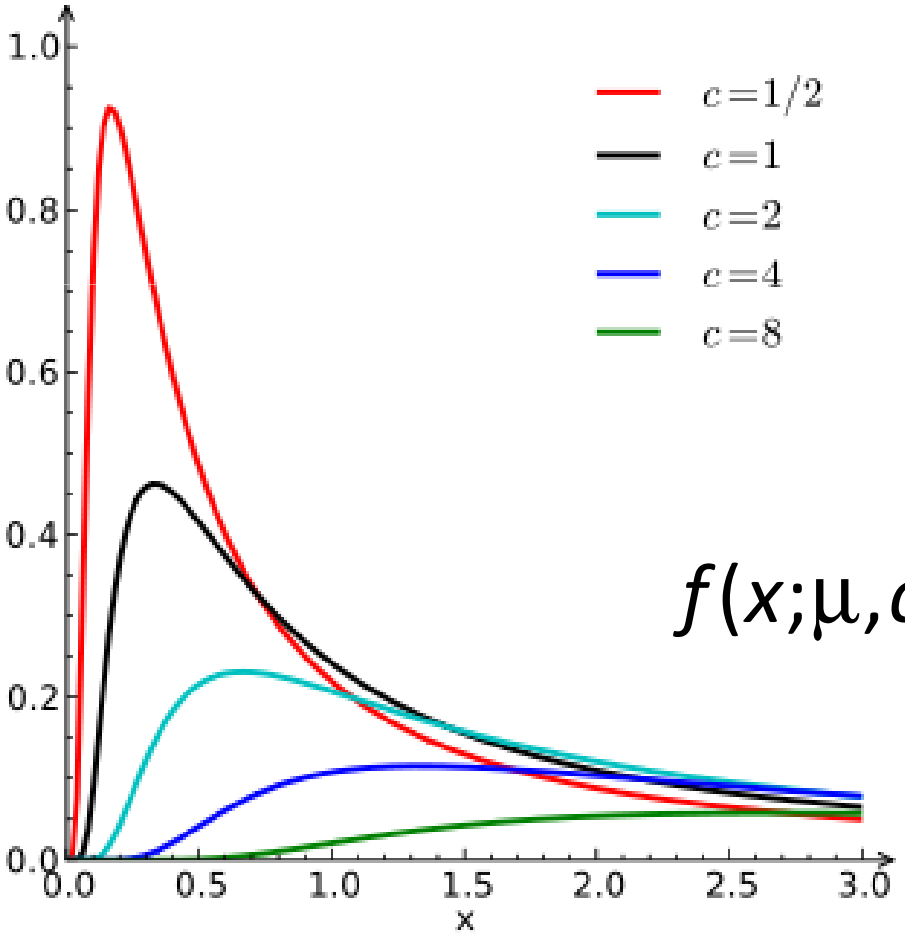
- Poźniejsi badacze rozszerzyli użycie określenia „lot Lévy’ego" dla przypadków gdzie błądzenie przypadkowe dotyczy dyskretnej przestrzeni

Algorytm lotów Levy’ego

- Rozkład Lévy’ego jest ciągłym rozkładem prawdopodobieństwa dla nieujemnych zmiennych losowych:
  - profil Waals’a (spetroskopia)
  - szczególny przypadek odwrotnego rozkładu gamma
  - jeden z nielicznych rozkładów, który jest stabilny i posiada funkcje gęstości prawdopodobieństwa wyrażoną analitycznie

Algorytm lotów Levy'ego

- Funkcja gęstości prawdopodobieństwa



$$f(x;\mu,c)=\sqrt{\frac{c}{2\pi}}\frac{e^{-\frac{c}{2(x-\mu)}}}{(x-\mu)^{1.5}}$$

Algorytm CS

- Trzy uproszczone reguły:

  1. Każdy kukułka składa jedno jajo w danym momencie czasu w losowo wybranym gnieździe
  2. Najlepsze gniazda z wysoką jakością jaj będą przenoszone na następne pokolenia
  3. Liczba dostępnych gniazd jest stała, a jajo podrzucone przez kukułkę jest wykryte z prawdopodobieństwem $p_a \in [0,1]$

Algorytm CS

- W tym przypadku, ptak gospodarz może albo wyrzucać jajko z gniazda lub zrezygnować z niego i zbudować zupełnie nowe
- Dla uproszczenia, ostatnie założenie może być oszacowaniem pewnej ilości gniazd, które zastępowane są przez nowe gniazda (z nowymi losowymi rozwiązaniami) p_a
- Jakości lub przydatność rozwiązania może być po proporcjonalna do wartości funkcji celu

Algorytm CS

- Jajo w gnieździe reprezentuje rozwiązanie
- Jajo kukułki jest nowym rozwiązaniem
- Nowego i potencjalnie lepsze rozwiązanie (jajo kukułki) zastępuje słabe rozwiązanie w gnieździe
- Algorytm może być rozszerzony do wersji gdzie w którym wiele jaj reprezentuje zbiór rozwiązań
- Proste podejście – każde gniazdo ma jedno jajo

Schemat algorytmu CS

Procedura CS

begin

Wylosuj początkową populację *n gniazd* x_i (i = 1, 2, ..., n)

while (t <MaxLiczbaPokoleń) lub (kryterium stopu)

Wybierz losowo kukułkę przez lot L´evy i oceń ich f_i

Wybierz losowo spośród n kukułek (np. j)

if ($f_i > f_j$), **then** podstaw za *j-tą* nową wartość **end**

Część (p_a) słabych gniazd opuszczona w celu zbudowania nowego

Zapamiętaj najlepsze gniazdo z odpowiednim przystosowaniem

Utwórz ranking dla osobników z populacji

end

end

Szczegółowy opis algorytmu CS

- Stosowane oznaczenia:

  $f(\boldsymbol{x})$ - funkcja celu

  $\boldsymbol{x} = [x_1 \quad x_2 \quad \cdots \quad x_m]$ - szukane rozwiązanie

  $\boldsymbol{x}_i$ - *i*-te gniazdo (jedno jajo kukułki w gnieździe)

  $f(\boldsymbol{x}_i)$ - przystosowanie (jakość) *i*-tej kukułki

Szczegółowy opis algorytmu CS

- Tworzenie nowego rozwiązania:

$$\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \alpha \cdot u$$

gdzie

$\alpha > 0$ - wielkość kroku (związana z wielkością dziedziny parametrów)

u - liczba wylosowana z rozkładem Lévy'ego (nieskończona wariancja i wartość średnia)

$$u \sim t^{-\lambda},\ 1 < \lambda \leq 3$$

Szczegółowy opis algorytmu CS

- Niektóre nowe rozwiązania powinny być generowane na podstawie lotu Lév'iego wokół najlepszego rozwiązania, co powoduje przyspieszenie lokalnych poszukiwań
- Znaczna część nowych rozwiązań powinna być generowana w odległych regionach niż obecne najlepsze rozwiązanie
- Pozwoli to upewnić się, że system nie będzie uwięziony w lokalnym optimum

