Particle Swarm Optimization

Budi Santosa
Dosen Teknik Industri ITS Surabaya
Email: budi_s@ie.its.ac.id

Tutorial ini disadur dari buku: Budi Santosa dan Paul Willy, Metoda Metaheuristik, Konsep dan Implementasi, Graha Ilmu, Surabaya, 2011.

1. Pendahuluan

Particle Swarm Optimization (PSO) didasarkan pada perilaku sekawanan burung atau ikan. Algoritma PSO meniru perilaku sosial organisme ini. Perilaku sosial terdiri dari tindakan individu dan pengaruh dari individu-individu lain dalam suatu kelompok. Kata partikel menunjukkan, misalnya, seekor burung dalam kawanan burung. Setiap individu atau partikel berperilaku dengan cara menggunakan *kecerdasannya* (intelligence) sendiri dan juga dipengaruhi perilaku kelompok kolektifnya. Dengan demikian, jika satu partikel atau seekor burung menemukan jalan yang tepat atau pendek menuju ke sumber makanan, sisa kelompok yang lain juga akan dapat segera mengikuti jalan tersebut meskipun lokasi mereka jauh di kelompok tersebut.

> Perilaku seekor hewan dalam kawanan (swarm) dipengaruhi perilaku individu dan juga kelompoknya.

Dalam Particle Swarm Optimization (PSO), kawanan diasumsikan mempunyai ukuran tertentu dengan setiap partikel posisi awalnya terletak di suatu lokasi yang acak dalam ruang multidimensi. Setiap partikel diasumsikan memiliki dua karakteristik: posisi dan kecepatan. Setiap partikel bergerak dalam ruang/space tertentu dan mengingat posisi terbaik yang pernah dilalui atau ditemukan terhadap sumber makanan atau nilai fungsi objektif. Setiap partikel menyampaikan informasi atau posisi terbaiknya kepada partikel yang lain dan menyesuaikan posisi dan kecepatan masing-masing berdasarkan informasi yang diterima mengenai posisi tersebut. Sebagai contoh, misalnya perilaku burung-burung dalam dalam kawanan

burung. Meskipun setiap burung mempunyai keterbatasan dalam hal kecerdasan, biasanya ia akan mengikuti kebiasaan (rule) seperti berikut :

1. Seekor burung tidak berada terlalu dekat dengan burung yang lain
2. Burung tersebut akan mengarahkan terbangnya ke arah rata-rata keseluruhan burung

3. Akan memposisikan diri dengan rata-rata posisi burung yang lain dengan menjaga sehingga jarak antar burung dalam kawanan itu tidak terlalu jauh

Dengan demikian perilaku kawanan burung akan didasarkan pada kombinasi dari 3 faktor simpel berikut:

1. Kohesi - terbang bersama
2. Separasi - jangan terlalu dekat
3. Penyesuaian (alignment) - mengikuti arah bersama

Jadi PSO dikembangkan dengan berdasarkan pada model berikut:

1. Ketika seekor burung mendekati target atau makanan (atau bisa minimum atau maximum suatu fungsi tujuan) secara cepat mengirim informasi kepada burung-burung yang lain dalam kawanan tertentu
2. Burung yang lain akan mengikuti arah menuju ke makanan tetapi tidak secara langsung
3. Ada komponen yang tergantung pada pikiran setiap burung, yaitu memorinya tentang apa yang sudah dilewati pada waktu sebelumnya.

Pada algoritma PSO ini, pencarian solusi dilakukan oleh suatu populasi yang terdiri dari beberapa partikel. Populasi dibangkitkan secara randomdengan batasan nilai terkecil dan terbesar. Setiap partikel merepresentasikan posisi atau solusi dari permasalahan yang dihadapi. Setiap partikel melakukan pencarian solusi yang optimal dengan melintasi ruang pencarian (search space). Hal ini dilakukan dengan cara setiap partikel melakukan penyesuaian terhadap posisi terbaik dari partikel tersebut (local best) dan penyesuaian terhadap posisi partikel terbaik dari seluruh kawanan (global best) selama melintasi ruang pencarian. Jadi, penyebaran pengalaman atau informasi terjadi di dalam partikel itu sendiri dan antara suatu partikel dengan partikel terbaik dari seluruh kawanan selama proses pencarian solusi. Setelah itu, dilakukan proses pencarian untuk mencari posisi terbaik setiap partikel dalam sejumlah iterasi tertentu sampai didapatkan posisi yang relatif

steady atau mencapai batas iterasi yang telah ditetapkan. Pada setiap iterasi, setiap solusi yang direpresentasikan oleh posisi partikel, dievaluasi performansinya dengan cara memasukkan solusi tersebut kedalam *fitness function*.

Setiap partikel diperlakukan seperti sebuah titik pada suatu dimensi ruang tertentu. Kemudian terdapat dua faktor yang memberikan karakter terhadap status partikel pada ruang pencarian yaitu posisi partikel dan kecepatan partikel [Kennedy and Eberhart, 1995].

