PERBANDINGAN ALGORITMA DBPSO, MBPSO, DAN HBPSO UNTUK MENYELESAIKAN PERMASALAHAN *MULTIDIMENSIONAL KNAPSACK 0/1*

SKRIPSI

Oleh:

FIDIA DENY TISNA AMIJAYA

0610940018-94

PROGRAM STUDI MATEMATIKA

JURUSAN MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

MALANG

2011

PERBANDINGAN ALGORITMA DBPSO, MBPSO, DAN HBPSO UNTUK MENYELESAIKAN PERMASALAHAN *MULTIDIMENSIONAL KNAPSACK 0/1*

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana Sains dalam bidang matematika

Oleh:
FIDIA DENY TISNA AMIJAYA
0610940018-94

PROGRAM STUDI MATEMATIKA
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2011

LEMBAR PENGESAHAN SKRIPSI

PERBANDINGAN ALGORITMA DBPSO, MBPSO, DAN HBPSO UNTUK MENYELESAIKAN PERMASALAHAN *MULTIDIMENSIONAL KNAPSACK 0/1*

Oleh:
FIDIA DENY TISNA AMIJAYA
0610940018-94

Setelah dipertahankan di depan Majelis Penguji pada tanggal
16 Agustus 2011
dan dinyatakan memenuhi syarat untuk
memperoleh gelar Sarjana Sains dalam bidang Matematika

Pembimbing I	Pembimbing II
Syaiful Anam, S.Si, MT	**Drs. Imam Nurhadi Purwanto, MT**
NIP. 19780115 200212 1 003	**NIP. 19620314 198903 1 001**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf Alghofari, M.Sc
NIP. 19670907 199203 1 001

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**: Fidia Deny Tisna Amijaya**
Nim	**: 0610940018-94**
Jurusan	**: Matematika**
Penulis Skripsi berjudul	**: Perbandingan Algoritma DBPSO, MBPSO, dan HBPSO untuk Menyelesaikan Permasalahan *Multidimensional Knapsack 0/1***

Dengan ini menyatakan bahwa :

1. **Skripsi ini adalah benar-benar karya saya sendiri dan bukan hasil plagiat dari karya orang lain. Karya-karya yang tercantum dalam Daftar Pustaka, semata-mata digunakan sebagai acuan/referensi.**
2. **Apabila di kemudian hari diketahui bahwa isi Skripsi saya merupakan hasil plagiat, maka saya bersedia menanggung akibat hukum dari keadaan tersebut.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 16 Agustus 2011
Yang menyatakan,

(Fidia Deny Tisna Amijaya)
NIM 0610940018

PERBANDINGAN ALGORITMA DBPSO, MBPSO, DAN HBPSO UNTUK MENYELESAIKAN PERMASALAHAN *MULTIDIMENSIONAL KNAPSACK 0/1*

ABSTRAK

Particle Swarm Optimization (PSO) adalah sebuah algoritma berbasis kecerdasan yang terinspirasi oleh sekawanan burung. Algoritma PSO telah banyak digunakan untuk menyelesaikan permasalahan optimasi, akan tetapi *basic PSO* dan sebagian besar variannya hanya dikembangkan untuk menyelesaikan permasalahan kontinu dan tidak dapat digunakan untuk menyelesaikan permasalahan diskrit. Untuk mengatasi masalah ini, Kennedy memperluas penggunaan penggunaaan *basic PSO* dan memperkenalkan algoritma *discrete binary PSO* (DBPSO). Akan tetapi kemampuan algoritma DBPSO untuk menyelesaikan masalah masih belum ideal. Kemudian Qi melanjutkan pengembangan algoritma ini dan memperkenalkan algoritma *Modified Binary PSO* (MBPSO). Hasil yang didapat lebih bagus dari algoritma DBPSO, akan tetapi kemungkinan solusi untuk terjebak pada optimum lokal masih besar. Selanjutnya pada skripsi ini, akan diperkenalkan algoritma *Hybrid Binary PSO* (HBPSO). Pada algoritma HBPSO, *update* kecepatan dan posisinya mengikuti algoritma DBPSO dan ditambahkan beberapa operator genetika seperti *crossover* dan mutasi untuk menjaga keberagaman solusi agar tidak mudah terjebak pada optimum lokal. Untuk melihat kemampuan algoritma HBPSO, permasalahan *multidimensional knapsack 0/1* digunakan sebagai tes uji. Hasil dari pengujian menunjukkan bahwa algoritma HBPSO mempunyai kemampuan yang lebih baik dalam menyelesaikan permasalahan *multidimensional knapsack 0/1* dibanding dua algoritma lainnya dalam hal keakuratan solusi.

.

Kata kunci : algoritma PSO diskrit, permasalahan *multidimensional knapsack 0/1*, optimasi.

THE COMPARISON OF DBPSO, MBPSO, AND HBPSO ALGORITHM TO SOLVE MULTIDIMENSIONAL KNAPSACK 0/1 PROBLEM

ABSTRACT

Particle swarm optimization (PSO) is an intelligent optimization algorithm inspired by flocking behavior of birds. PSO algorithm has been widely used to solve the optimization problem, but the basic PSO algorithm and most of its variants are developed for continuous problems and not able to solve the discrete problem. To tackle this problem, Kennedy extended the basic PSO algorithm and proposed a discrete binary PSO (DBPSO) algorithm. But its performance is not ideal. Then Qi continued development DBPSO algorithm and proposed modified binary PSO (MBPSO) algorithm. The result is better than DBPSO algorithm but it still have a chance to trap at local optimum. Furthermore in this paper, we presented a new variant discrete PSO algorithm called hybrid binary PSO (HBPSO) algorithm. In HBPSO algorithm, updating velocity and position followed DBPSO algorithm and added by genetic operator like crossover and mutation to keep the diversity of the solution in order not to trap at local optimum. To see the performance of HBPSO algorithm, multidimensional knapsack 0/1 problems are used as the test benchmark. The experimental results show that HBPSO algorithm has a better performance to solve multidimensional knapsack 0/1 problems than the others in terms of to get the precise solution.

Keywords: the discrete PSO algorithm, multidimensional knapsack 0/1 problems, optimization.

KATA PENGANTAR

Segala puji dan syukur Alhamdulillah penulis panjatkan kehadirat Allah SWT, yang telah mencurahkan rahmat, hidayah dan inayah-Nya sehingga penulis dapat menyelesaikan skripsi ini dengan judul "**Perbandingan Algoritma DBPSO, MBPSO, dan HBPSO untuk Menyelesaikan Permasalahan *Multidimensional Knapsack 0/1***". Sholawat serta salam semoga tercurahkan kepada Baginda Rosulullah Nabi Muhammad SAW. Banyak pihak yang telah memberikan dukungan baik moral maupun spiritual secara langsung maupun tidak langsung dalam penyelesaian skripsi ini. Penulis mengucapkan terima kasih kepada :

1. Syaiful Anam, S.Si, MT selaku pembimbing I atas segala bimbingan dan motivasi yang telah diberikan selama penulisan skripsi ini.
2. **Drs. Imam Nurhadi Purwanto, MT** selaku pembimbing II atas segala bimbingan dan motivasi yang telah diberikan selama penulisan skripsi ini.
3. Dra. Ari Andari, MS, Prof. Dr. Marjono, M.Phil, dan Dr. Ratno Bagus Edy Wibowo, M.Si selaku dosen penguji atas segala saran yang diberikan untuk perbaikan skripsi ini.
4. Dr. Abdul Rouf Alghofari, M.Sc selaku Ketua Jurusan Matematika atas nasihat dan motivasi yang telah diberikan.
5. Prof. Dr. Marjono, M.Phil selaku dosen pembimbing akademik atas segala bimbingan dan motivasi yang telah diberikan.
6. Seluruh Bapak/Ibu dosen Jurusan Matematika yang telah memberikan ilmunya kepada penulis, serta segenap staf dan karyawan Tata Usaha Jurusan Matematika atas segala bantuannya.
7. Ayahku tercinta Bapak Mufid, Ibuku tercinta Ibu Tutik dan Adikku tersayang Dik Indri tercinta atas doa dan dukungan yang telah diberikan baik secara moril, spiritual dan material.
8. Sahabat-sahabatku (Suryo, Agus, Dhebi), teman-teman jurusan matematika khususnya matematika 2006 yang telah memberikan banyak kenangan dan kebahagian tersendiri bagi penulis.

9. Teman-teman kost 206 serta keluarga Ibu Halima jalan sumbersari gg. III no. 206 yang selalu ada dalam keadaan dan kondisi apapun bagi penulis.
10. Serta semua pihak yang tidak dapat penulis sebutkan satu persatu.

Penulis menyadari bahwa manusia adalah tempatnya kekhilafan dan tak akan lepas dari kesalahan, oleh karena itu segala kritik dan saran dari pembaca sangat diharapkan demi perbaikan selanjutnya. Kritik dan saran bisa dikirim di anjaoye@yahoo.com.

Akhir kata, penulis berharap semoga skripsi ini dapat memberikan manfaat dan sumbangan yang berarti bagi bidang sains matematika pada masa yang akan datang

Malang, 1 Agustus 2011

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 LatarBelakang

Permasalahan *knapsack* adalah salah satu permasalahan optimasi kombinatorial dalam riset operasi. Permasalahan ini mempunyai banyak sekali aplikasi misalkan dalam pemilihan sumber alokasi, penyimpanan barang dan lain-lain. Permasalahan *knapsack* termasuk dalam *nondeterministic polynomial* (NP) *problem*, yang biasanya membutuhkan waktu untuk mencari solusinya (Yanbing dkk, 2010). Pengembangan dari permasalahan *knapsack* salah satunya adalah permasalahan *multidimensional knapsack 0/1* yang mana permasalahan ini lebih kompleks dari permasalahan *knapsack*. Sehingga dibutuhkan algoritma yang lebih cepat dan akurat untuk menyelesaikan permasalahan ini.