Szczegółowy opis algorytmu CS

- Podobieństwo CS do losowej wspinaczki
- Znaczące różnice:
  - CS populacyjny algorytm
  - wykorzystuje elityzm
  - losowość w wielkości kroku daje wiekszą skuteczność poszukiwań
  - liczba parametrów do strojenia algorytmu znacznie mniejsza
  - każde gniazdo reprezentuje zbiór rozwiązań

Przykładowy problem szukania

- Funkcja Michalewicza:

$$f(x,y) = -\sin(x)\sin^{20}\left(\frac{x^2}{\pi}\right) - \sin(y)\sin^{20}\left(\frac{2y^2}{\pi}\right)$$

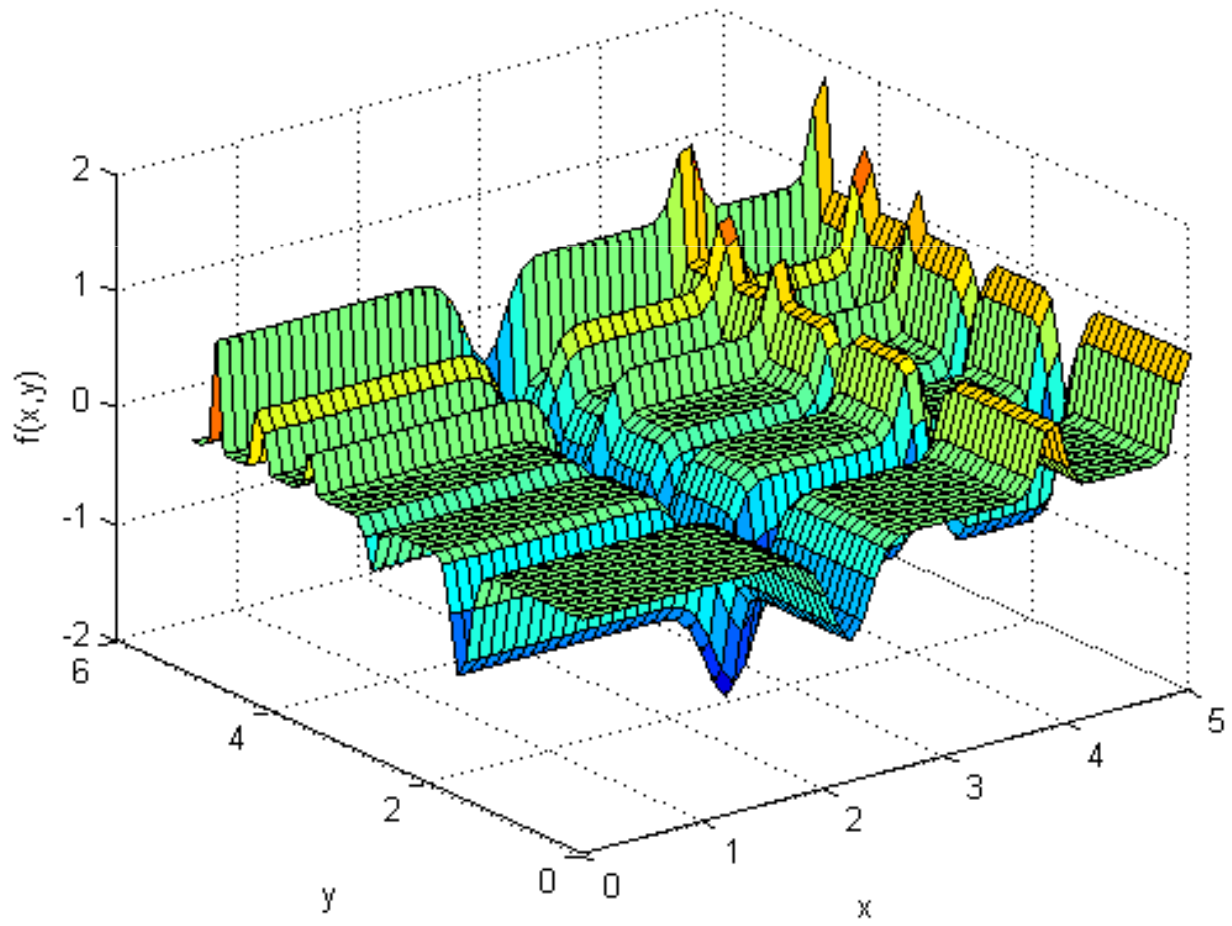
gdzie

$(x,y) \in [0,5] \times [0,5]$ - przestrzeń poszukiwań

globalne minimum -1.8013 w (2.2031, 1.5704)