Setiap partikel diperlakukan seperti sebuah titik pada suatu dimensi ruang tertentu. Kemudian terdapat dua faktor yang memberikan karakter terhadap status partikel pada ruang pencarian yaitu posisi partikel dan kecepatan partikel

Berikut ini merupakan formulasi matematika yang menggambarkan posisi dan kecepatan partikel pada suatu dimensi ruang tertentu :

$$X_i(t) = x_{i1}(t), x_{i2}(t), \dots, x_{iN}(t) \quad (1)$$

$$V_i(t) = v_{i1}(t), v_{i2}(t), \dots, v_{iN}(t) \quad (2)$$

dimana
X = posisi partikel
V = kecepatan partikel
i = indeks partikel
t = iterasi ke-t
N = ukuran dimensi ruang
Berikut ini merupakan model matematika yang menggambarkan mekanisme updating status partikel Kennedy and Eberhart [1995]:

$$V_i(t) = V_i(t-1) + c_1 r_1 \left(X_i^L - X_i(t-1)\right) + c_2 r_2 \left(X^G - X_i(t-1)\right) \quad (3)$$

$$X_i(t) = V_i(t) + X_i(t-1) \quad (4)$$

dimana
$X_i^L = x_{i1}^L, x_{i2}^L, \dots, x_{iN}^L$ merepresentasikan local best dari partikel ke-i. Sedangkan $X^G = x_{i1}^G, x_{i2}^G, \dots, x_{iN}^G$ merepresentasikan global best dari seluruh kawanan. Sedangkan c_1 dan c_2 adalah suatu konstanta yang bernilai positif yang biasanya disebut sebagai *learning factor*. Kemudian r_1 dan r_2 adalah suatu bilangan random yang bernilai antara 0 sampai 1. Persamaan (3) digunakan untuk menghitung kecepatan partikel yang baru berdasarkan kecepatan sebelumnya, jarak antara posisi saat ini dengan posisi terbaik partikel (local best), dan jarak antara posisi saat

ini dengan posisi terbaik kawanan (global best). Kemudian partikel terbang menuju posisi yang baru berdasarkan persamaan (4). Setelah algoritma PSO ini dijalankan dengan sejumlah iterasi tertentu hingga mencapai kriteria pemberhentian, maka akan didapatkan solusi yang terletak pada global best.

Perlu diingat bahwa posisi terbaik individu dan posisi terbaik kelompok perlu disimpan untuk keseluruhan iterasi

Model ini akan disimulasikan dalam ruang dengan dimensi tertentu dengan sejumlah iterasi sehingga di setiap iterasi, posisi partikel akan semakin mengarah ke target yang dituju (minimasi atau maksimasi fungsi). Ini dilakukan hingga maksimum iterasi dicapai atau bisa juga digunakan kriteria penghentian yang lain.

Algoritma PSO meliputi langkah berikut

- Bangkitkan posisi awal sejumlah partikel sekaligus kecepatan awalnya secara random.
- Evaluasi *fitness* dari masing-masing partikel berdasarkan posisinya.
- Tentukan partikel dengan *fitness* terbaik, dan tetapkan sebagai *Gbest*. Untuk setiap partikel, *Pbest* awal akan sama dengan posisi awal.

Ulangi langkah berikut sampai stopping criteria dipenuhi

1. Menggunakan *Pbest* dan *Gbest* yang ada, perbarui kecepatan setiap partikel menggunakan persamaan .3. Lalu dengan kecepatan baru yang didapat, perbarui posisi setiap partikel menggunakan persamaan .4.
2. Evaluasi *fitness* dari setiap partikel.
3. Tentukan partikel dengan *fitness* terbaik, dan tetapkan sebagai *Gbest*. Untuk setiap partikel, tentukan Pbest dengan membandingkan posisi sekarang dengan Pbest dari iterasi sebelumnya.
4. Cek stopping criteria. Jika dipenuhi, berhenti. Jika tidak, kembali ke 1

2. Implementasi PSO

Misalkan kita mempunyai fungsi berikut

$$\text{minimasi} \quad f(x) \tag{5}$$

dimana

$$x^{(B)} \leq x \leq x^{(A)}$$

dimana $x^{(B)}$ adalah batas bawah dan $x^{(A)}$ adalah batas atas dari x. Prosedur PSO dapat dijabarkan dengan langkah-langkah sebagai berikut Rao [2009]:

1. Asumsikan bahwa ukuran kelompok atau kawanan (jumlah partikel) adalah *N*. Untuk mengurangi jumlah evaluasi fungsi yang diperlukan untuk menemukan solusi, sebaiknya ukuran *N* tidak terlalu besar, tetapi juga tidak terlalu kecil,agar ada banyak kemungkinan posisi menuju solusi terbaik atau optimal. Jika terlalu kecil sedikit kemungkinan menemukan posisi partikel yang baik. Terlalu besar juga akan membuat perhitungan jadi panjang. Biasanya digunakan ukuran kawanan adalah 20 sampai 30 partikel.

2. Bangkitkan populasi awal x dengan rentang $x^{(B)}$ dan $x^{(A)}$ secara random sehingga didapat $x_1, x_2, \dots, x_N$. Partikel *j* dan kecepatannya pada iterasi *i* dinotasikan sebagai $x_j^{(i)}$dan $v_j^{(i)}$, sehingga partikel-partikel awal ini dinotasikan

   $$x_1(0), x_2(0), \dots, x_N(0)$$

   Vektor

   $$v_1(0), v_2(0), \dots, v_N(0)$$

   disebut partikel atau vektor koordinat dari partikel (seperti kromosom dalam algoritma genetika). Selanjutnya lakukan evaluasi nilai fungsi tujuan untuk setiap partikel dan nyatakan dengan

   $$f\big(x_1(0)\big), f\big(x_2(0)\big), \dots, f\big(x_N(0)\big).$$

3. Hitung kecepatan dari semua partikel. Semua partikel bergerak menuju titik optimal dengan suatu kecepatan tertentu. Awalnya semua kecepatan dari partikel diasumsikan sama dengan nol. Set iterasi *i* = 1.