Particle Swarm Optimization (PSO) adalah algoritma yang dibuat oleh Kennedy dan Eberhart pada tahun 1995. Algoritma PSO merupakan algoritma pencarian acak yang didasarkan pada proses adaptasi burung di alam bebas dan telah banyak dgunakan untuk menyelesaikan permasalahan optimasi yang sulit. Algoritma PSO sangat mudah dijalankan karena algoritma PSO mempunyai sedikit parameter yang dirubah-rubah. Selain itu, algoritma PSO juga mempunyai tingkat kekonvergenan yang cepat karena hanya mempunyai dua rumus perhitungan di setiap iterasinya. Berdasarkan kelebihan-kelebihan di atas algoritma PSO tidak hanya digunakan dalam bidang keilmuan yang bersifat eksak, tetapi juga dibidang terapan seperti jaringan syaraf, optimasi non linier, dan masalah *power flow*. (Ling dkk, 2008)

Basic PSO dan pengembangannya rata-rata digunakan untuk menyelesaikan permasalahan kontinu dan tidak dapat digunakan untuk mengoptimalkan penyelesaian masalah kombinatorial diskrit. Untuk mengatasi masalah ini, pengembangan *discrete PSO* mulai dilakukan diberbagai bidang. Kennedy dan Eberhart pertama kali memperluas penggunaan *basic PSO* dan mengembangkan *Discrete Binary PSO* (DBPSO) untuk mengoptimalkan optimasi permasalahan biner. Kemudian Qi Shen mengembangkan *Modified Binary PSO* (MBPSO) untuk *feature selection* dalam *Multiple Linear Regression* dan *Partial Least Square Modelling*. Karena kemampuan optimasi algoritma *discrete PSO* ini tidak sempurna, hanya sedikit

peneliti yang mengembangkan algoritma *discrete PSO* ini. (Wang Ling dkk, 2008)

Untuk mendorong perkembangan algoritma *discrete PSO*, skripsi ini akan membahas perkembangan terbaru dari algoritma PSO untuk permasalahan diskrit yaitu *Hybrid Binary PSO* (HBPSO). Algoritma HBPSO menggabungkan algoritma PSO dengan beberapa operator genetika dari algoritma genetika, seperti *crossover* dan mutasi. Algoritma PSO digunakan untuk memperbaharui kecepatan dan posisi, sedangkan operator genetika digunakan untuk menjaga keberagaman solusi agar algoritma tidak terjebak pada titik optimum lokal. (Ling dkk, 2008)

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka masalah yang akan dibahas dalam skripsi ini adalah sebagai berikut.

1. Bagaimanakah implementasi algoritma DBPSO, MBPSO dan HBPSO untuk menyelesaikan permasalahan *multidimensional knapsack 0/1*?
2. Bagaimanakah perbandingan algoritma DBPSO, MBPSO dan HBPSO untuk menyelesaikan permasalahan *multidimensional knapsack 0/1*?

1.3 Tujuan

Berdasarkan pada masalah di atas, tujuan penulisan skripsi ini adalah :

1. Menjelaskan implementasi algoritma DBPSO, MBPSO dan HBPSO untuk menyelesaikan permasalahan *multidimensional knapsack 0/1*.
2. Melihat perbandingan algoritma DBPSO, MBPSO dan HBPSO untuk menyelesaikan permasalahan *multidimensional knapsack 0/1*.

BAB II
TINJAUAN PUSTAKA

Dalam tinjauan pustaka ini, akan membahas materi-materi yang menjadi dasar penulisan skripsi. Dimulai dari pengertian optimasi yang menjadi tujuan umum penulisan skripsi, cabang ilmu yang mempelajari permasalahan yang dibahas, permasalahan yang dibahas sampai penjelasan algoritma yang digunakan untuk menyelesaikan permasalahan yang akan dibahas.

2.1 Optimasi

Optimasi merupakan masalah memaksimumkan atau meminimumkan suatu besaran tertentu, yang disebut dengan fungsi tujuan. Fungsi tujuan bergantung pada sejumlah variabel yang tidak saling berhubungan atau saling bergantung melalui satu atau lebih kendala.

(Bronson, 1996)

2.2 Maksimum Lokal dan Maksimum Global

Misalkan $f: \mathbb{X} \to \mathbb{R}$ dengan $\mathbb{X} \subseteq \mathbb{R}^m$, titik $\boldsymbol{x}^* \in \mathbb{X}$ dikatakan sebagai :

1. titik maksimum lokal fungsi f jika untuk semua $\boldsymbol{x}$ persekitaran $\boldsymbol{x}^*$ berlaku $f(\boldsymbol{x}^*) \geq f(\boldsymbol{x})$
2. titik maksimum global dari fungsi f jika untuk semua $\boldsymbol{x} \in \mathbb{X}$ berlaku $f(\boldsymbol{x}^*) \geq f(\boldsymbol{x})$.

(Weise, 2009)

2.3 Fungsi Kontinu dan Fungsi Diskrit

Sebuah fungsi $f : \mathcal{R}^n \to \mathcal{R}$ dapat digambarkan pada setiap titik x dalam persekitaran p. Maka fungsi f dikatakan kontinu pada titik p jika :

1. f terdefinisi pada p, dan
2. $\lim_{x \to p} f(x) = f(p)$

(Apostol, 1967)

Menurut Mark G. Karpovsky dkk, suatu perlengkapan kombinasi yang mempunyai nilai masukan yang berhingga dan sejumlah nilai kemungkinan yang berhingga sesuai dengan nilai keluarannya dapat disebut sebagai fungsi diskrit berhingga

$$f : \times_{i=0}^{m-1} D_i \to \times_{i=0}^{k-1} \mathcal{R}_i$$

yang mana $\times$ menyatakan hasil kali kartesian dan D_i , $i = 0, 1, \ldots, m-1$ adalah himpunan berhingga.

Himpunan $\mathcal{R}_i$ dapat berupa bilangan berhingga atau tak berhingga. Dan biasanya, jika tak berhingga, himpunan $\mathcal{R}_i$ dapat berupa himpunan $\mathcal{R}$ bilangan riil atau himpunan $\mathbb{C}$ bilangan kompleks.

(Karpovsky dkk, 2008)

2.4 Fungsi Obyektif dan Fungsi Kendala

Sebuah masalah optimasi yang disimbolkan dengan D sebagai daerah asal dan sebuah fungsi bilangan real $f : D \to \mathcal{R}$ dinamakan fungsi obyektif. $f(x) \in \mathcal{R}$ menyatakan "keuntungan" atau "harga" yang mana $x \in D$.

Masalah optimasi mempunyai dua tujuan : memaksimumkan dan meminimumkan. Pada masalah maksimasi, tujuannya adalah untuk menemukan sebuah $x \in D$ yang mana $f(y) \leq f(x)$ untuk semua $y \in D$. Dengan kata lain, ingin didapatkan daerah asal dengan nilai keuntungan yang paling tinggi. Pada masalah minimasi, merupakan kebalikan dari masalah maksimasi. Tujuannya adalah menemukan sebuah $x \in D$ yang mana $f(x) \leq f(y)$ untuk semua $y \in D$. Pada kasus ini ingin dicari daerah asal yang mempunyai harga terendah.

(Glazer, 2005)

Fungsi kendala adalah fungsi yang menjadi kendala/syarat/batas dari fungsi obyektif. Dalam kasus nyata, fungsi kendala merupakan kemampuan perusahaan untuk membuat fungsi menjadi maksimal atau minimal.

2.5 Kombinatorial

Kombinatorial adalah cabang matematika yang mempelajari pengaturan obyek-obyek. Dalam matematika terdapat 2 kaidah dasar dalam menghitung, yaitu :

a. Kaidah Perkalian (*rule of product*)

   Misalkan terdapat 2 percobaan, percobaan 1 menghasilkan p hasil dan percobaan 2 menghasilkan q hasil. Maka percobaan 1 dan percobaan 2 adalah $p \times q$ hasil.

   Contoh :

Akan dipilih ketua masing-masing satu dari 2 organisasi, organisasi A berjumlah 65 orang. Organisasi B berjumlah 15 orang. Berapa banyak cara untuk memilih ketua tersebut?

Penyelesaian : $65 \times 15 = 975$ cara

b. Kaidah Penjumlahan (*sum of product*)

Misalkan terdapat 2 percobaan, percobaan 1 menghasilkan p hasil dan percobaan 2 menghasilkan q hasil. Maka percobaan 1 atau percobaan 2 adalah $p + q$ hasil.

Contoh :

Akan dipilih satu ketua dari 2 organisasi, organisasi A berjumlah 65 orang. Organisasi B berjumlah 15 orang. Berapa banyak cara untuk memilih ketua tersebut?

Penyelesaian : $65 + 15 = 80$ cara

Dari 2 kaidah dasar menghitung tersebut dapat diperluas. Misalkan terdapat *n* percobaan, masing-masing dengan p_i hasil, maka:

a. Kaidah Perkalian (*rule of product*)

$$p_1 \times p_2 \times ... \times p_n \textbf{ hasil} \quad (2.1)$$

b. Kaidah Penjumlahan (*sum of product*)

$$p_1 + p_2 + \cdots + p_n \textbf{ hasil} \quad (2.2)$$

(Siang, 2002)

2.6 Kompleksitas Algoritma

Kompleksitas algoritma adalah besaran yang dipakai untuk menerangkan ukuran waktu/ruang dari suatu algoritma. Kompleksitas algoritma dibagi menjadi dua :

a. Kompleksitas waktu *T(n)*

Jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan *n*. Kompleksitas waktu biasanya disebut juga waktu polinomial atau dalam dunia komputasi dinamakan waktu komputasi.

Contoh :

Proses untuk menghitung rerata

```
Procedure HitungRerata(input a1, a2, a3, …, an : integer.
output r: real)
```

```
Deklarasi
  k : integer
  jumlah : real

Algoritma
  jumlah ← 0
  k ← 1
  while k ≤ n do
        jumlah ← jumlah + a_k
    k ← k+1
  endwhile
  r ← jumlah/n
```

(i) Operasi pengisian nilai

t1 = (jumlah ← 0)+(k ← 1)+(jumlah ← jumlah + a_k)+
(k ← k+1)+(r ← jumlah/n)

t1 = 1 + 1 + n + n + 1 = **3 + 2*n***

(ii) Operasi penjumlahan

t2 = (jumlah + a_k)+(k+1)

t2 = n + n = **2*n***

(iii) Operasi pembagian

t3 = (jumlah/n)

t3 = **1**

Total kebutuhan waktu algoritma *HitungRerata*:

t = t1 + t2 + t3 = (3 + 2n)a + (2n)b + c detik

b. Kompleksitas ruang *S(n)*

Jumlah memori yang digunakan oleh struktur data yang terdapat di dalam algoritma sebagai fungsi dari ukuran masukan *n*. Biasanya ukuran ini menggunakan satuan *byte*.

(Stroppa, 2006)

2.7 *Class Polynomial Time (Class P)*

Class polynomial time didefinisikan sebagai suatu masalah yang dapat diputuskan/diselesaikan dalam waktu polinomial.