Przykładowy problem szukania

- Funkcja Michalewicza



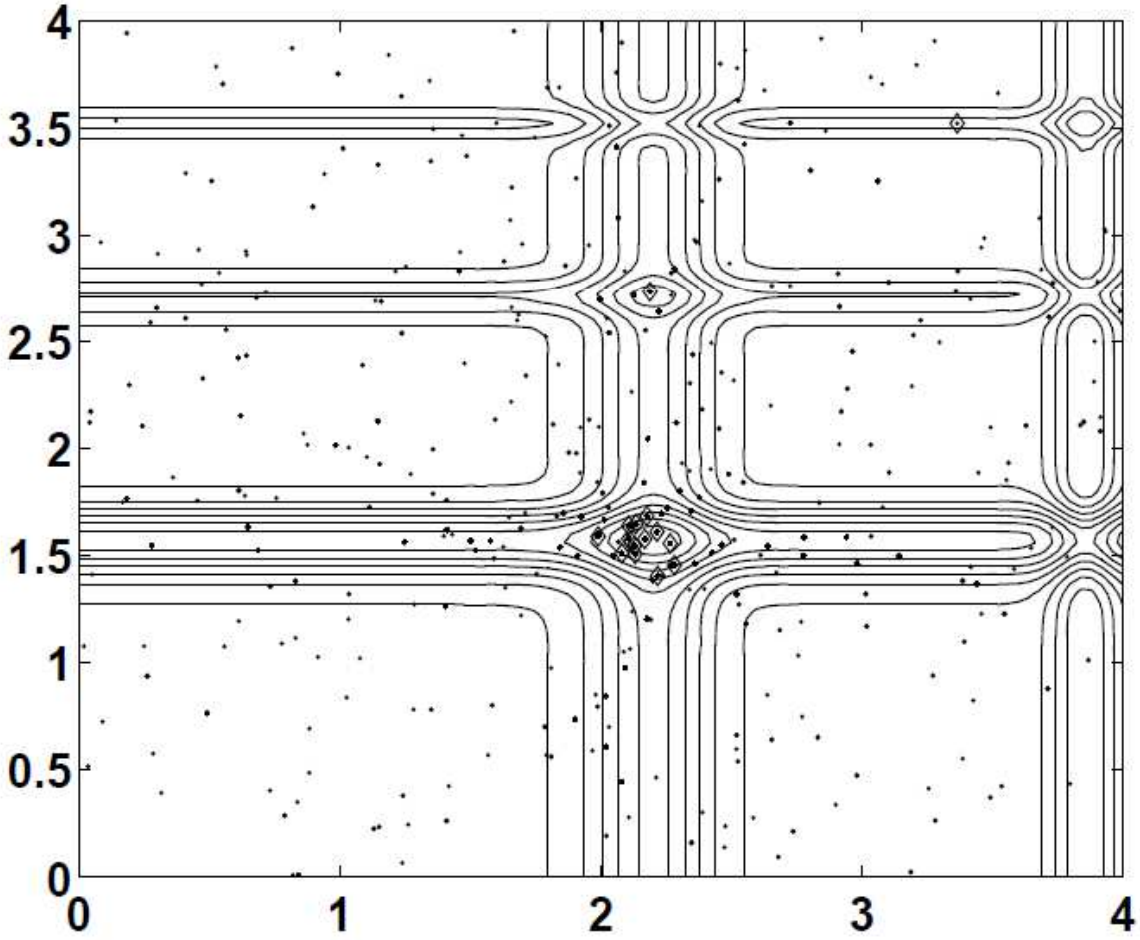
Przykładowy problem szukania

- Parametry symulacji CS:
  - n=15 gniazd
  - α= 1
  - p_a = 0.25

- W wielu symulacjach zwykle n $\in$ [15, 50]

Przykładowy problem szukania

- Rozkład rozwiązań na płaszczyźnie poszukiwań



Przykładowy problem szukania

- CS może znaleźć wszystkie optima równocześnie jeżeli liczba gniazd jest znacznie większa niż liczba lokalnych optimów

- Ta zaleta może staje się bardziej znacząca dla zadań optymalizacji wielokryterialnej i multimodalnej

Zmodyfikowany algorytm CS

- Poprawienie zbieżności CS (Walton et al.)
- Modyfikacja wprowadza dodatkowy krok o przekazywaniu informacji pomiędzy najlepszymi rozwiązaniami (jajami kukułki)
- Zmodyfikowany CS (MCS) przewyższa standardowy CS pod względem szybkości zbieżności (wynik przeprowadzonych badań dla standardowych zadań optymalizacji)

Zmodyfikowany algorytm CS

- Jajo kukułki bardzo podobne do jaja ptaka gospodarza – mniejsze prawdopodobieństwo wykrycia pasożyta
- Zatem jakość rozwiązania powinna być proporcjonalna do różnicy pomiędzy jajem kukułki a jajem ptaka gospodarza
- CS i MCS używają losowej wielkości kroku do generowania nowych rozwiązań

Zastosowania algorytmu CS

- Przykładowo:
  - projektowanie sprężyn i elementów spawanych
  - ploblemy planowania i harmonogramowania zadań
  - problemy fuzji danych w sieciach czujników bezprzewodowych
  - problem plecakowy
  - scheduling problem
  - projektowanie systemów wbudowanych
  - trenowanie sztucznych sieci neuronowych

Poszukiwania systemem naładowanych cząstek (Charged System Search - CSS)

- CSS - algorytm optymalizacji oparty na prawach fizyki i mechaniki
- CSS stosuje prawo Culomba i Gaussa w elektrostatyce i prawa Newtona z mechaniki
- CSS jest wieloagentowym podejściem, w tórym agentem jest naładowana cząstka (Charged Particle - CP)
- CP mogą wpływać na siebie na podstawie wartości przystosowania oraz odległości

Charged System Search (CSS)

- Wielkość wynikowej siły jest określana poprzez zastosowanie praw z elektrostatyki
- Wielkość poruszania się określana jest z wykorzystaniem praw mechaniki Newtona
- CSS ma zastosowanie do wielu problemów optymalizacji:
  - nie gładkich powierzchni
  - nie wypukłych przestrzeni

Charged System Search (CSS)

- Algorytm pozwala na umiejętny balans pomiędzy paradygmatami:
  - eksploracji
  - eksploatacji

- CSS - równocześnie skuteczna metoda optymalizacji globalnej jak i lokalnej

Podstawy CSS

- Ładunek elektryczny wytwarza pole elektryczne w otaczającej przestrzeni, które wywiera nacisk na inne naładowane elektrycznie obiekty
- Pole elektryczne pewnej naładowanej cząstki opisane jest przez prawo Culomba
- Siła elektryczna między dwoma małymi kulkami jest proporcjonalna do odwrotności kwadratu ich odległości

Podstawy CSS

Równoważnia Culomba, wykorzystana do ustalenia siły elektrycznej między dwoma kuklami

Podstawy CSS

- Siła elektryczna pomiędzy dwoma naładowanymi cząstkami:
  - odwrotnie proporcjonalna do kwadratu odległości pomiędzy nimi
  - proporcjonalna do ładunków q_i i q_j
  - przyciąganie ładunków o różnych znakach
  - odpychanie ładunków o jednakowych znakach

Podstawy CSS

- Prawo Coulomba przewiduje wielkość siły elektrycznej pomiędzy dwoma ładunkami

$$F_{ij} = k_e \frac{q_i q_j}{r_{ij}^2}$$

gdzie

k_e - stała Culomba

r_{ij} - odległość pomiędzy ładunkami

Podstawy CSS

- Rozmażmy izolowana kulę o promieniu *a,* która posiada jednorodną gęstość ładunku q_i o znaku dodatnim