4. Pada iterasi ke-i, temukan 2 parameter penting untuk setiap partikel *j* yaitu:

   (a) Nilai terbaik sejauh ini dari $x_j^{(i)}$ (koordinat partikel *j* pada iterasi *i*) dan nyatakan sebagai $P_{best,j}$, dengan nilai fungsi tujuan paling rendah (kasus minimasi), $f\big[x_j(i)\big]$, yang ditemui sebuah partikel *j* pada semua iterasi sebelumnya. Nilai terbaik untuk semua partikel $x_j^{(i)}$ yang ditemukan sampai iterasi ke-i, G_{best}, dengan nilai fungsi tujuan paling kecil/minimum diantara semua partikel untuk semua iterasi sebelumnya, $f\big[x_j(i)\big]$,

(b) Hitung kecepatan partikel *j* pada iterasi ke *i* dengan rumus sebagai berikut

$$v_j(i) = v_j(i-1) + c_1 r_1\big[P_{best,j} - x_j(i-1)\big] + c_2 r_2\big[G_{best} - x_j(i-1)\big], \quad j = 1, 2, \dots, N \tag{6}$$

dimana c_1 dan c_2 masing-masing adalah *learning rates* untuk kemampuan individu (cognitive) dan pengaruh sosial (kawanan), dan r_1 dan r_2 bilangan random yang

berdistribusi uniforml dalam interval 0 dan 1. Jadi parameter c_1 dan c_2 menunjukkan bobot dari memory (position) sebuah partikel terhadap memory (posisi) dari kelompok (swarm). Nilai dari c_1 dan c_2 biasanya adalah 2 sehingga perkalian c_1r_1 dan c_2r_2 memastikan bahwa partikel-partikel akan mendekati target sekitar setengah selisihnya.

(c) Hitung posisi atau koordinat partikel *j* pada iterasi ke-i dengan cara

$$x_j(i) = x_j(i-1) + v_j(i), \qquad j = 1, 2, \dots, N \tag{7}$$

Evaluasi nilai fungsi tujuan untuk setiap partikel dan nyatakan sebagai

$$f[x_1(i)], f[x_2(i)], \dots, f[x_N(i)]$$

.

5. Cek apakah solusi yang sekarang sudah konvergen. Jika posisi semua partikel menuju ke satu nilai yang sama, maka ini disebut konvergen. Jika belum konvergen maka langkah 4 diulang dengan memperbarui iterasi $i = i + 1$, dengan cara menghitung nilai baru dari $P_{best,j}$ dan G_{best}. Proses iterasi ini dilanjutkan sampai semua partikel menuju ke satu titik solusi yang sama. Biasanya akan ditentukan dengan kriteria penghentian (stopping criteria), misalnya jumlah selisih solusi sekarang dengan solusi sebelumnya sudah sangat kecil.

Contoh 1

Misalkan kita mempunyai persoalan optimasi dengan satu variabel sebagai berikut

$$\min f(x) = (100 - x)^2 \tag{8}$$

$$\text{dimana } 60 \le x \le 120$$

1. Tentukan jumlah partikel $N = 4$.

Tentukan populasi awal secara random, misalkan didapat

$x_1(0) = 80,$
$x_2(0) = 90,$
$x_3(0) = 110,$
$x_4(0) = 75.$

2. Evaluasi nilai fungsi tujuan untuk setiap partikel $x_j(0)$ untuk $j = 1, 2, 3, 4.$ dan nyatakan dengan

$f_1 = f(80) = 400,$
$f_2 = f(90) = 100,$
$f_3 = f(110) = 100,$
$f_4 = f(75) = 625,$

3. Tentukan kecepatan awal $v_1(0) = v_2(0) = v_3(0) = v_4(0) = 0$.
Tetapkan iterasi i = 1; Lalu ke langkah nomer 4.

4. Temukan
$P_{best,1}$ = 80,
$P_{best,2}$ = 90,
$P_{best,3}$ = 110,
$P_{best,4}$ = 75,
G_{best} = 90.
Hitung $v(j)$ dengan c_1 = c_2 = 1. Misalkan nilai random yang didapat, r_1 = 0.4, r_2 = 0.5, dengan rumus

$$V_j(i) = V_j(i-1) + c_1 r_1\left[P_{best,j} - x_j(i-1)\right] + c_2 r_2\left[G_{best,j} - x_j(i-1)\right]$$

diperoleh
$v_1(1) = 0 + \ 0.4(80 - 80) + 0.5(90 - 80) = 5$
$v_2(1) = 0 + 0.4(90 - 90) + 0.5(90 - 90) = 0$
$v_3(1) = 0 + 0.4(110 \ - \ 110) + 0.5(90 - 110) = -10$
$v_4(1) = 0 + 0.4(75 \ - \ 75) + 0.5(90 - 75) = 7.5$

Sedangkan untuk nilai x adalah
$v_1(1) = 80 + 5 = 85$
$x_2(1) = 90 + 0 = 90$
$x_3(1) = 110 - 10 = 100$
$x_4(1) = 75 + 7.5 = 82.5$

5. Evaluasi nilai fungsi tujuan sekarang pada partikel $x_j(1)$,

$f_1(1) = f(85) = 225$,
$f_2(1) = f(90) = 100$,
$f_3(1) = f(100) = 0$,
$f_4(1) = f(82.5) = 306.25$.
Sedangkan pada iterasi sebelumnya kita dapatkan
$f_1(0) = f(80) = 400$,
$f_2(0) = f(90) = 100$,
$f_3(0) = f(110) = 100$,
$f_4(0) = f(75) = 625$.
Nilai dari f dari iterasi sebelumnya tidak ada yang lebih baik sehingga P_{best} untuk masing-masing partikel sama dengan nilai x-nya. G_{best} = 100.