Contoh :

Diberikan suatu himpunan bilangan bulat. Tentukan bilangan prima yang ada dalam himpunan tersebut.

Diberikan suatu matriks. Tentukan apakah matriks tersebut mempunyai determinan.

(Wigderson, 2006)

2.8 *Nondeterministic Polynomial Time (NP)*

Nondeterministic polynomial time adalah suatu masalah yang dapat diputuskan/diselesaikan dalam waktu polinomial dan mempunyai calon solusi.

Contoh :

Dalam bidang ilmu pengetahuan, diberikan suatu data dari suatu fenomena. Tentukan teori yang bisa menjelaskan fenomena tersebut.

Dalam bidang detektif, diberikan suatu adegan kejahatan. Temukan siapa pelakunya.

Dalam bidang riset operasi, *Traveling Salesman Problem* (TSP).

(Wigderson, 2006)

2.9 Masalah *Knapsack*

Masalah ini dapat diilustrasikan sebagai permasalahan seorang pencuri yang akan mengisi tasnya dengan memilih semua barang yang mungkin sehingga dia mendapatkan hasil yang maksimal. Secara matematis, didalam masalah *knapsack* ini terdapat sejumlah obyek dari 1 sampai n dan variabel x_i $(i = 1, \dots, n)$ yang mempunyai arti :

$$x_i = \begin{cases} 1, & \text{memuat barang ke} - \text{i kedalam tas} \\ 0, & \text{selainnya} \end{cases}$$

kemudian, jika a_i merupakan ukuran dari kepuasan dari obyek ke-i, b_i adalah berat barang ke-i dan B adalah kapasitas maksimum tas, maka dapat dirumuskan :

$$\text{memaksimalkan} \sum_{i=1}^{n} a_i x_i \text{ dengan syarat } \sum_{i=1}^{n} b_i x_i \leq B \qquad (2.3)$$

Masalah *knapsack* mempunyai banyak sekali variasi diantaranya :

1. *Knapsack 0/1*

   memaksimalkan $\sum_{i=1}^{n} a_i x_i$

   dengan syarat $\sum_{i=1}^{n} b_i x_i \leq B, \quad x_i \in \{0,1\}, \quad i = 1, \dots, n$

2. Masalah *Knapsack* Terbatas

   memaksimalkan $\sum_{i=1}^{n} a_i x_i$

   dengan syarat $\sum_{i=1}^{n} b_i x_i \leq B\,, x_i \in \{0,1, \dots, c_i\}, i = 1, \dots, n$

   c_i merupakan batas atas jumlah barang ke-i.

3. Masalah *Knapsack* Tak Terbatas

   memaksimalkan $\sum_{i=1}^{n} a_i x_i$

   dengan syarat $\sum_{i=1}^{n} b_i x_i \leq B\,, x_i \in \{0,1, \dots, +\infty\}\,, i = 1, \dots, n$

4. Masalah *Multidimensional Knapsack 0/1*

   memaksimalkan $\sum_{i=1}^{n} a_i x_i$

   dengan syarat $\sum_{i=1}^{n} b_{ki} x_i \leq B_k\,, x_i \in \{0,1\}, k = 1, \dots, c\,,$

   $i = 1, \dots, n$
   c adalah banyak kendala.

(Martello, 1990)

2.10 *Particle Swarm Optimization* (PSO)

Particle swarm optimization (PSO) adalah populasi berdasarkan teknik optimasi stokastik yang dikembangkan oleh Eberheart dan Kennedy pada tahun 1995. Algoritma PSO di ilhami oleh tingkah laku sosial dari kawanan burung, gerombolan hewan, dan kumpulan ikan. Solusi potensial pada PSO dipengaruhi oleh partikel yang terbang melalui ruang masalah dengan mengikuti partikel yang optimum.

Seperti dijelaskan sebelumnya, PSO menirukan kebiasaan kawanan burung. Andaikan kebiasaan tersebut seperti digambarkan pada masalah berikut ini. Sekelompok burung secara acak mencari makanan di suatu area. Hanya ada sepotong makanan pada area tersebut yang akan dicari. Tidak semua burung tahu dimana makanan tersebut berada. Oleh karena itu, strategi terbaik untuk menemukan makanan adalah dengan mengikuti burung yang sangat dekat dengan makanan tersebut. PSO belajar dari masalah itu, dan menggunakannya untuk menyelesaikan masalah optimasi. Pada PSO, setiap solusi dianalogikan dengan burung/partikel pada ruang pencarian.

(Abdelhalim dan Habib, 2009)

Secara matematis, posisi partikel ke-i dinyatakan sebagai $\boldsymbol{X_i} = (X_{i1}, X_{i2}, \dots, X_{iD})$ yang mana D merupakan dimensi tempat partikel tersebut berada. Setiap partikel mempunyai posisi terbaik yang berhubungan dengan nilai *fitness* terbaik yang ditemukan oleh setiap partikel ke-i. Posisi terbaik tersebut dinyatakan sebagai ***pbest*** yang disimbolkan sebagai $\boldsymbol{P_i} = (P_{i1}, P_{i2}, \dots, P_{iD})$. Pada skripsi ini posisi yang optimal adalah posisi yang mempunyai nilai fungsi terbesar sehingga penentuan ***pbest*** setiap partikel dinyatakan sebagai

$$\boldsymbol{P_i}(t+1) = \begin{cases} \boldsymbol{P_i}(t) \text{ jika } f\big(\boldsymbol{P_i}(t)\big) > f\big(\boldsymbol{X_i}(t+1)\big) \\ \boldsymbol{X_i}(t+1) \text{ jika } f\big(\boldsymbol{P_i}(t)\big) < f\big(\boldsymbol{X_i}(t+1)\big) \end{cases} \quad (2.4)$$

Partikel terbaik diantara ***pbest*** disebut ***gbest***. Pada skripsi ini nilai terbaik adalah nilai yang bernilai maksimum maka ***gbest***$(\boldsymbol{P_g})$ dapat dinyatakan sebagai

$$f\left(\boldsymbol{P_g}(t)\right) > f\big(\boldsymbol{P_i}(t)\big),\ i = 1,2,3,..,m \quad (2.5)$$

dengan m adalah banyak partikel.

Setiap partikel bergerak dengan kecepatan yang dinyatakan $\boldsymbol{V_i} = (V_{i1}, V_{i2}, \dots, V_{iD})$. Setiap partikel bergerak berdasarkan informasi mengenai jarak antara posisi $\boldsymbol{X_i}$ dan ***pbest***, dan jarak antara posisi $\boldsymbol{X_i}$ dan ***gbest***. Untuk itu perbaikan kecepatan dinyatakan sebagai

$$\boldsymbol{V_i}(t+1) = w\boldsymbol{V_i}(t) + c_1 r_1\big(\boldsymbol{P_i} - \boldsymbol{X_i}(\boldsymbol{t})\big) + c_2 r_2(\boldsymbol{P_g} - \boldsymbol{X_i}(t)), \quad (2.6)$$

dimana $\boldsymbol{V_i}$ = kecepatan tiap partikel
$\boldsymbol{X_i}$ = posisi tiap partikel
w = faktor inersia
c_1 = konstanta kecepatan individu
c_2 = konstanta kecepatan sosial
P_i = *pbest* partikel ke-i
P_g = *gbest*
r_1, r_2 = bilangan acak
$i \in \{1,2,3, \dots, m\}$.

Posisi partikel diperbaiki dengan menggunakan nilai kecepatan partikel yang baru yaitu

$$\boldsymbol{X_i}(t+1) = \boldsymbol{V_i}(t+1) + \boldsymbol{X_i}(\boldsymbol{t}) \quad (2.7)$$

(Premalatha dan Natarajan, 2009)

2.10.1 Parameter PSO

Terdapat beberapa parameter dalam algoritma PSO yang berpengaruh terhadap keberhasilan algoritma, yaitu:

a. Bobot inersia (w).

   Konstanta ini mengatur seberapa besar kecepatan iterasi sebelumnya dan mempengaruhi kecepatan iterasi berikutnya. Konstanta ini sangat mempengaruhi konvergensi algoritma. Nilai $w \geq 1$ meningkatkan kecepatan seiring waktu dan menyebabkan partikel gagal merubah arah ke posisi yang terbaik sehingga menyebabkan solusi divergen. Untuk $w < 1$ menyebabkan partikel mengurangi kecepatan seiring dengan waktu sampai kecepatan nol.

b. Koefisien percepatan ($c_1$dan c_2).

   Kedua koefisien ini tidak terlalu berpengaruh terhadap kekonvergenan algoritma PSO, tetapi pemilihan nilai yang tepat dapat meningkatkan kecepatan konvergensi dan

menyebabkan algoritma tidak mudah terjebak dalam optimum lokal.

c. Nilai Acak (r_1 dan r_2).

Nilai acak digunakan untuk memelihara keberagaman partikel. Nilai tersebut dibangkitkan mengikuti distribusi seragam dengan parameter (0,1).

d. Ukuran *swarm* (m).

Ukuran populasi merupakan parameter yang berfungsi untuk menentukan jumlah partikel yang berada pada populasi. Semakin banyak dan beragamnya partikel akan memungkinkan pencarian yang semakin luas akan tetapi semakin besar ukuran populasi akan meningkatkan waktu komputasi.

e. Jumlah iterasi.

Jumlah iterasi mempunyai andil besar dalam menemukan hasil yang lebih baik, jumlah iterasi yang terlalu kecil akan menghentikan pencarian terlalu dini akan tetapi terlalu besar juga akan memperlama waktu komputasi.

(Parsopaulus, 2002).