- Natężenie pola elektycznego E_{ij} w punkcie poza kulą

$$E_{ij} = k_e \frac{q_i}{r_{ij}^2}$$

Podstawy CSS

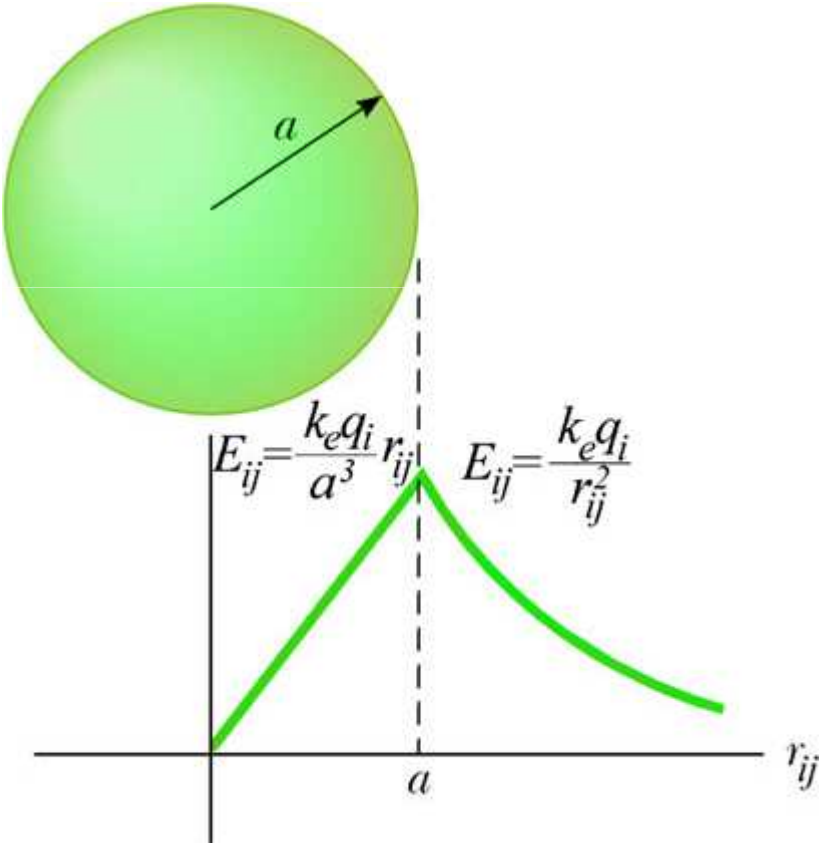
- Natężenie pola elektrycznego wewnątrz kuli można wyznaczyć na podstawie prawa Gaussa

$$E_{ij} = k_e \frac{q_i}{a^3} r_{ij}$$

- Jeżeli $r_{ij} \to 0$ to $E_{ij} \to 0$
- Pole elektryczne zmienia się wewnątrz kuli w sposób liniowy

Podstawy CSS

- Natężenie pola wewnątrz kuli jest identyczne jak dla ładunku q_i w odległości $r_{ij} = 0$
- Natężenie pola elektrycznego pokrywa się dla przypadku gdy $r_{ij} = a$



Podstawy CSS

- Natężenie pola elektrycznego dla grupy ładunków

$$E_j = \sum_{i=1, i\neq j}^{N} E_{ij} \qquad E_{ij} = \begin{cases} \dfrac{k_e q_i}{a^3} r_{ij}, & if\ r_{ij} < a \\ \dfrac{k_e q_i}{r_{ij}^2}, & if\ r_{ij} \geq a \end{cases}$$

gdzie

N – liczba naładowanych cząstek

Podstawy CSS

- Wartości i kierunku wynikowej siły na ładunek q_j w punkcie r_j spowodowanej ładunkiem q_i w punkcie r_i , można wyrazić następująco

$$\boldsymbol{F}_{ij} = E_{ij} q_j \frac{\boldsymbol{r}_i - \boldsymbol{r}_j}{\left\| \boldsymbol{r}_i - \boldsymbol{r}_j \right\|}$$

Podstawy CSS

- Dla wielu naładowanych cząstek, można zapisać

$$\boldsymbol{F}_j = k_e q_j \sum_{i, i \neq j} \left(\frac{q_i}{a^3} r_{ij} \cdot \alpha + \frac{q_i}{r_{ij}^2} \cdot \beta \right) \frac{\boldsymbol{r}_i - \boldsymbol{r}_j}{\left\| \boldsymbol{r}_i - \boldsymbol{r}_j \right\|}$$

gdzie

$$\begin{cases} \alpha = 1, \beta = 0 \Leftrightarrow r_{ij} < a \\ \alpha = 0, \beta = 1 \Leftrightarrow r_{ij} \geq a \end{cases}$$

Podstawy CSS

- Poruszający się obiekt jest opisany jako cząstka niezależnie od jego wielkości
- Cząstka jest punktem masy o nieskończenie małym rozmiarze
- Ruch cząstek jest do końca znany – potrafimy jednoznacznie ocenić miejsce i czas cząstki
- Przemieszczenie cząstek jest zdefiniowane jako zmiana pozycji

Podstawy CSS

- Przemieszczenie – naładowana cząstka zmienia pozycję z r_{old} na pozycję końcową r_{new}

$$\Delta \boldsymbol{r} = \boldsymbol{r}_{new} - \boldsymbol{r}_{old}$$

- Prędkość – nachylenie stycznej do pozycji naładowanej cząstki

$$\boldsymbol{v} = \frac{\boldsymbol{r}_{new} - \boldsymbol{r}_{old}}{t_{new} - t_{old}} = \frac{\Delta \boldsymbol{r}}{\Delta t}$$

Podstawy CSS

- Przyspieszenie – zmiana prędkości cząstki w czasie

$$\boldsymbol{a} = \frac{\boldsymbol{v}_{new} - \boldsymbol{v}_{old}}{\Delta t} = \frac{\Delta \boldsymbol{v}}{\Delta t}$$

- Przesunięcie dowolnego obiektu w funkcji czasu

$$\boldsymbol{r}_{new} = \frac{1}{2}\boldsymbol{a} \cdot \Delta t^2 + \boldsymbol{v}_{old} \cdot \Delta t + \boldsymbol{r}_{old}$$

Podstawy CSS

- Drugie prawo Newtona wyjaśnia, co dzieje się z obiektem, na który działa niezerowa wypadkowa siła

$$\boldsymbol{F} = m \cdot \boldsymbol{a}$$

- Przemieszczenie cząstki można zapisać jako

$$\boldsymbol{r}_{new} = \frac{1}{2}\frac{\boldsymbol{F}}{m} \cdot \Delta t^2 + \boldsymbol{v}_{old} \cdot \Delta t + \boldsymbol{r}_{old}$$

Algorytm CSS

- Każde rozwiązanie $\boldsymbol{x}_i$ zawierające zmienne decyzyjne reprezentuje naładowaną cząstkę
- Naładowana cząstka (agent) ma wpływ na pole elektryczne innych cząstek (agentów)
- Wielkość wynikowej siły jest określana z wykorzystaniem praw elektrostatyki

Algorytm CSS

- Wielkość przemieszczenia jest określana za pomocą praw mechaniki Newtona
- Agenci lepsi wywierają mocniejszy wpływ na agentów słabszych
- Wartość ładunku jest definiowana w oparciu o wartość funkcji celu $f(\boldsymbol{x}_i)$

Algorytm CSS (reguła 1)

- CSS przetwarza grupę naładowanych cząstek (CP)
- CP posiada ładunek q_i i wytwarza pole elektryczne wokół siebie
- Wielkość ładunku dla i-tego agenta

$$q_i = \frac{f(\boldsymbol{x}_i) - f_{worst}}{f_{best} - f_{worst}}, \quad i = 1,2,\ldots,N$$

Algorytm CSS (reguła 1)