Cek apakah solusi x sudah konvergen, dimana nilai x saling dekat. Jika tidak, tingkatkan ke iterasi berikutnya i = 2. Lanjutkan ke langkah 4.

1. $P_{best,1}$ = 85,
   $P_{best,2}$ = 90,
   $P_{best,3}$ = 100,
   $P_{best,4}$ = 82.5,
   G_{best} = 100
   Hitung kecepatan baru dengan r_1 = 0.3 dan r_2 = 0.6 (ini hanya sekedar contoh untuk menjelaskan penghitungan, dalam implementasi angka ini dibangkitkan secara random).
   $v_1(2) = 5 + 0.3(85 - 85) + 0.6(100 - 85) = 14$
   $v_2(2) = 0 + 0.3(90 - 90) + 0.6(100 - 90) = 6.$
   $v_3(2) = -10 + 0.3100 - 100) + 0.6(100 - 100) = -10$
   $v4(2) = 7.5 + 0.3(82.5 - 82.5) + 0.6(100 - 82.5) = 18$
   Sedangkan untuk nilai x adalah
   $x_1(2) = 85 + 14 = 99$
   $x_2(2) = 90 + 6 = 96$
   $x_3(2) = 100 - 10 = 90$
   $x_4(2) = 82.5 + 18 = 100.5$

2. *Evaluasi* nilai fungsi tujuan sekarang pada partikel $x_j(2)$,
   $f_1(2) = f(99) = 1,$
   $f_2(2) = f(96) = 16,$
   $f_3(2) = f(90) = 100$
   $f_4(2) = f(100.5) = 0.25.$

   Jika dibandingkan dengan nilai *f* dari iterasi sebelumnya, ada nilai yang lebih baik dari nilai *f* sekarang yaitu $f_3(1) = 0$, sehingga P_{best} untuk partikel 3 sama dengan 100, dan G_{best} dicari dari $\min\{1, 16, 0, 0.25\} = 0$ yang dicapai pada $x_3(1) = 100$. Sehingga untuk iterasi berikutnya P_{best} = (99, 96, 100, 100.5) dan G_{best} = 100.

Cek apakah solusi sudah konvergen, dimana nilai x saling dekat. Jika belum konvergen, set *i* = 3, masuk ke iterasi berikutnya. Lanjutkan ke langkah berikutnya dengan menghitung kecepatan *v* dan ulangi langkah-langkah selanjunya sampai mencapai konvergen.

3. Modifikasi PSO

Dalam implementasinya, ditemukan bahwa kecepatan partikel dalam PSO standard *diupdate* terlalu cepat dan nilai minimum fungsi tujuan yang dicari sering terlewati. Karena itu kemudian dilakukan modifikasi atau perbaikan terhadap algoritma PSO standard. Perbaikan itu berupa penambahan suatu term inersia θ untuk mengurangi kecepatan pada formula update kecepatan. Biasanya nilai θ dibuat sedemikian hingga semakin besar iterasi

yang dilalui, semakin mengecil kecepatan partikel. Nilai ini bervariasi secara linier dalam rentang 0.9 hingga 0.4. Secara matematis perbaikan ini bisa dituliskan

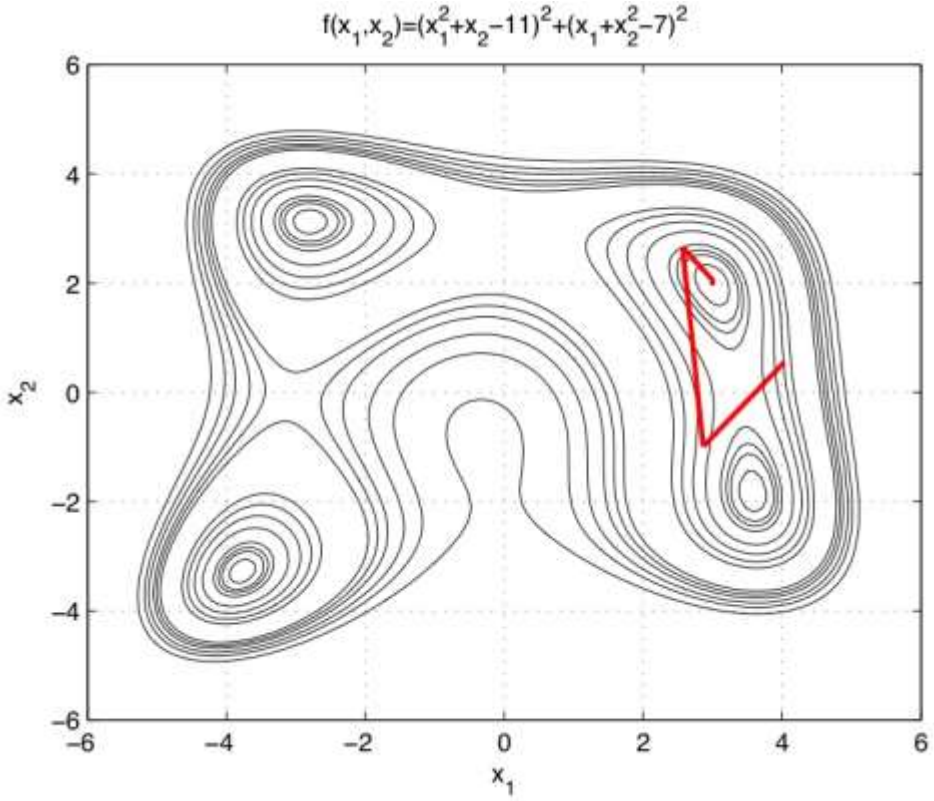
$$v_j(i) = \theta v_j(i-1) + c_1 r_1\left[P_{best,j} - x_j(i-1)\right] + c_2 r_2\left[G_{best} - x_j(i-1)\right], \qquad j = 1, 2, \dots, N \tag{9}$$