2.10.2 Kekonvergenan Algoritma PSO

Nilai yang dihasilkan algoritma PSO tidak dijamin optimum. Parameter bobot inersia (w) dalam algoritma ini adalah parameter yang mempengaruhi kekonvergenan, untuk itu perlu dicari kisaran nilai yang sesuai. Langkah yang dilakukan adalah mencari bentuk eksplisit posisi partikel ke-i dalam suatu dimensi ke-j pada waktu ke-t dan kemudian diselidiki posisi partikel pada waktu $t \to \infty$

Dengan mensubtitusikan persamaan (2.6) pada persamaan (2.7) dengan mengasumsikan $\emptyset_1 = c_1 r_1$, $\emptyset_2 = c_2 r_2$, $p_b = \boldsymbol{P_i}(t)$, $p_g = \boldsymbol{P_g}(t)$ konstan selama iterasi *t* berjalan, maka diperoleh

$$x_{t+1} = wv_t + \emptyset_1(p_b - x_t) + \emptyset_2(p_g - x_t) + x_t$$
$$x_{t+1} = (1 - \emptyset_1 - \emptyset_2)x_t + \emptyset_1 p_b + \emptyset_2 p_g + wv_t$$

pada persamaan (2.7) diperoleh $x(t) = v(t) + x(t-1)$ maka persamaan menjadi

$$x_{t+1} = (1 - \emptyset_1 - \emptyset_2)x_t + \emptyset_1 p_b + \emptyset_2 p_g + w(x_t - x_{t-1})$$
$$x_{t+1} - (1 + w + \emptyset_1 - \emptyset_2)x_t + wx_{t-1} = \emptyset_1 p_b + \emptyset_2 p_g \,, \quad (2.8)$$

yang merupakan relasi nonhomogen.

Untuk menentukan solusi relasi rekurensi nonhomogen tersebut, langkah yang dilakukan adalah sebagai berikut:

1. Mencari solusi umum persamaan nonhomogen.
   Misalkan $x_{t-1} = \alpha^t$ maka persamaan (2.8) menjadi
   $$\alpha^{t+2} - (1 + w - \emptyset_1 - \emptyset_2)\alpha^{t+1} + w\alpha^t = \emptyset_1 p_b + \emptyset_2 p_g$$
   bentuk homogennya adalah
   $$\alpha^{t+2} - (1 + w - \emptyset_1 - \emptyset_2)\alpha^{t+1} + w\alpha^t = 0,$$
   dengan persamaan karakteristiknya
   $$\alpha^2 - (1 + w - \emptyset_1 - \emptyset_2)\alpha^1 + w = 0.$$
   Solusi persamaan karakteristik tersebut adalah
   $$\begin{aligned} \alpha_1 &= \frac{(1+w-\emptyset_1-\emptyset_2)+\gamma}{2} \\ \alpha_2 &= \frac{(1+w-\emptyset_1-\emptyset_2)-\gamma}{2} \end{aligned} \qquad (2.9)$$
   dengan $\gamma = \sqrt{(1 + w - \emptyset_1 - \emptyset_2)^2 - 4w}$.
   Karena $\alpha_1 \neq \alpha_2$ maka solusi relasi rekurensi homogen berbentuk
   $$x_t^{(h)} = k_1 {\alpha_1}^t + k_2 {\alpha_2}^t. \qquad (2.10)$$
   Nilai $\varphi(n) = \emptyset_1 p_b + \emptyset_2 p_g$ adalah konstan sehingga bentuk solusi coba juga berupa konstanta, misalkan A_0 dengan demikian solusi relasi rekurensi nonhomogennya berbentuk
   $$x_t = k_1 {\alpha_1}^t + k_2 {\alpha_2}^t + A_0. \qquad (2.11)$$
2. Mencari solusi khusus relasi rekurensi nonhomogen.
   Syarat awal x_0, x_1 dan x_2 diperoleh dari solusi relasi rekurensi nonhomogen dengan memasukkan $t = 0$, 1 dan 2 masing-masing x_0, x_1 dan x_2, sehingga diperoleh sebuah sistem persamaan linier
   $$\begin{aligned} x_0 &= k_1 + k_2 + A_0 \\ x_1 &= k_1 {\alpha_1}^1 + k_2 {\alpha_2}^1 + A_0 \\ x_2 &= k_1 {\alpha_1}^2 + k_2 {\alpha_2}^2 + A_0. \end{aligned}$$
   Dengan melakukan eliminasi diperoleh
   $$\begin{aligned} A_0 &= \frac{\alpha_1\alpha_2 x_0 - x_1(\alpha_1+\alpha_2)+x_2}{(\alpha_1-1)(\alpha_2-1)} \\ k_1 &= \frac{\alpha_1(x_0-x_1)-x_1+x_2}{(\alpha_1-\alpha_2)(\alpha_1-1)} \\ k_2 &= \frac{\alpha_1(x_1-x_0)+x_1-x_2}{(\alpha_1-\alpha_2)(\alpha_2-1)}. \end{aligned} \qquad (2.12)$$

Karena $x_2 = (1 + w + \emptyset_1 - \emptyset_2)x_1 - wx_0 + \emptyset_1 p_b + \emptyset_2 p_g$, $\alpha_1 - \alpha_2 = \gamma$ dan $w = \frac{(1+w-\emptyset_1-\emptyset_2)^2-4\gamma}{2}$, maka persamaan (2.12) menjadi

$$A_0 = \frac{\emptyset_1 p_b + \emptyset_2 p_g}{\emptyset_1 + \emptyset_2}$$

$$k_1 = \frac{\alpha_1(x_0 - x_1) - x_1 + x_2}{\gamma(\alpha_1 - 1)} \quad (2.13)$$

$$k_2 = \frac{\alpha_1(x_1 - x_0) + x_1 - x_2}{\gamma(\alpha_2 - 1)}.$$

Dengan demikian solusi relasi rekurensi nonhomogennya berbentuk

$$x_t = \frac{\alpha_1(x_0 - x_1) - x_1 + x_2}{\gamma(\alpha_1 - 1)} {\alpha_1}^t + \frac{\alpha_1(x_1 - x_0) + x_1 - x_2}{\gamma(\alpha_2 - 1)} {\alpha_2}^t + \frac{\emptyset_1 p_b + \emptyset_2 p_g}{\emptyset_1 + \emptyset_2} \quad (2.14)$$

yang merupakan bentuk eksplisit dari posisi partikel pada waktu ke-t.

(Dianto, 2009)

2.10.3 Penentuan kisaran bobot inersia (*w*)

Berdasarkan bentuk eksplisit posisi partikel pada waktu ke-t diketahui bahwa x_t konvergen ke $\frac{\emptyset_1 p_b + \emptyset_2 p_g}{\emptyset_1 + \emptyset_2}$ atau dapat ditulis

$$\lim_{t \to \infty} x_t = \frac{\emptyset_1 p_b + \emptyset_2 p_g}{\emptyset_1 + \emptyset_2} \quad (2.15)$$

jika $\lim_{t \to \infty} {\alpha_1}^t = \lim_{t \to \infty} {\alpha_2}^t = 0$. Nilai $\lim_{t \to \infty} {\alpha_1}^t$ dan $\lim_{t \to \infty} {\alpha_2}^t$ akan bernilai nol jika $|\alpha_1| < 1$ dan $|\alpha_2| < 1$.

Berdasarkan persamaan (2.9) diketahui bahwa nilai α_1 dan α_2 bergantung pada nilai γ , sehingga untuk menentukan kisaran parameter *w* dapat dibagi menjadi 3 kasus, yaitu:

1. Nilai γ bernilai real $\gamma > 0$, yaitu $(1 + w - \emptyset_1 - \emptyset_2)^2 - 4w > 0$, sehingga $1 + w - \emptyset_1 - \emptyset_2 > 2\sqrt{w}$ atau $1 + w - \emptyset_1 - \emptyset_2 < -2\sqrt{w}$ yang dapat menyebabkan $\alpha_1 > 1$, karena $1 + w - \emptyset_1 - \emptyset_2 > 2\sqrt{w}$ atau $\alpha_2 < -1$ karena $1 + w - \emptyset_1 - \emptyset_2 < -2\sqrt{w}$ sehingga algoritma PSO tidak konvergen.

2. Nilai γ bernilai nol, yaitu $(1+w-\emptyset_1-\emptyset_2)^2-4w=0$, sehingga $\alpha_1=\alpha_2=\frac{1+w-\emptyset_1-\emptyset_2}{2}$, yang menyebabkan solusi umum rekurensi nonhomogen menjadi

$$x_t = k_1{\alpha_1}^t + tk_2{\alpha_2}^t + A_0.$$

Adanya faktor t menyebabkan x_t tidak konvergen sehingga algoritma PSO tidak konvergen

3. Nilai γ bernilai kompleks, yaitu $(1+w-\emptyset_1-\emptyset_2)^2-4w<0$, sehingga

$$\alpha_1=\frac{1+w-\emptyset_1-\emptyset_2+i\sqrt{4w-(1+w-\emptyset_1-\emptyset_2)^2}}{2}$$

$$\alpha_2=\frac{1+w-\emptyset_1-\emptyset_2-i\sqrt{4w-(1+w-\emptyset_1-\emptyset_2)^2}}{2}.$$

Nilai α_1 dan α_2 dapat dinyatakan sebagai $\alpha_1=\eta(\cos\theta+i\sin\theta)$ dan $\alpha_2=\eta(\cos\theta-i\sin\theta)$ dengan $\eta=\sqrt{\frac{(1+w-\emptyset_1-\emptyset_2)^2}{4}+\frac{4w-(1+w-\emptyset_1-\emptyset_2)^2}{4}}=\sqrt{w}$ dan $\theta=\arg(\alpha_1)=\arg(\alpha_2)$, sehingga ${\alpha_1}^n=\eta^n(\cos(n\theta)+$ *i*sin*(nθ)* dan *α2n=ηn*cos*nθ−i*sin*(nθ).*

$\lim_{n\to\infty}{\alpha_1}^n$ dan $\lim_{n\to\infty}{\alpha_2}^n$ akan sama dengan nol jika nilai $\eta<1$. Karena $\eta=\sqrt{w}$ maka $\sqrt{w}<1$, dengan demikian nilai $\sqrt{w}<1$ akan menyebabkan $\lim_{n\to\infty}{\alpha_1}^n=0$ dan $\lim_{n\to\infty}{\alpha_2}^n=0$ sehingga $\lim_{t\to\infty}x_t=\frac{\emptyset_1 p_b+\emptyset_2 p_g}{\emptyset_1+\emptyset_2}$ atau algoritma PSO akan konvergen.

(Dianto, 2009)

2.10.4 Kisaran Nilai koefisien percepatan (c_1 dan c_2)

Nilai $\sqrt{w}<1$ akan menyebabkan algoritma PSO konvergen, sehingga berdasarkan $(1+w-\emptyset_1-\emptyset_2)^2-4w<0$ akan dapat diketahui kisaran parameter $\emptyset_1$ dan $\emptyset_2$. Dari $(1+w-\emptyset_1-\emptyset_2)^2-4w<0$ diketahui bahwa $1+w-2\sqrt{w}<\emptyset_1+\emptyset_2<1+w+2\sqrt{w}$, karena $\sqrt{w}<1$ maka kisaran nilai $\emptyset_1+\emptyset_2$ adalah $0<\emptyset_1+\emptyset_2<4$. Karena $\emptyset_1=c_1r_1$, $\emptyset_2=c_2r_2$, $0\le r_1\le 1$ dan $0\le r_2\le 1$ maka $0<c_1+c_2<4$.