- Odległość $\boldsymbol{r}_{ij}$ pomiędzy dwoma naładowanymi cząstkami

$$r_{ij} = \frac{\left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|}{\left\| \frac{1}{2}(\boldsymbol{x}_i + \boldsymbol{x}_j) - \boldsymbol{x}_{best} \right\| + \varepsilon}$$

gdzie

$\boldsymbol{x}_i$, $\boldsymbol{x}_j$ – pozycje *i-t*ego i *j*-ego agenta

$\boldsymbol{x}_{best}$ - pozycja najlepszego agenta

ε - mała dodatnia liczba – uniknięcie osobliwości

Algorytm CSS (reguła 2)

- Początkowa pozycja *j*-tego CP

$$x_{k,j}^{(0)} = x_{k,\min} + r(x_{k,\max} - x_{k,\min}), \quad k = 1,2,\ldots,n$$

gdzie

$x_{k,min}$, $x_{k,max}$ - dolna/górna wartość zakresu *k*-tej zmiennej

r - liczba pseudolosowa z zakresu [0,1]

n - liczba zmiennych decyzyjnych

- Początkowa prędkość agentów $v_{i,j}^{(0)} = 0$

Algorytm CSS (reguła 3)

- Trzy możliwe warunki związane z rodzajem siły przyciągania:
  1. Każda agent CP może wpływać na innych (słaby CP może wpływać na lepszego i odwrotnie ($p_{ij} = 1$))
  2. Agent CP może wpływać na innych jeżeli jego wielkość ładunku jest lepsza od innych agentów

$$p_{ij} = \begin{cases} 1, & f(\boldsymbol{x}_j) > f(\boldsymbol{x}_i) \\ 0, & else \end{cases}$$

Algorytm CSS (reguła 3)

3. Wszystkie lepsze CP mogą przyciągać słabsze CP i tylko część słabych agentów wpływa na lepszych, biorąc pod uwagę następujące funkcje prawdopodobieństwa

$$p_{ij} = \begin{cases} 1, & \dfrac{f(\boldsymbol{x}_i) - f_{best}}{f(\boldsymbol{x}_j) - f(\boldsymbol{x}_i)} > r \vee f(\boldsymbol{x}_j) > f(\boldsymbol{x}_i) \\ 0, & else \end{cases}$$

Algorytm CSS (reguła 3)

- Lepszy agent wpływa na słabego – przeprowadzana eksploatacja
- Słabszy agent wpływa na lepszego - realizowana eksploracja
- Kiedy dany agent porusza się w kierunku dobrego agenta – poprawa wydajności CP (charakter samoadaptacji)

Algorytm CSS (reguła 3)

- Przesunięcie lepszych CP w kierunku słabych może powodować utratę poprzedniego dobrego rozwiązania lub zwiększenie kosztów obliczeniowych

- Zastosowana pamięć, która zachowuje najlepsze dotychczasowe rozwiązania

Algorytm CSS (reguła 4)

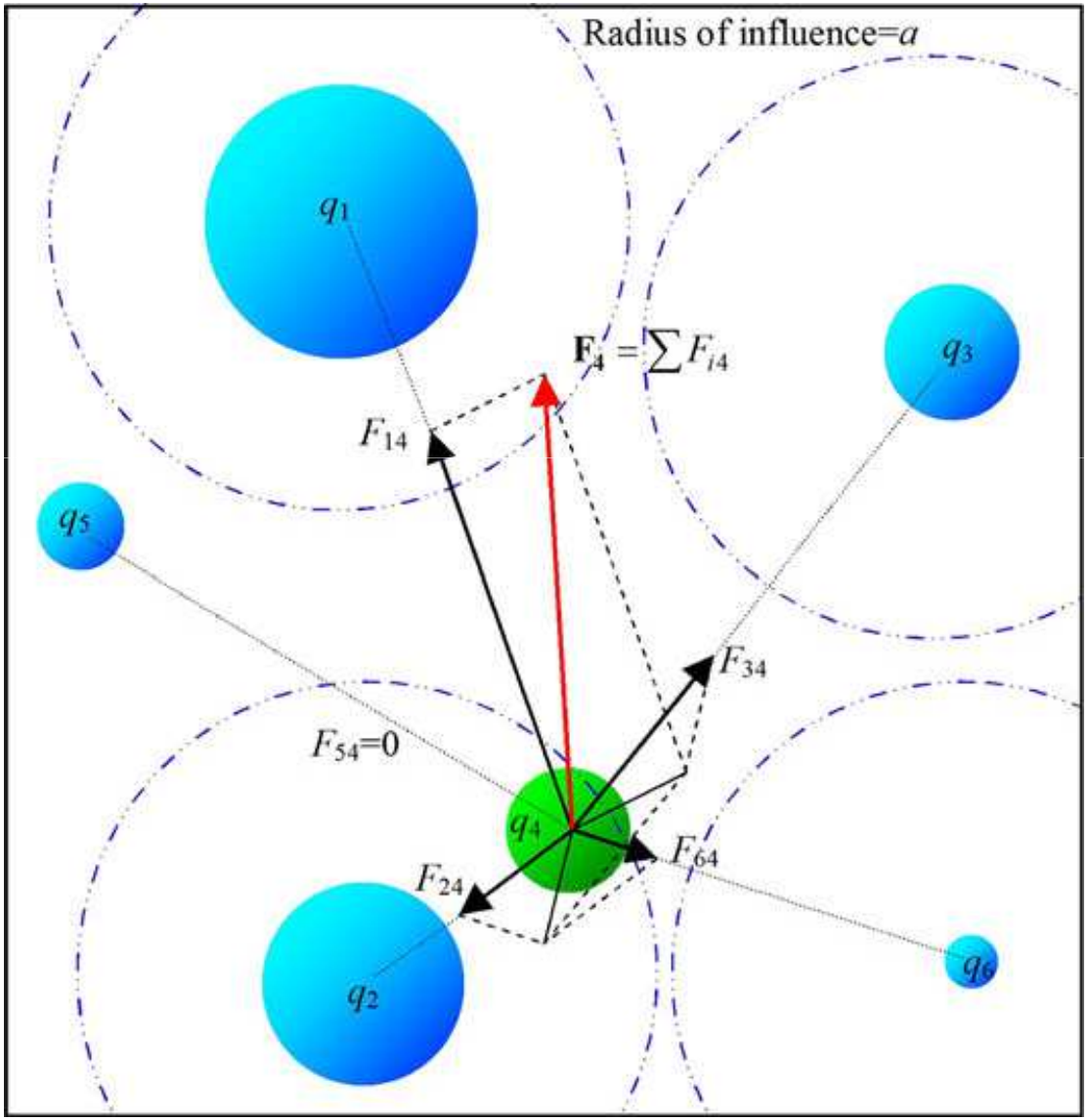
- Wypadkowa wartość siły działająca na *j*-tą CP

$$\boldsymbol{F}_j = k_e q_j \sum_{i,i\neq j} \left(\frac{q_i}{a^3} r_{ij} \cdot \alpha + \frac{q_i}{r_{ij}^2} \cdot \beta \right) p_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j)$$

$$\begin{cases} j = 1,2,\ldots,N \\ \alpha = 1, \beta = 0 \Leftrightarrow r_{ij} < a \\ \alpha = 0, \beta = 1 \Leftrightarrow r_{ij} \geq a \end{cases}$$