Bobot inersia ini diusulkan oleh Shi and Eberhart [1998] untuk meredam kecepatan selama iterasi, yang memungkinkan kawanan burung menuju (converge) titik target secara lebih akurat dan efisien dibandingkan dengan algoritma aslinya. Formula 10.9 adalah modifikasi terhadap formula 10.6. Nilai bobot inersia yang tinggi menambah porsi pencarian global (global exploration), sedangkan nilai yang rendah lebih menekankan pencarian lokal (local search). Untuk tidak terlalu menitikberatkan pada salah satu bagian dan tetap mencari area pencarian yang baru dalam ruang berdimensi tertentu, maka perlu dicari nilai bobot inersia (θ) yang secara imbang menjaga pencarian global dan lokal. Untuk mencapai itu dan mempercepat konvergensi, suatu bobot inersia yang mengecil nilainya dengan bertambahnya iterasi digunakan dengan formula

$$\theta(i) = \theta_{max} - \left(\frac{\theta_{max} - \theta_{min}}{i_{max}}\right) i \tag{10}$$

dimana θ_{max} dan θ_{min} masing-masing adalah nilai awal dan nilai akhir bobot inersia, i_{max} adalah jumlah iterasi maksimum yang digunakan dan i adalah iterasi yang sekarang. Biasanya digunakan nilai $\theta_{max} = 0.9$ dan $\theta_{min} = 0.4$. Perubahan atau modifikasi formula untuk mengupdate kecepatan ini seperti step size α dalam algoritma Steepest Descent, dimana nilai α yang terlalu besar akan memungkinan suatu optimum lokal akan terlewati sehingga algoritma justru menemukan optimum lokal yang lain yang tidak lebih baik nilainya.

Untuk implementasi ini digunakan fungsi Himmelblau, $f = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$.
Lihat Gmbar 1, bagaimana PSO mencapai titik optimal fungsi Himmelblau.



Gambar 1: Pergerakan titik menuju local minimum (3, 2) dengan PSO

4. PSO untuk TSP

PSO telah diterapkan diberbagai persoalan. Sebagai contoh, PSO diterapkan untuk vehicle routing problem (VRP) [Ai and Kachitvichyanukul, 2009a,b], penjadwalan proyek dengan resourse terbatas (RCPSP) [Zhang et al., 2006], job shop scheduling, travelling salesman problem (TSP) [Shi et al., 2007]. Juga diterapkan dalam data mining seperti klastering [Marinakis et al., 2007]. Dalam kasus-kasus seperti ini, maka kita perlu melakukan modifikasi sehingga PSO bisa diterapkan.

Dalam kasus TSP seperti dijelaskan dalam sub bab 2.3 misalnya, kita harus menemukan rute dengan jarak terpendek. Maka algoritma PSO seperti dalam kasus minimasi fungsi kontinyus tidak bisa langsung diterapkan. Perlu adanya modifikasi terhadap PSO sehingga bisa menyelesaikan problem TSP ini. Dalam sesi ini kita akan memperlihatkan bagaimana PSO diterapkancuntuk menyelesaikan kasus TSP. Seperti kita ketahui bahwa TSP adalah problem menemukan rute optimal dengan jarak total minimum

drengan banyak kombinasi solusi. Prosedur utama PSO akan kita ikuti dengan modifikasi sebagai berikut.

Prosedur PSO untuk TSP

- Lakukan inisialisasi: jumlah partikel, rute awal, c_1, c_2, jumlah iterasi, kecepatan awal, inersia,ba,bb
- Evaluasi jarak total dari masing-masing rute yang dihasilkan dari setiap partikel berdasarkan matrik jarak.
- Tentukan partikel dengan jarak total terkecil, dan tetapkan partikel ini sebagai G_{best}. Untuk setiap partikel, gunakan nilai awalnya sebagai P_{best}.
- Ulangi langkah berikut sampai stopping criteria dipenuhi

    1. Menggunakan *Pbest* dan *Gbest* yang ada, perbarui kecepatan setiap partikel menggunakan persamaan .3. Lalu dengan kecepatan baru yang didapat, perbarui nilai dari setiap partikel menggunakan persamaan .4.
    2. Jika nilai elemen setiap partikel di luar batas yang diijinkan, yaitu antara bb dan ba, maka lakukan penyesuaian,

$$pop(i,j) = \begin{cases} bb \text{ jika } pop(i,j) < bb \\ ba \text{ jika } pop(i,j) > ba \end{cases}$$

    3. Evaluasi jarak dari setiap rute yang dihasilkan oleh setiap partikel.
    4. Tentukan partikel dengan jarak total minimum, dan tetapkan partikel yang bersangkutan sebagai G_{best}. Untuk setiap partikel, tentukan P_{best} dengan membandingkan partikel sekarang dengan P_{best} dari iterasi sebelumnya berdasarkan jarak total minimumnya.
    5. Cek stopping criteria. Jika dipenuhi, berhenti. Jika tidak, kembali ke 1.