(Dianto, 2009)

2.10.5 Kriteria Penghentian

1. *Error*

   Syarat penghentian iterasi algoritma PSO bermacam-macam. Jika nilai optimum fungsi tujuan diketahui misalkan $\boldsymbol{x}^*$ maka kriteria dapat dihentikan dengan $|f(\boldsymbol{x_i}) - f(\boldsymbol{x}^*)| \leq \varepsilon$, untuk ε adalah toleransi *error* yang dapat diperoleh. Jika *error* terlalu besar maka nilai optimum yang dicapai akan kurang baik, sebaliknya jika *error* terlalu kecil maka kemungkinan iterasi tidak akan berhenti. Hal ini dikarenakan algoritma PSO sulit untuk menemukan titik yang benar-benar optimal sesuai dengan nilai eksaknya.

2. Perbaikan

   Iterasi dihentikan jika tidak ada perbaikan yang diamati dalam beberapa iterasi. Bila kecepatan yang dihasilkan terlalu kecil maka perubahan posisi setiap partikel terlalu kecil sehingga dapat ditarik kesimpulan bahwa partikel telah berkumpul.

3. Maksimum iterasi

   Iterasi berhenti jika jumlah iterasi telah mencapai maksimum iterasi. Maksimum iterasi juga membatasi waktu komputasi yang dilakukan. Jika maksimum iterasi terlalu besar maka waktu komputasi yang akan semakin lama, tetapi jika maksimum iterasi terlalu kecil maka mungkin iterasi berhenti sebelum mencapai solusi optimal.

(Engelbretch, 2007)

BAB III
PEMBAHASAN

Pada bab ini, yang pertama akan membahas tentang permasalahan *multidimensional knapsack 0/1*. Kedua, tentang pembentukan algoritma PSO untuk menyelesaikan permasalahan *multidimensional knapsack 0/1*, dan yang ketiga membahas tentang perbandingan penyelesaian permasalahan *multidimensional knapsack 0/1* dengan menggunakan algoritma DBPSO, MBPSO, dan HBPSO.

1.4 Permasalahan *Multidimensional Knapsack 0/1*

Permasalahan *multidimensional knapsack 0/1* atau *multidimensional rucksack 0/1* adalah permasalahan optimasi kombinatorial, ilustrasinya adalah : "Diberikan sekumpulan barang, yang mempunyai atribut, kendala dan nilai, harus dapat ditentukan jumlah dari masing-masing barang untuk dimasukan dalam sebuah tas sehingga total kendalanya kurang dari sama dengan batas kendala yang telah ditentukan, dan total nilai barangnya harus sebesar mungkin." dengan kata lain : "Diberikan n macam barang, $i = 1, \dots, n$ dan c macam kendala, $k = 1, \dots, c$. Tiap barang i mempunyai nilai a_i dan kendala b_{ki}, yang mana $a_i, b_{ki} > 0$. Kapasitas maksimal tas untuk menampung barang adalah B_k. Kemudian akan ditentukan barang yang akan dimasukkan kedalam tas sehingga nilainya maksimal."

Secara matematis

Memaksimalkan $\sum_{i=1}^{n} a_i \, x_i$ (3.1)

Dengan syarat $\sum_{i=1}^{n} b_{ki} x_i \leq B_k$ (3.2)

$x_i \in \{0,1\}, \quad k = 1, \dots, c,$ dan c adalah banyak kendala.

Contoh 3.1 :

Misalkan diketahui 5 buah barang dengan atribut yaitu harga, berat dan volum :

Tabel 3.1 Daftar barang beserta atribut-atributnya

No (i)	Barang (x)	Harga (a)	Berat (b_1)	Volum (b_2)
1	Laptop	\$ 15	5 kg	5000 cm^3
2	Setrika	\$ 5	3 kg	1000 cm^3
3	TV	\$ 10	7 kg	64000 cm^3
4	Radio	\$ 3	2 kg	750 cm^3
5	Kotak kayu	\$ 10	4 kg	16000 cm^3

dan sebuah tas dengan kapasitas berat maksimal 10 kg dan volum 2000 cm^3. Tentukan barang yang harus dimasukkan kedalam tas sehingga harganya menjadi maksimal ?

Jawab :

Model matematika dari contoh 3.1 adalah :

$$\max \text{ harga} = 15x_1 + 5x_2 + 10x_3 + 3x_4 + 10x_5$$

Dengan syarat,

$$5x_1 + 3x_2 + 7x_3 + 2x_4 + 4x_5 \leq 10$$

$$5000x_1 + 1000x_2 + 64000x_3 + 750x_4 + 16000x_5 \leq 2000$$

$$x_i \in \{0,1\} \qquad i = 1, 2, 3, 4, 5 \qquad j=1,2$$

Solusi didapatkan dengan cara mengganti nilai-nilai x_i dengan nilai {0,1} sehingga harga menjadi maksimum, beratnya dan volumnya tidak boleh melebihi kapasitas. Nilai x_i yang paling tepat untuk contoh soal 3.1 adalah :

Tabel 3.2 Cara pengerjaan dan solusi permasalahan *multidimensional knapsack 0/1*

No (i)	**Harga (a)/\$**	**Berat (b_1)/kg**	**Volum (b_2)/cm^3**	**Solusi (x)**	**Solusi berat**	**Solusi volum**	**Solusi harga**
1	15	5	5000	0	0	0	0
2	5	3	1000	1	3	1000	5
3	10	7	64000	0	0	0	0
4	3	2	750	1	2	750	3
5	10	4	16000	0	0	0	0
Total					**5**	**1750**	**8**

Jadi barang yang harus dimasukkan kedalam tas sehingga harganya maksimal adalah setrika dan radio.

3.2 Algoritma *Particle Swarm Optimization (*PSO*)* untuk menyelesaikan Permasalahan *Multidimensional Knapsack 0/1*

3.2.1 Proses Algoritma Secara Umum

Langkah 1 : Melakukan inisialisasi awal, meliputi banyak partikel (*m*), banyak data (*n*), iterasi maksimum (*maxiter*), posisi* partikel (x) secara acak*, kecepatan partikel (v) secara acak*, konstanta (c_1 dan c_2), bobot inersia (w), *pbest* (matrik nol ukuran m × n), dan *gbest* (matrik nol ukuran 1 × n).

*acak = pengacakan nilai berdasarkan distribusi *uniform* dengan parameter (0,1).
*posisi = posisi partikel diasumsikan keras/tegas yaitu bernilai 0 atau 1. Mengikuti persamaan :

$$x_{ij} = \begin{cases} 0, rand(0,1) < 0{,}5 \\ 1, rand(0,1) \geq 0{,}5 \end{cases} \quad \begin{matrix} i=1,\dots,n \\ j=1,\dots,m \end{matrix} \qquad (3.3)$$

Langkah 2 : Menghitung nilai *fitness*

Nilai *fitness* yang dimaksud adalah nilai barang maksimal yang dapat dimasukkan, dengan syarat tidak boleh melebihi kendala-kendala yang telah ditentukan.

Nilai barang

$$f_j(a,x) = \sum_{i=1}^{n} a_i\, x_i \qquad j = 1,\dots,m \qquad (3.4)$$

Nilai kendala

$$h_j(b_k,x) = \sum_{i=1}^{n} b_{ki}\, x_i \quad j = 1,\dots,m \;\; k = 1,\dots,c \qquad (3.5)$$

Nilai *fitness*

$$f_j(a,x) = \begin{cases} 0, & h_j(b_k,x) > B_k \quad j = 1,\dots,m \\ f_j(a,x), & h_j(b_k,x) \leq B_k \quad k = 1,\dots,c \end{cases} \qquad (3.6)$$

Langkah 3 : Mencari nilai *pbest*

Membandingkan nilai *fitness*, $f_j(a,x) \;\; j = 1,\dots,m$. Nilai *fitness* terbesar pada partikel ke-*j* akan menjadi solusi sementara. Memperbaiki matrik *pbest* pada baris ke-*j* dengan nilai *x* pada baris ke-*j*.

Langkah 4 : Mencari nilai *gbest*

Pada iterasi pertama, nilai *gbest* sama dengan nilai *fitness* terbesar (solusi sementara). Pada iterasi selanjutnya nilai *gbest* diambil dengan cara membandingkan nilai *gbest* pada iterasi ini dengan nilai *gbest* pada iterasi sebelumnya. Nilai yang terbesar akan menjadi *gbest* dan menjadi solusi terbaik pada akhir iterasi.

Langkah 5 : Memperbaharui kecepatan dan posisi partikel dengan mengikuti persamaan :

$$v_j(t+1) = v_j(t) \times w + c_1 \times r_1 \times \left(pbest_j - x_j(t)\right) + c_2 \times r_2 \times (gbest_j - x_j(t)) \qquad (3.7)$$

$$x_j(t+1) = x_j(t) + v_j(t+1) \qquad (3.8)$$

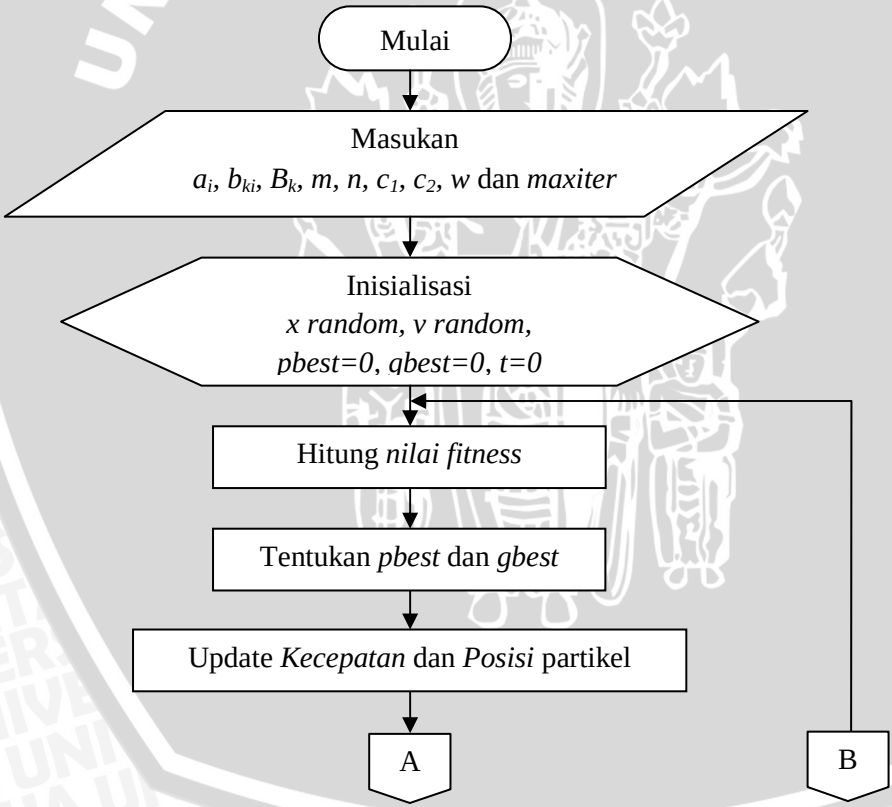
Karena posisi partikel harus keras/tegas (0/1) maka nilai posisi partikel terbaru mengikuti persamaan :