Algorytm CSS (reguła 4)

- Wypadkowa siła elektryczna działająca na cząstkę



Algorytm CSS (reguła 4)

- Każdy agent CP jest traktowany jako naładowana kula o jednolitej gęstość ładunku i promieniu *a*
- Zwykle promień jest równy wartości 1
- Odpowiednią wartość promienia należy określić biorąc pod uwagę wielkość przestrzeni poszukiwań

$$a = 0.1 \cdot \max\left\{\left|x_{i,\max} - x_{i,\min}\right|\right\}$$

Algorytm CSS (reguła 4)

- W pierwszych iteracjach CSS, agenci są w znacznie większych odległościach od siebie - wypadkową siła działająca na CP jest odwrotnie proporcjonalna do kwadratu odległości między agentami

- Stąd poszukiwania w tym stanie algorytmu są najbardziej obiecujące (eksploracja)

Algorytm CSS (reguła 4)

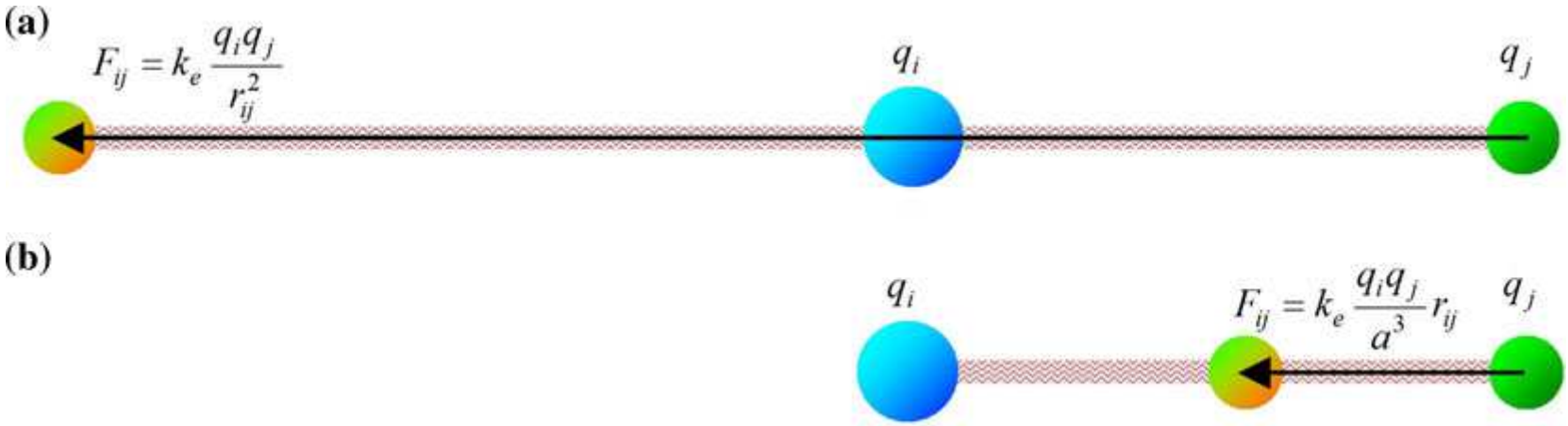
- Konieczne jest stopniowe zwiększenie eksploatacji i zmniejszanie eksploracji algorytmu
- Po pewnej liczbie cykli algorytmu CSS odległości pomiędzy agentami mogą być bardzo małe
- Wtedy siła staje się proporcjonalna do odległości cząstek zamiast być odwrotnie proporcjonalna do kwadratu odległości

Algorytm CSS (reguła 4)

- Kooperacja pomiędzy dwoma równaniami

(a) $F_{ij} \propto 1/r_{ij}^2$

(b) $F_{ij} \propto r_{ij}\,, \quad if\ r_{ij} < a$



Algorytm CSS (reguła 4)

- Parametr *a* oddziaływuje na globalną i lokalną fazę poszukiwań
- Większość agentów jest gromadzona w przestrzeni o promieniu *a* – globalne poszukiwania są zatrzymane i proces optymalizacji kontynuowany jest poprzez lokalne poszukiwania
- Ponadto używając tych reguł można sterować pomiędzy eksploracją a eksploatacją

Algorytm CSS (reguła 4)

- Należy zauważyć, że rozważana reguła obliczania siły w oparciu o promień *a* wykorzystuje pewien mechanizm konkurencji
- Ponieważ siła jest proporcjonalna do wielkości ładunku lepiej przystosowane cząstki (duże q_i) może stworzyć silniejsza siłę przyciąga
- Skłonność do działań w kierunku rozwiązań dobrych CP jest większa niż do słabszych cząstek

Algorytm CSS (reguła 5)

- Nowa pozycja i prędkość cząstki CP

$$\boldsymbol{x}_j^{new} = \rho_{j1} \cdot k_a \frac{\boldsymbol{F}_j}{m_j} \cdot \Delta t^2 + \rho_{j2} \cdot k_v \cdot \boldsymbol{v}_j^{old} \cdot \Delta t + \boldsymbol{x}_j^{old}$$

$$\boldsymbol{v}_j^{new} = \left(\boldsymbol{x}_j^{new} - \boldsymbol{x}_j^{old} \right) / \Delta t$$

k_a, k_v - współczynniki wzmocnienia przyspieszenia i prędkości

ρ_{j1}, ρ_{j2} - liczby losowe (0,1)

m_j - masa j-tej czastki CP (równa q_j)

Algorytm CSS (reguła 5)

- Efekt poprzedniej prędkości i wynikowej siły działającej na CP może być zmniejszany lub zwiększany za pomocą k_v i k_a odpowiednio
- Nadmierne wyszukiwania na początkowym cyklu CSS, może poprawić zdolność poszukiwania
- Jednakże, musi być stopniowo zmniejszany (jak opisano powyżej)

Algorytm CSS (reguła 5)

- Parametr k_a związany z siłami przyciągania,
- Wybierając dużą wartość tego parametru możemy spowodować szybką zbieżność
- Wybierając małą wartości możemy zwiększyć czas obliczeń
- k_a steruje eksploatacją
- Należy wybrać funkcje monotonicznie rosnąca w celu poprawnego zachowania algorytmu