Berikut adalah contoh bagaimana implementasi PSO untuk kasus TSP dengan 5 kota. Misalkan seorang salesman ingin mengunjungi 5 kota. Kelima kota hanya dikunjungi sekali. Untuk itu dia ingin agar jarak total yang ditempuh minimum. Berikut adalah matrik jarak antar kota. Rute awal dibangkitkan dengan cara membangkitkan bilangan random antara (0, 1) sebanyak kota yang akan dikunjungi. Hal ini diulangi sebanyak jumlah partikel atau populasi. Kemudian kecepatan awal

	Kota 1	Kota 2	Kota 3	Kota 4	Kota 5
Kota 1	0	132	217	164	58
Kota 2	132	0	290	201	79
Kota 3	217	290	0	113	303
Kota 4	164	201	113	0	196
Kota 5	58	79	303	196	0

juga dibangkitkan melalui bilangan random (0, 1) sejumlah populasi. Perhatikan contoh berikut

$$pop = \begin{bmatrix} 0.8023 & 0.0732 & 0.0392 & 0.5178 & 0.9350 \\ 0.4242 & 0.5910 & 0.9463 & 0.9942 & 0.4795 \\ 0.7289 & 0.9102 & 0.7637 & 0.8549 & 0.2318 \\ 0.4984 & 0.1938 & 0.5588 & 0.9624 & 0.3963 \\ 0.8090 & 0.4324 & 0.1838 & 0.6789 & 0.7051 \\ 0.3565 & 0.7492 & 0.4979 & 0.4035 & 0.5586 \end{bmatrix}$$

Untuk setiap partikel, urutkan bilangan random dari yang terkecil. Pengurutan ini akan menghasilkan rute.

$$popurut = \begin{bmatrix} 0.0392 & 0.0372 & 0.5178 & 0.8023 & 0.9350 \\ 0.4242 & 0.4795 & 0.5910 & 0.9463 & 0.9942 \\ 0.2318 & 0.7289 & 0.7637 & 0.8549 & 0.9102 \\ 0.1938 & 0.3963 & 0.4984 & 0.5588 & 0.9624 \\ 0.1838 & 0.4324 & 0.6789 & 0.7051 & 0.8090 \\ 0.3565 & 0.4035 & 0.4979 & 0.5586 & 0.7492 \end{bmatrix}$$

Setelah masing-masing partikel diurutkan, maka akan didapat rute yang dibentuk dari posisi bilangan random tersebut sebelum diurutkan.

$$rute = \begin{bmatrix} 3 & 2 & 4 & 1 & 5 \\ 1 & 5 & 2 & 3 & 4 \\ 5 & 1 & 3 & 4 & 2 \\ 2 & 5 & 1 & 3 & 4 \\ 3 & 2 & 4 & 5 & 1 \\ 1 & 4 & 3 & 5 & 2 \end{bmatrix}$$

Sebagai ilustrasi misalkan partikel pertama sebelum diurutkan adalah

$$pop_1 = \begin{bmatrix} 0.8023 & 0.0732 & 0.0392 & 0.5178 & 0.9350 \end{bmatrix}$$

Dari urutan bilangan random ini menghasilkan rute 3 – 2 – 4 – 1 – 5. Rute ini didapat karena 0.0392 sebelum diurutkan berada pada posisi ke 3, 0.0732 berada pada posisi ke 2, 0.5178 berada pada posisi ke 4 dan seterusnya. Untuk partikel yang lain dilakukan cara yang sama untuk mendapatkan rute. G_{best} adalah nilai bilangan random yang dibangkitkan, bukan rutenya. Misalnya jarak minimum didapat dari partikel ke 3, maka G_{best} adalah

$$G_{best} = \begin{bmatrix} 0.7289 & 0.9102 & 0.7637 & 0.8549 & 0.2318 \end{bmatrix}$$

Untuk setiap partikel, *Pbest* awal akan sama dengan nilai partikel awal. Setelah itu akan dilakukan update untuk kecepatan. Misalkan kecepatan awal diberikan

$$v_0 = \begin{bmatrix} 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\ 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\ 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\ 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\ 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \\ 0.1000 & 0.1000 & 0.1000 & 0.1000 & 0.1000 \end{bmatrix},$$

maka kecepatan partikel ke 1 bisa diupdate dengan rumus

$$v_1 = \theta v_0 + r_1\left(Pbest_1 - x_1\right) + r_2\left(Pbest_1 - x_i\right)$$

$$v_1 = (0.9)([0.1\ \ 0.1\ \ 0.1\ \ 0.1\ \ 0.1]) + 0.4([0.8023\ \ 0.0732\ \ 0.0392\ \ 0.5178\ \ 0.9350] -$$

$$[0.8023\ \ 0.0732\ \ 0.0392\ \ 0.5178\ \ 0.9350]) +$$

$$0.6([0.7289\ \ 0.9102\ \ 0.7637\ \ 0.8549\ \ 0.2318] - [0.8023\ \ 0.0732\ \ 0.0392\ \ 0.5178\ \ 0.9350])$$

$$v_1 = [\ 0.5273\ \ 0.6361\ \ 0.5482\ \ 0.6029\ \ 0.2291\ ]$$

Untuk partikel 1 nilainya diupdate dengan

$$pop_1 = pop_1 + v_1$$

$$pop_1 = [\ 0.8023\ \ 0.0732\ \ 0.0392\ \ 0.5178\ \ 0.9350\ ]$$

$$+ [\ 0.5273\ \ 0.6361\ \ 0.5482\ \ 0.6029\ \ 0.2291\ ]$$

$$pop_1 = [\ 1.3296\ \ 0.7093\ \ 0.5874\ \ 1.1207\ \ 1.1641\ ]$$

Ini dilakukan untuk semua partikel. Disusul kemudian membuat nilai dalam setiap partikel antara [0,1] dengan cara membagi setiap nilai tersebut dengan jumlah totalnya. Dalam hal

ini jumlah totalnya adalah 4.9111. Sehingga nilai baru setelah setiap sel dibagi dengan 4.9111 adalah