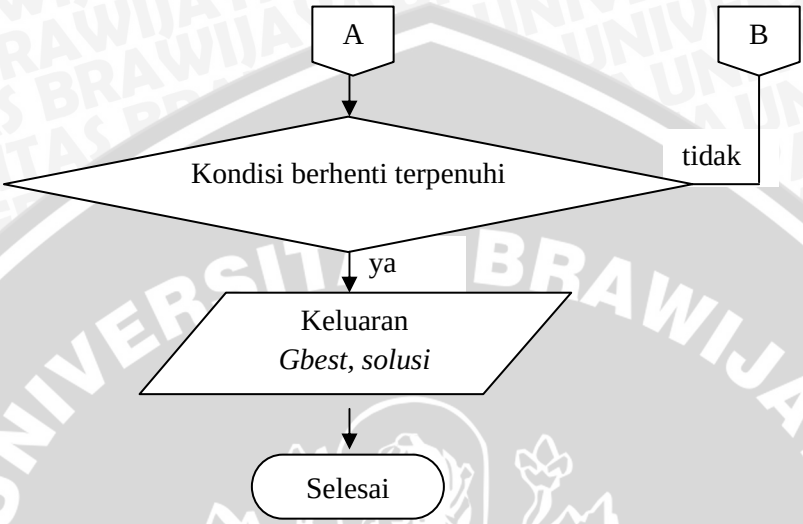
$$x_{ij} = \begin{cases} 0, x_j(t+1) < 0{,}5 \\ 1, x_j(t+1) \geq 0{,}5 \end{cases} \quad \begin{matrix} \text{i=1,...,n} \\ \text{j=1,...,m} \end{matrix} \qquad (3.9)$$

Terdapat variabel r_1 dan r_2 pada persamaan kecepatan, nilai dari variabel tersebut diambil secara acak mengikuti distribusi *uniform* dengan parameter (0,1).

Langkah 6 : Mengulangi langkah 2 sampai langkah 5 hingga nilai iterasi sama dengan maksimum iterasi.

Proses *flowchart* secara umum algoritma PSO untuk menyelesaikan permasalahan *multidimensional knapsack 0/1* dapat digambarkan sesuai Gambar 3.1.





Gambar 3.1 *Flowchart* algoritma PSO secara umum

3.2.2 Algoritma *Discrete Binary PSO* (DBPSO)

Algoritma DBPSO diperkenalkan oleh Eberhart dan Kennedy pada tahun 1998. Cara kerja algoritma DBPSO hampir sama dengan algoritma PSO secara umum, akan tetapi persamaan perbaikan posisi (langkah 5, persamaan 3.8, persamaan 3.9) ditentukan oleh *sigmoid limiting transformation* sesuai dengan persamaan 3.10 dan persamaan 3.11.

$$S(v_{ij}) = \frac{1}{1+e^{-v_{ij}}} \tag{3.10}$$

$$x_{ij} = \begin{cases} 0, & r < S(v_{ij}) \\ 1, & r \geq S(v_{ij}) \end{cases} \tag{3.11}$$

yang mana, $S(v_{ij})$ adalah *sigmoid limiting transformation* dan r adalah nilai acak U(0,1).

3.2.3 Algoritma *Modified Binary PSO* (MBPSO)

Algoritma MBPSO diperkenalkan oleh Shen Qi dan Jiang pada tahun 2004. Algoritma MBPSO merupakan perbaikan dari algoritma DBPSO. Persamaan perbaikan posisi didefinisikan sebagai persamaan 3.12.

$$x_{ij} = \begin{cases} x_{ij} \text{ sebelumnya}, & 0 < v_{ij} \leq \alpha \\ p_{ij}, & \alpha < v_{ij} \leq \frac{1}{2}(1+\alpha) \\ g_{ij}, & \frac{1}{2}(1+\alpha) < v_{ij} \leq 1 \end{cases} \quad (3.12)$$

yang mana, α adalah *static probability*, nilai awalnya adalah 0,5 kemudian selanjutnya dibangkitkan secara acak U(0,1).

3.2.4 Algoritma *Hybrid Binary PSO* (HBPSO)

Algoritma HBPSO diperkenalkan oleh Muhammad Ilyas Menhas, Min Rui Fei, Ling Wang dan Xiping Fu pada tahun 2011 dalam jurnalnya yang bejudul "*A Novel Hybrid Binary PSO Algorithm*". Algoritma HBPSO merupakan gabungan dari algoritma PSO dengan beberapa konsep algoritma genetika (c*rossover* dan mutasi). Penerapan konsep algoritma genetika dalam algoritma HBPSO digunakan untuk menambah tingkat eksplorasi solusi global. Algoritma HBPSO terdiri dari 3 tahap perbaikan nilai posisinya, yaitu :

1. Algoritma DBPSO
   Menentukan posisi menggunakan persamaan 3.10 dan 3.11.
2. *Crossover*
   Merupakan adaptasi dari algoritma genetika. Pada algoritma genetika, *crossover* digunakan untuk membangkitkan kandidat solusi baru untuk generasi selanjutnya. Analog dengan algoritma genetika, istilah *crossover* pada algoritma HBPSO digunakan untuk perpindahan solusi dari solusi x_{ij} ke solusi p_{ij} atau g_{ij}. Proses *crossover* dapat dituliskan dengan persamaan :

   $$x_{ij} = \begin{cases} x_{ij}, & 0 \leq r_{ij} \leq \alpha \\ p_{ij}, & \alpha < r_{ij} \leq 2\alpha \\ g_{ij}, & 2\alpha < r_{ij} \leq 1 \end{cases} \quad (3.13)$$

   yang mana, r adalah bilangan acak U(0,1) dan α adalah persentase rasio tingkat kemungkinan *crossover* terjadi. Nilai α pada algoritma HBPSO adalah 33,33%.

Contoh 3.2 :

Misalkan pada iterasi sebelumnya didapatkan solusi :

p_{ij} :	0	0	0	1

g_{ij} :	1	0	1	0

Dan nilai x_{ij} pada iterasi saat ini adalah

x_{ij} :	1	0	1	1

Kemudian bilangan acak yang dibangkitkan (r_{ij}) adalah 0,5. Karena nilai α pada algoritma HBPSO adalah 33,33% maka nilai x_{ij} masuk ke dalam *range* $\alpha < r_{ij} \leq 2\alpha$ dan nilainya bukan nilai x_{ij} lagi melainkan harus diganti dengan nilai p_{ij}.

x_{ij} :	0	0	0	1

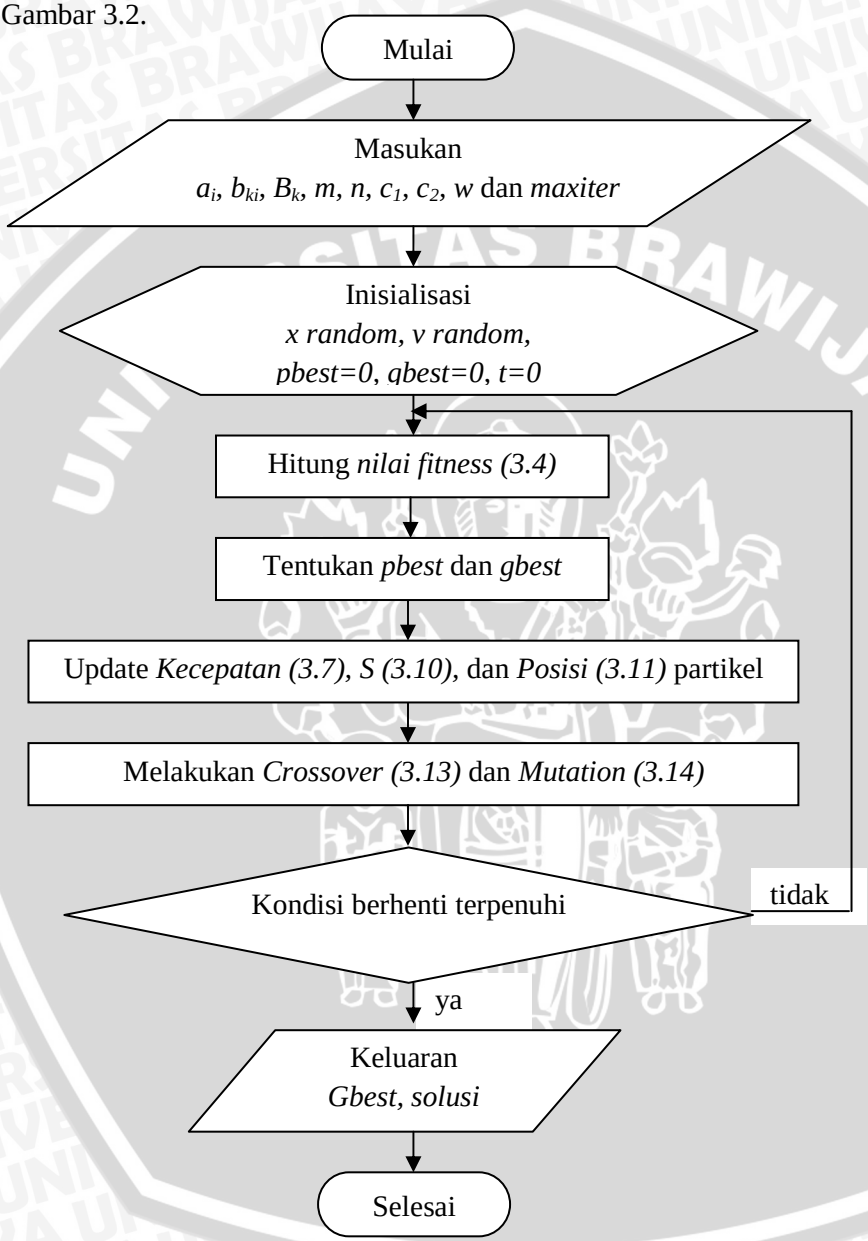
3. Mutasi (*Mutation*)

Mutasi juga biasa digunakan pada algoritma genetika untuk menjaga keberagaman populasi. Pada algoritma HBPSO mutasi ditambahkan dengan dua tujuan. Pertama, untuk menjaga keberagaman dari populasi. Yang kedua untuk membantu algoritma agar tidak konvergen terjebak pada optimum lokal. Proses mutasi dapat ditunjukkan oleh persamaan 3.14.

$$x_{ij} = \begin{cases} \overline{x_{ij}}, & r_{ij} \leq mut \\ x_{ij}, & \text{selainnya} \end{cases} \quad (3.14)$$

yang mana, r adalah bilangan acak U(0,1), mut adalah kemungkinan mutasi, dan , $\overline{x_{ij}}$ adalah *invers* biner dari x_{ij} (artinya invers dari 0 adalah 1 dan sebaliknya).