Algorytm CSS (reguła 5)

- Kierunek poprzedniej prędkości cząstki CP nie musi być koniecznie taki sam jak wynikowej siły
- Współczynnik wzmocnienia prędkości k_v steruje procesem eksploracji
- Funkcja opisującą ten współczynnik powinna mieć charakter monotonicznie malejący

Algorytm CSS (reguła 5)

- Definicja k_v i k_a

$$k_{v,a} = 0.5(1 \pm t / t_{\max})$$

gdzie

t - aktualny numer iteracji

t_{max} – maksymalna liczba iteracji

- k_v zmniejsza się liniowo do zera
- k_a zwiększa się w kolejnych iteracjach
- Równowaga między szukaniem i szybkim tempem zbieżności jest zachowana

Algorytm CSS (reguła 5)

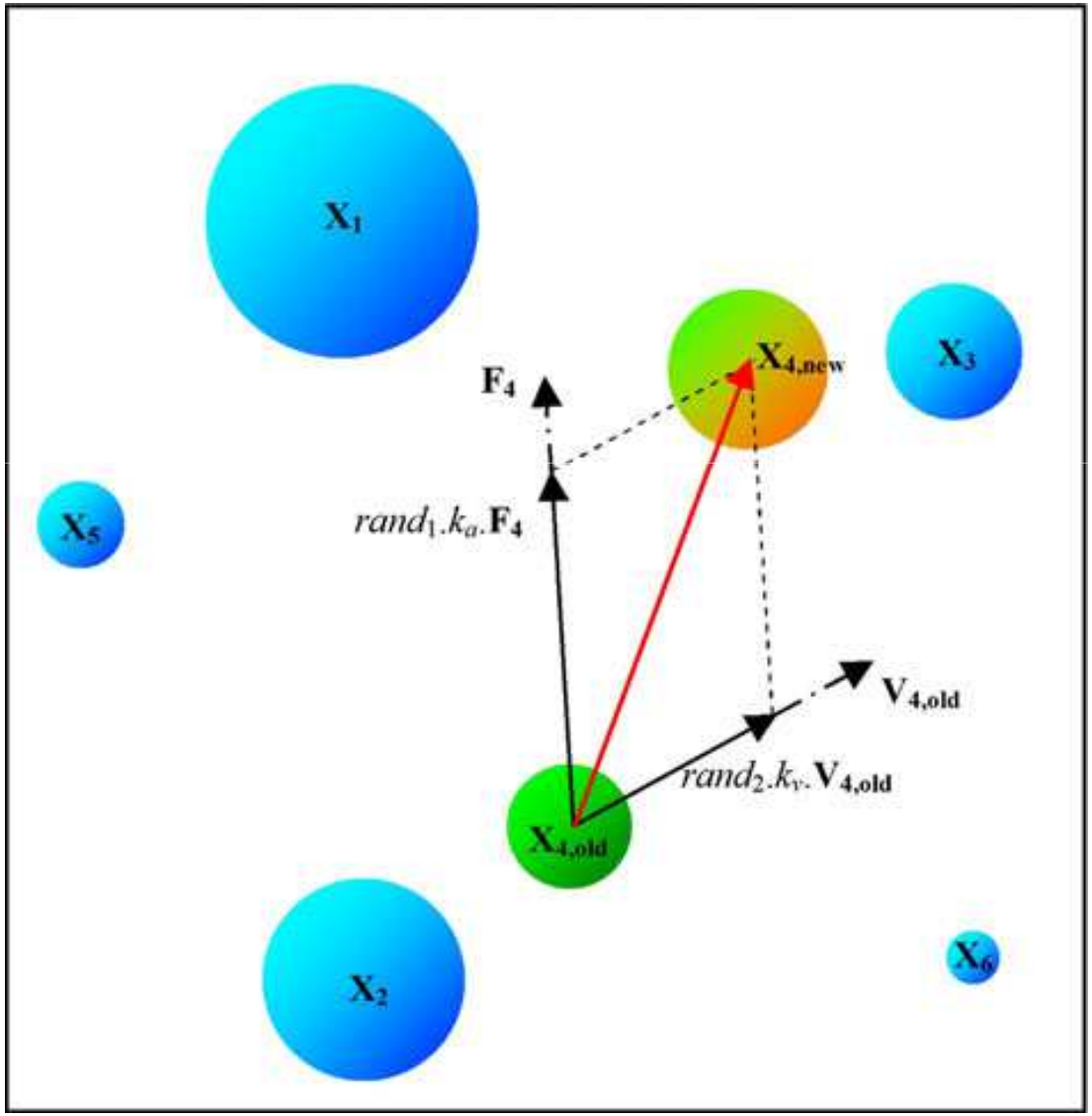
- Nowa pozycja agenta

$$\boldsymbol{x}_j^{new} = 0.5\rho_{j1} \cdot (1 + t/t_{\max}) \cdot \sum_{i,i\neq j} \left(\frac{q_i}{a^3} r_{ij} \cdot \alpha + \frac{q_i}{r_{ij}^2} \cdot \beta \right) p_{ij} \left(\boldsymbol{x}_i - \boldsymbol{x}_j \right) + 0.5\rho_{j2} \cdot (1 - t/t_{\max}) \cdot \boldsymbol{v}_j^{old} + \boldsymbol{x}_j^{old}$$

$$\boldsymbol{v}_j^{new} = \boldsymbol{x}_j^{new} - \boldsymbol{x}_j^{old}$$

Algorytm CSS (reguła 5)

- Przesunięcie cząstki naładowanej do nowej pozycji



Algorytm CSS (reguła 6)

- Zastosowanie pamięci, która zachowuje najlepsze rozwiązania i związane z nimi wartości funkcji celu
- Zwiększa to wydajność algorytm bez zwiększania kosztów obliczeniowych

- Pamięć ładunków (Charged Memory CM) jest wykorzystywana do zapisywania najlepszych do tej pory znalezionych rozwiązań

Algorytm CSS (reguła 6)

- Rozmiar CM może wynosić np. N/4

- Wykorzystanie pamięci CM do prowadzenia bieżących poszukiwań i wpływania na aktualne naładowane cząstki

- Zakłada się, że taka sama liczba aktualnie cząstek najgorszych nie może przyciągnąć innych

Algorytm CSS (reguła 7)

- Istnieją dwa główne problemy w odniesieniu do meta- heurystycznych algorytmów:
  - Jak zapewnić równowagę między eksploracją i eksploatacją na początku, w trakcie i końcowych krokach wyszukiwania?
  - Jak radzić sobie z agentem naruszającym ograniczenia na zmienne decyzyjne?