[0.2707 0.1444 0.1196 0.2282 0.2370]

Lalu kita urutkan dari yang terkecil sebagai

[0.1196 0.1444 0.2282 0.2370 0.2707]

Sehingga rute yang terbentk adalah 3 - 2 - 4 - 5 – 1 - 3

Setelah itu akan diulangi lagi langkah mengevaluasi setiap rute yang dihasilkan setiap partikel, memilih P_{best}, G_{best}, update kecepatan dan seterusnya sampai stopping criteria dipenuhi.

6 Lampiran

Berikut ini diberikan kode Matlab untuk implementasi PSO asli untuk fungsi satu variabel.

```
function [xopt,fmin,it]=simpelpso(N,maxit)
%written by budi santosa to implement original PSO
% budi_s@ie.its.ac.id, faculty of industrial eng, ITS
%based on Engineering Optimization Theory and Practice
%book by Singiresu S. Rao
%to find minimum of single variable function
dim = 1; % Dimension of the problem
upbnd = 90; % Upper bound for init. of the swarm
lwbnd = 120; % Lower bound for init. of the swarm

% Initializing swarm and velocities
x = rand(N,dim)*(upbnd-lwbnd) + lwbnd ;
v = rand(N,dim); %kecepatan awal
[brs,kol]=size(x);
f=zeros(N,1);
%bobot inertia
for i=1:brs
     f(i)=fungsi2(x(i,:));
end
it=1;
Pbest=x;
fbest=f;
[minf,idk]=min(f);
```

```
      Gbest=x(idk,:);
      lastbest=[0 0];
      minftot=[];
while it<maxit
       r1=rand;r2=rand;
       for j=1:brs
       v(j,:)=v(j,:)+r1.*(Pbest(j,:)-x(j,:))+r2.*(Gbest-x(j,:));
       x(j,:)=x(j,:)+v(j,:);
       f(j)=fungsi2(x(j,:));
       end
      %update Pbest
      changerow = f < fbest;
      fbest=fbest.*(1-changerow)+f.*changerow;
      Pbest(find(changerow),:)=x(find(changerow),:);
      [minf,idk]=min(fbest);
      minftot=[minftot;minf];
      Gbest=Pbest(idk,:);
      if sum(var(Pbest))<1e-8
           break
      end
      it=it+1;
      lastbest=Gbest;
end
      xopt=Gbest;
      fmin=minf;
      plot(minftot)
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function f=fungsi2(x)
f=(100-x)^2;
```

Berikut adalah PSO yang dimodifikasi dengan menambahkan inertia untuk update kecepatan untuk minimasi fungsi Himmelblau.

```
function [xopt,fmin,it]=simpelpso2(N,maxit)
%written by budi santosa to implement modified PSO
% budi_s@ie.its.ac.id, faculty of industrial eng, ITS
%%to find minimum of Himmelblau function with two variables
dim = 2; % Dimension of the problem/number of variable
upbnd = [ 6 6]; % Upper bound for init. of the swarm
lwbnd = [-6 -6]; % Lower bound for init. of the swarm
% Initializing swarm and velocities
x = rand(N,dim).*repmat((upbnd-lwbnd),N,1)+repmat(lwbnd,N,1) ;
v = rand(N,dim); %kecepatan awal
[brs,kol]=size(x);
f = zeros(N,1);
rhomax=0.9;rhomin=0.4;
for it=1:maxit
rho(it)=rhomax-((rhomax-rhomin)/maxit)*it;
end
for i=1:brs
f(i)=himel(x(i,:));
end
%v=zeros(brs,kol);
it=1;
Pbest=x;
fbest=f;
[minf,idk]=min(f);
Gbest=x(idk,:);
minftot=[];
while it<maxit
 r1=rand;r2=rand;
 for j=1:brs
 v(j,:)=rho(it).*v(j,:)+r1.*(Pbest(j,:)-x(j,:))+...
 r2.*(Gbest-x(j,:));
 x(j,:)=x(j,:)+v(j,:);
 f(j)=himel(x(j,:));
 end
 %update Pbest
changerow = f < fbest;
fbest=fbest.*(1-changerow)+f.*changerow;
Pbest(find(changerow),:)=x(find(changerow),:);
[minf,idk]=min(fbest);
minftot=[minftot;minf];
Gbest=Pbest(idk,:);
if sum(var(Pbest))<1e-8
break
end
it=it+1;
end
xopt=Gbest;
fmin=minf;
```

```
plot(minftot)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function f=himel(x)
f=(x(1).^2+x(2)-11).^2+(x(1)+x(2).^2-7).^2;
```