Proses *flowchart* dari algoritma HBPSO digambarkan sesuai Gambar 3.2.



Gambar 3.2 *Flowchart* algoritma HBPSO

Sedangkan *pseudocod*e dari algoritma HBPSO adalah sebagai berikut :

Masukan *knapsack* : a_i, b_{ki}, kendala, dan B_k
PSO : m, n, c_1, c_2, w dan $maxiter$
GA : alpha, mut

1. $x_{mn} \leftarrow$ round(rand(0,1))
2. $v_{mn} \leftarrow$rand(0,1)
3. $pbest_{mn} \leftarrow 0$
4. $gbest_{1n} \leftarrow 0$
5. solusilaba $\leftarrow 0$
6. nilaimax $\leftarrow 0$
7. t $\leftarrow 0$
8. while t < maxiter
   a. t $\leftarrow$ t + 1
   b. for i=1 to m do
      1) $fx_i \leftarrow 0$
      2) cek $\leftarrow 0$
      3) for j=1 to kendala do
         a) $fc_j \leftarrow 0$
         b) nilai $\leftarrow 0$
         c) for k=1 to n do
            ①. nilai $\leftarrow$ nilai + a_{1k} * x_{ik}
            ②. $fc_j \leftarrow fc_j + c_{jk} * x_{ik}$
         d) if $fc_j > b_j$
            ①. cek $\leftarrow$ 1
      4) if cek =1
         a) $fx_i \leftarrow 0$ else $fx_i \leftarrow$ nilai
   c. [nilai, posisipbest] $\leftarrow$ max(fx)
   d. $pbest_{posisipbest:} \leftarrow x_{posisipbest:}$
   e. if nilai>nilaimax
      1) nilaimax $\leftarrow$ nilai
      2) $gbest_{1:} \leftarrow x_{posisipbest:}$

f. for i=1 to m do
  1) if i<>posisipbest
    a) for j=1 to n do
      ①. r1 ← rand(0,1)
      ②. r2 ← rand(0,1)
      ③. $v_{ij} \leftarrow w * v_{ij} + c1 * r1 *(pbest_{ij} - x_{ij}) + c2 * r2 * (gbest_{1j} - x_{ij})$
      ④. $s_{ij} \leftarrow 1/(1+\exp(-v_{ij}))$
      ⑤. r ← rand(0,1)
      ⑥. if $r \leq s_{ij}$
        i. $x_{ij} \leftarrow 1$ else $x_{ij} \leftarrow 0$
      ⑦. if $0 \leq r \leq$ alpha
        i. $x_{ij} \leftarrow x_{ij}$ else if alpha < $r \leq 2 *$ alpha
          01. $x_{ij} \leftarrow pbest_{ij}$ else if $2 *$ alpha < $r \leq 1$
            One. $x_{ij} \leftarrow gbest_{1j}$
      ⑧. if $r \leq$ mut
        i. $x_{ij} \leftarrow$ □□□ else $x_{ij} \leftarrow x_{ij}$
  Keluaran solusi terbaik

3.3 Perbandingan Algoritma DBPSO, MBPSO dan HBPSO dalam menyelesaikan Permasalahan *Multidimensional Knapsack 0/1*

3.3.1 Data Uji dan Parameter yang digunakan

Untuk mengetahui tingkat keefektifan varian dari algoritma PSO ini diperlukan suatu data uji. Data uji diambil dari *OR-Library test data sets* yang diakses dari situs http://people.brunel.ac.uk/~mastjjb/jeb/info. Data yang akan digunakan adalah data *mknap1.txt* dipilih 4 buah data dengan spesifikasi seperti Tabel 3.3 :

Tabel 3.3 Spesifikasi data uji *mknap1.txt*

Data Uji	Jumlah Data	Jumlah Kendala	Solusi Eksak
Data 1	10	10	8706,1
Data 2	20	10	6120
Data 3	28	10	12400
Data 4	39	5	10618

Kemudian parameter-parameter algoritma DBPSO, MBPSO, dan HBPSO yang akan digunakan dalam pengujian data uji adalah sebagai berikut :

Tabel 3.4 Parameter uji algoritma DBPSO, MBPSO, dan HBPSO

Parameter	Swarm size	maxiter	c1 & c2	w	alpha	mut
DBPSO	30	100 1000 5000 8000	2	1	-	-
MBPSO					-	-
HBPSO					33 %	0.05

Pengujian dilakukan dengan menggunakan program yang dibuat dengan bahasa pemrogram *Matlab 7*. Program tersebut dijalankan pada komputer jinjing yang memiliki spesifikasi : Intel(R) Core(TM)2 Duo T5800 @ 2.00GHz, RAM 2GB, dan *harddisc* 250GB.

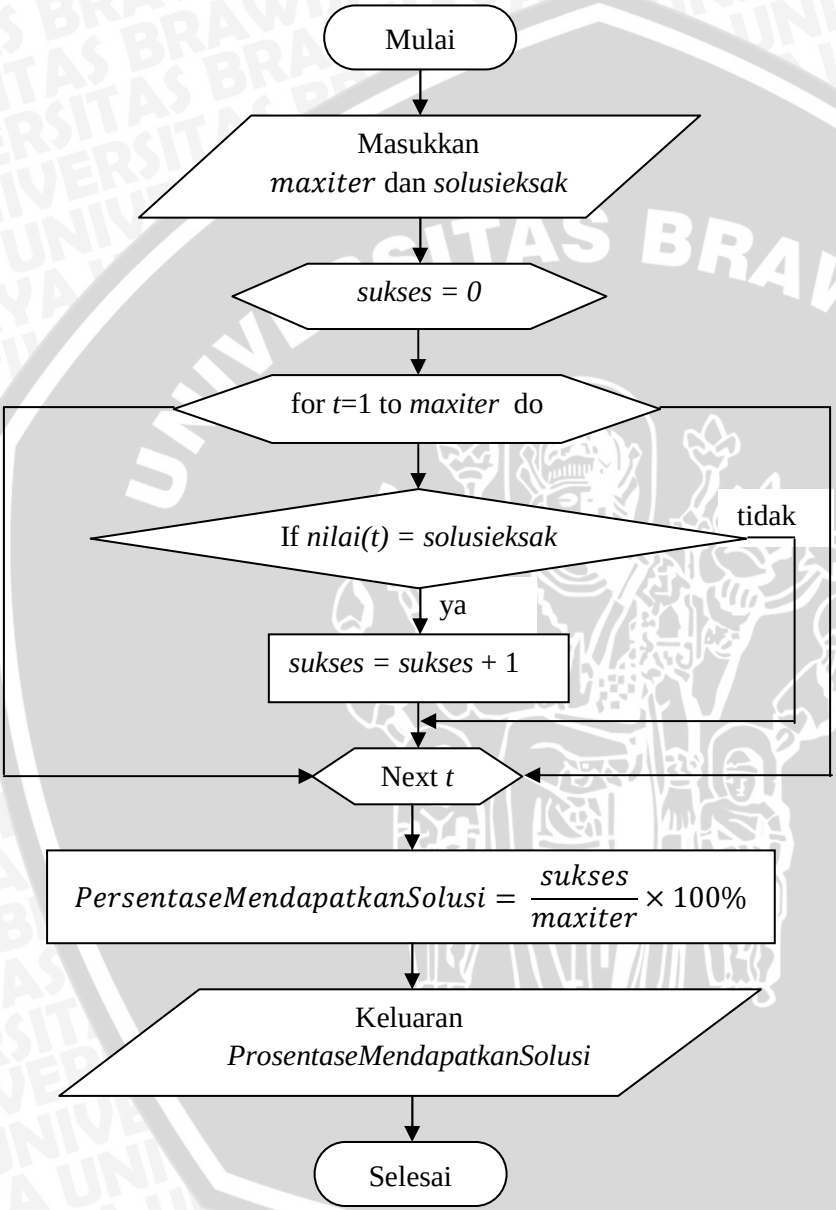
3.3.2 Analisis Algoritma

Analisis ini bertujuan untuk melihat perbandingan tingkat keakuratan solusi ketiga algoritma untuk berbagai macam permasalahan *multidimensional knapsack 0/1* yang akan diujikan. Teknik analisis yang akan dilakukan adalah dengan menyelesaikan tiap-tiap data uji dengan algoritma DBPSO, MBPSO, dan HBPSO. Kemudian menampilkan hasilnya kedalam grafik, solusi maksimal, total solusi dan melihat persentase mendapatkan solusi eksaknya. Skema pengujian dapat dilihat seperti Tabel 3.5 :