Algorytm CSS (reguła 7)

- Wykorzystanie najbliższej wartości granicznej dla naruszonej zmiennej
- Powrót do swojej poprzedniej pozycji
- Zmniejszenie maksymalnej wartość prędkości, aby umożliwić mniejszej liczbie cząstek naruszenia ograniczeń
- Proste metody, ale nie wystarczająco skuteczne i mogą prowadzić do zmniejszenia eksploracji przestrzeni poszukiwań

Algorytm CSS (reguła 7)

- Podejście z harmonicznym wspomaganiem poszukiwań

$$x_{i,j} = \begin{cases} z.p.CMCR & \Rightarrow \text{ wybierz nową wartość dla zmiennej z CM} \\ z.p.\ (1 - PAR) & \Rightarrow \text{ pozostaw bez zmian} \\ z.p.\ PAR & \Rightarrow \text{ wybierz sąsiednią wartość} \\ z.p.(1 - CMCR) & \Rightarrow \text{ wybierz nową wartość losowo} \end{cases}$$

"z.p." - "z prawdopodobieństwem"

$x_{i,j}$ - i-ta współrzędna cząstki CP j

CMCR$\in$ [0,1] (Charged Memory Considering Rate) - wskaźnik wyboru wartości w nowy wektor na podstawie historii przechowywanych wartości w CM

(1 − CMCR) - wskaźnik losowego wyboru danej wartości z jej zakresu

(1−PAR) - wskaźnik „nic nie rób" oraz PAR ustawione na szybkość wyboru wartości z sąsiedztwa z najlepszego sąsiedztwa CP

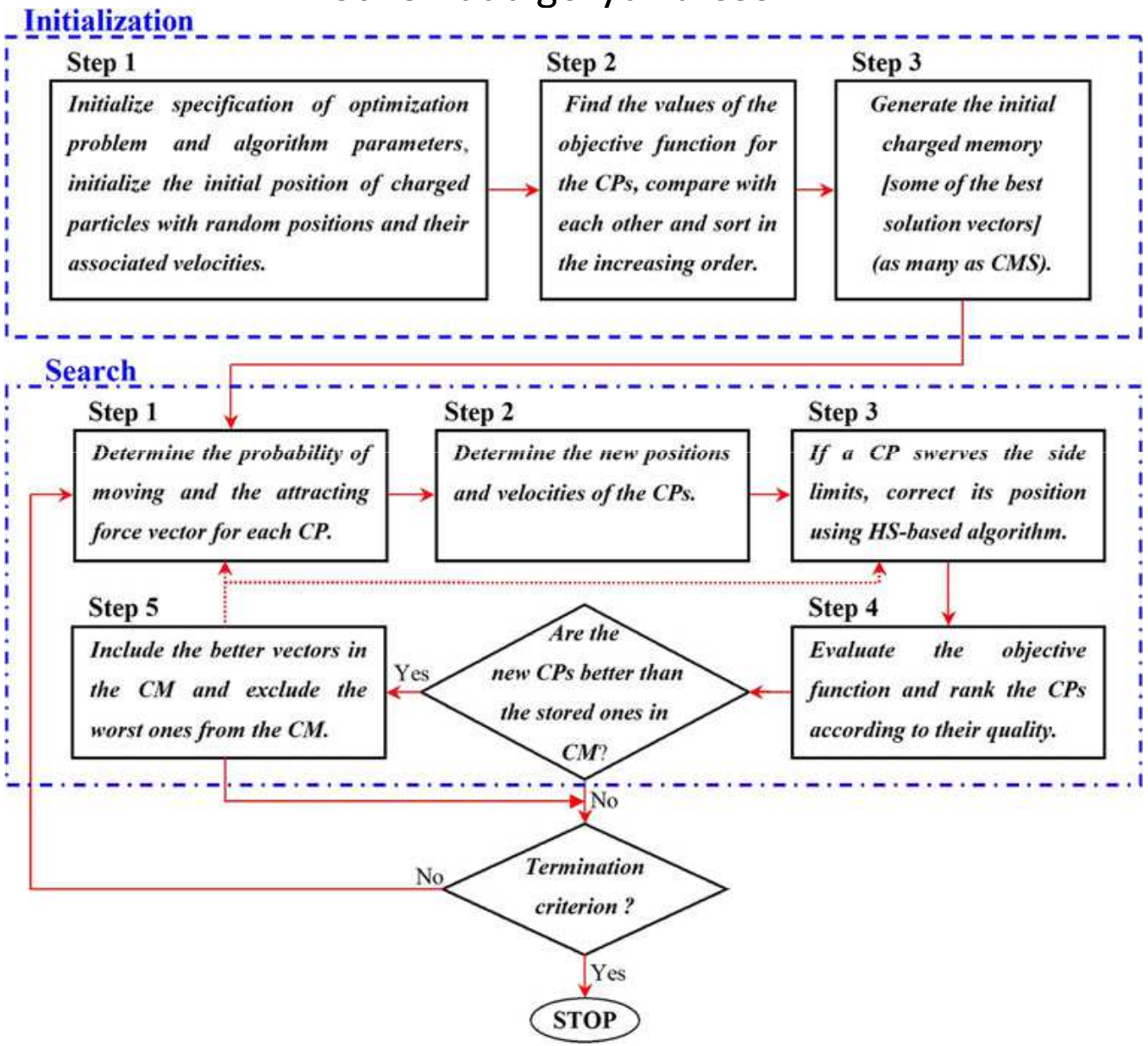
Algorytm CSS (reguła 8)

- Kryteria zatrzymania algorytmu CSS:
  - maksymalna liczba iteracji: proces optymalizacji jest zakończony po ustalonej liczby iteracji
  - liczba iteracji bez poprawy: proces optymalizacji jest zakończone po pewnej ustalonej liczby iteracji bez poprawy

Algorytm CSS (reguła 8)

- Kryteria zatrzymania CSS
  - minimum błędu funkcji celu: Różnica między wartościami z najlepszych funkcji celu i optymalnego globalnego położenia jest mniejszy niż wcześniej ustalonego progu przewidywanych
  - różnica między najlepszym i najgorszym CP: proces optymalizacji jest zatrzymany, jeśli różnica między obiektywnych wartości najlepsze i najgorsze CP staje się mniej niż określoną
  - Dokładnością pomiarową m distance of CPs: the maximum distance between CPs is less than a pre-fixed value

Schemat algorytmu CSS



Literatura

1. X.-S. Yang; S. Deb (December 2009). "Cuckoo search via Lévy flights". World Congress on Nature & Biologically Inspired Computing (NaBIC 2009). IEEE Publications. pp. 210–214
2. Kaveh A, Talatahari S. A novel heuristic optimization method: charged system search, *Acta Mechanica, 2010, DOI: 10.1007/s00707-009-0270-4.*.
3. http://en.wikipedia.org/wiki/Levy_distribution
4. http://en.wikipedia.org/wiki/Cuckoo_search
5. http://en.wikipedia.org/wiki/Levy_flights