Berikut adalah kode Matlab untuk penyelesaian TSP dengan PSO.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [rute_optim,jarak_minim,t]=pso_tsp(dx,ba,bb,np,itmax)
%fungsi ini digunakan utk penyelesaian optimasi kombinatorial
%pada tsp dengan fungsi tujuan minimasi jarak menggunakan
%pendekatan pso
%input:
     %dx = matrik jarak rute tsp, dimensi N x N, N:jumlah kota
     %ba,bb,np = batas atas (1),batas bawah(0),jml partikel
     %itmax = iterasi maksimum
%output:
     %rute_optim = rute tsp yang terbaik (optimal)
     %jarak_minim = jarak dari rute tsp yang terbaik
     %t = waktu komputasi

%STEP-1: INISIALISASI SECARA RANDOM PARTICLES xi
     %DAN KECEPATAN vi dalam RUANG PENCARIAN PROBLEM p-dimensi
t=cputime;
[r,c]=size(dx);
nk=c; %jumlah kota
  x = rand(np,nk)*(ba-bb)+ bb;
  v = rand(np,nk);
  %urutkan nilai random secara ascending untuk mendapatkan rute
  [min1 perm]=sort(x,2);
  perm_tsp=[perm perm(:,1)]; % menjamin kembai ke kota awal
  %STEP-2: EVALUASI NILAI FUNGSI TUJUAN jarak total tiap rute
  jarak=zeros(np,1);
  for i=1:np
     x1=perm_tsp(i,:);
     jarak(i)=jartsp(x1,dx);%memanggil fungsi penghitungan
     %jarak rute tsp
end
f=jarak; %fungsi tujuan


%STEP-3: MEMPERBARUHI NILAI Pbest DAN Gbest PARTIKEL AWAL
Pbest=x; %posisi terbaik individu (best local)
```

```
fbest=f; %fungsi tujuan terbaik
[minf,idk]=min(fbest);
Gbest=x(idk,:); %posisi terbaik swarm (best global)
minftot=[];
minfk=[];

%STEP-4:MEMPERBARUHI POSISI DAN KECEPATAN PARTIKEL
it=1; %setting iterasi
rhomax=0.9;rhomin=0.4; %rentang nilai inersia yg umum digunakan
while it<itmax
     r1=rand;r2=rand;
     rho=rhomax-((rhomax-rhomin)/itmax)*it; %bobot inersia
     for j=1:np
          v(j,:)=rho.*v(j,:)+r1.*(Pbest(j,:)-x(j,:))...
          + r2.*(Gbest-x(j,:));
          x(j,:)=x(j,:)+v(j,:);
     end
     %adjust agar x tidak melanggar interval (bb,ba)
     for i=1:np
          for j=1:nk-1
          if x(i,j) > ba
               x(i,j) = ba;
          end
          if x(i,j) < bb
               x(i,j)= bb;
          end
     end
end
%%urutkan nilai random untukmendapatkan rute dari
%terkecil ke terbesar
[min1 perm]=sort(x,2);
perm_tsp=[perm perm(:,1)]; %permutasi rute tsp

%EVALUASI NILAI FUNGSI TUJUAN PERMUTASI TSP
jarak=zeros(np,1);
for i=1:np
     x1=perm_tsp(i,:);
     jarak(i)=jartsp(x1,dx);%memanggil fungsi
     %penghitungan jarak rute tsp
end
f=jarak; %fungsi tujuan

%memperbaruhi fbest, Pbest dan Gbest
```

```
changerow = f < fbest;
fbest=fbest.*(1-changerow)+f.*changerow;
Pbest(find(changerow),:)=x(find(changerow),:);
[minf,idk]=min(fbest);
Gbest=Pbest(idk,:);
minftot=[minftot;minf];

it=it+1;%penambahan jumlah iterasi
end

%STEP-6:OUTPUT SOLUTION
lastbest=Pbest;%nilai random partikel terbaik pada
%iterasi terakhir
[min1 perm]=sort(lastbest,2);
perm_tsp=[perm perm(:,1)]; %rute tsp kembali ke kota awal

%Evaluasi Nilai Fungsi Tujuan Pada Iterasi Tahap Akhir
jarak=zeros(np,1);
for i=1:np
     x1=perm_tsp(i,:);
     jarak(i)=jartsp(x1,dx);%memanggil fungsi penghitungan
     % jarak rute tsp
end
f=jarak; %fungsi tujuan
[jarak_minim,idx]=min(f);
rute_optim=perm_tsp(idx,:);
t=cputime - t;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function jarak=jartsp(x1,dx)
%fungsi jartsp
%input:
     %x1= rute tsp (contoh: 1 2 3 4 5 1, 1=kota asal,
     % 2-3-4-5=kota tujuan)
     %dx= matrik jarak antar kota
%output:
     %jarak = jarak total rute tsp
[r,c]=size(x1);
k=c-1;%jumlah kota dalam rute tsp
s=0; %jarak awal di kota pertama
for j=1:k
     s=s+dx(x1(j),x1(j+1)); %pengakumulasian jarak rute tsp
end
jarak=s;
```

References

[1] Budi Santosa dan Paul Willy, Metoda metaheuristik, Konsep dan Implementasi, Graha Ilmu 2011.
[2] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*. IEEE Service Center, Piscataway, 1995.
[3] Singiresu S. Rao. *Engineering Optimization, Theory and Practice*. John Wiley & Sons, New York, fourth edition, 2009.
[4] Y. Shi and R. C. Eberhart. Parameter selection in particle swarm optimization.
In V. W. Porto, N. Saravanan, D. Waagen, and A. Eibe, editors, *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, page 591-600. Springer-Verlag, 1998.