Tabel 3.5 Skema pengujian algoritma DBPSO, MBPSO, dan HBPSO

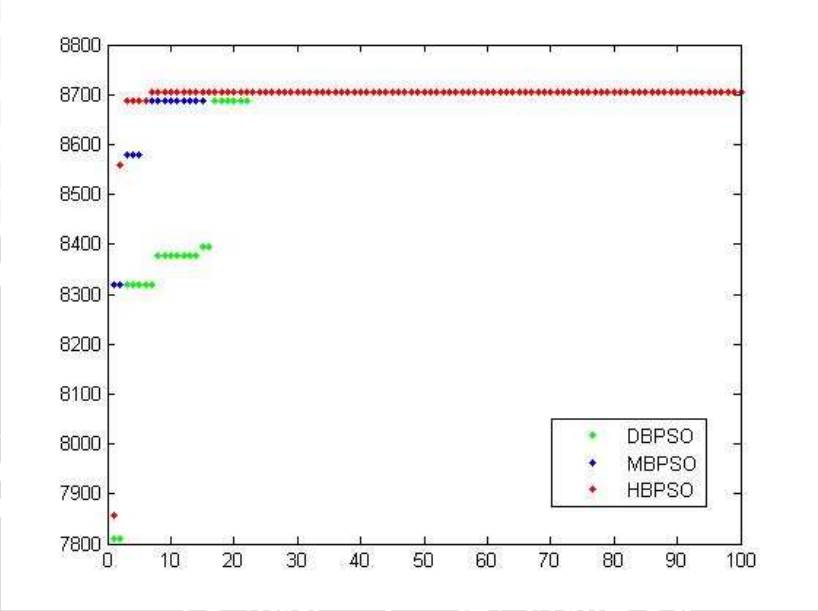
Uji	**Algoritma**	**Maxiter**	**Hasil**	
Data 1	DBPSO	100	Gambar 3.4	Tabel 3.6
	MBPSO			
	HBPSO			
Data 2	DBPSO	1000	Gambar 3.5	Tabel 3.7
	MBPSO			
	HBPSO			
Data 3	DBPSO	5000	Gambar 3.6	Tabel 3.8
	MBPSO			
	HBPSO			
Data 4	DBPSO	8000	Gambar 3.7	Tabel 3.9
	MBPSO			
	HBPSO			

Proses analisis dapat digambarkan seperti Gambar 3.3 :



Gambar 3.3 Proses analisis keakuratan program

1. Uji Data 1



Gambar 3.4 Grafik solusi maksimal tiap iterasi program DBPSO, MBPSO, dan HBPSO pada Data 1

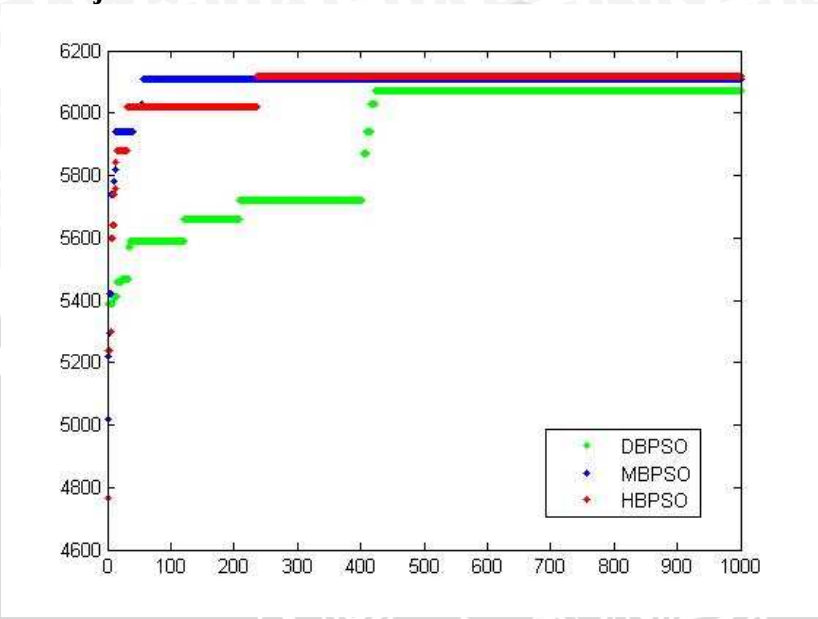
Tabel 3.6 Hasil pengujian pada Data 1

Algoritma	DBPSO	MBPSO	HBPSO
Solusi Maks	8706,1	8706,1	8706,1
PMS	78%	85%	94%

Dari Gambar 3.4 dapat dilihat bahwa pada iterasi awal ketiga algoritma tersebut berangkat pada nilai solusi yang berlainan. Perbedaan nilai solusi ini disebabkan oleh pemilihan inisialisasi nilai posisi dan kecepatan tiap-tiap partikel di awal iterasi yang dilakukan secara acak oleh komputer. Akan tetapi setelah mendekati iterasi terakhir ketiga algoritma tersebut akan cenderung konvergen ke satu nilai.

Dari Tabel 3.6 dapat diketahui bahwa ketiga algoritma tepat mendapatkan solusi eksak yaitu 8706,1. Akan tetapi, nilai PMS (Persentase Mendekati Solusi) terbesar jatuh pada algoritma HBPSO dengan nilai 94%. Artinya algoritma lebih sering menemukan solusi eksak di tiap iterasinya dibandingkan dengan algoritma lainnya.

2. Uji Data 2



Gambar 3.5 Grafik solusi maksimal tiap iterasi program DBPSO, MBPSO, dan HBPSO pada Data 2

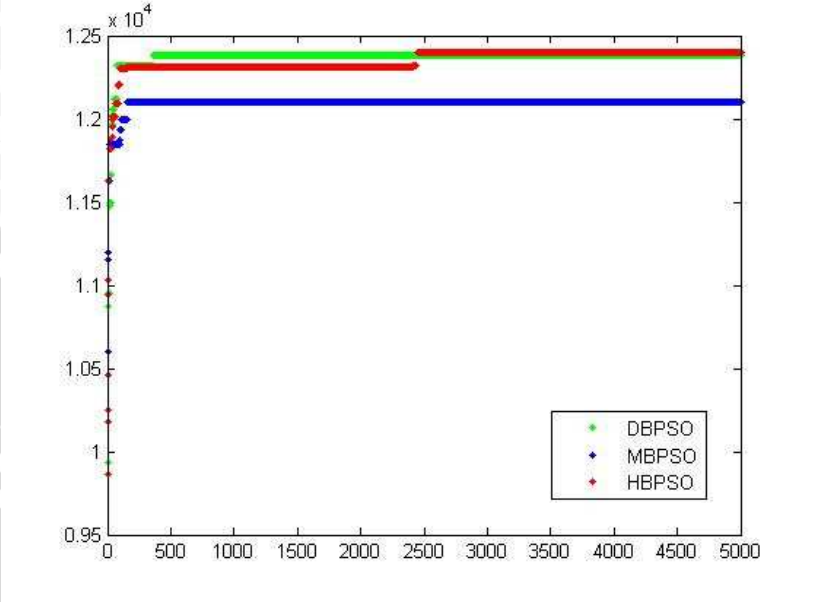
Tabel 3.7 Hasil pengujian pada Data 2

Algoritma	DBPSO	MBPSO	HBPSO
Solusi Maks	6070	6110	6120
PMS	0%	0%	76,4%

Dari Gambar 3.5 dapat dilihat bahwa nilai solusi algoritma HBPSO lebih baik daripada algoritma DBPSO dan MBPSO pada iterasi ke ±250. Hal ini disebabkan karena Algoritma HBPSO mempunyai kemampuan yang lebih baik untuk meningkatkan solusinya.

Dari Tabel 3.7 algoritma yang tepat mendapatkan solusi eksak adalah algoritma HBPSO yaitu 6120. Nilai PMS (Persentase Mendekati Solusi) terbesar jatuh pada algoritma HBPSO dengan nilai 76,4%. Artinya algoritma lebih sering menemukan solusi eksak di tiap iterasinya dibandingkan dengan algoritma lainnya.

3. Uji Data 3



Gambar 3.6 Grafik solusi maksimal tiap iterasi program DBPSO, MBPSO, dan HBPSO pada Data 3

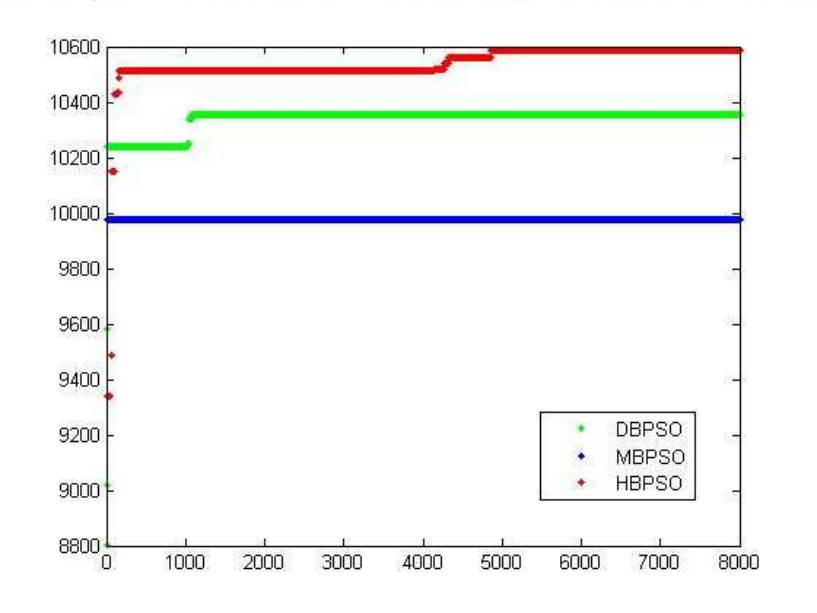
Tabel 3.8 Hasil pengujian pada Data 3

Algoritma	DBPSO	MBPSO	HBPSO
Solusi Maks	12380	12100	12400
PMS	0%	0%	51,16%

Dari Gambar 3.6 dapat dilihat bahwa pada data uji dengan jumlah data dan dimensi yang besar, diperlukan iterasi dalam jumlah yang lebih besar untuk mengetahui tingkat keakuratan algoritma HBPSO. Pada Gambar 3.6 algoritma HBPSO mampu melewati algoritma DBPSO dan MBPSO pada iterasi ±2600.

Dari Tabel 3.8 algoritma yang tepat mendapatkan solusi eksak adalah algoritma HBPSO yaitu 12400 meskipun dengan jumlah iterasi yang besar. Nilai PMS (Persentase Mendekati Solusi) terbesar jatuh pada algoritma HBPSO dengan nilai 51,16%. Artinya algoritma lebih sering menemukan solusi eksak di tiap iterasinya dibandingkan dengan algoritma lainnya.

4. Uji Data 4



Gambar 3.7 Grafik solusi maksimal tiap iterasi program DBPSO, MBPSO, dan HBPSO pada Data 4

Tabel 3.9 Hasil pengujian pada Data 4

Algoritma	DBPSO	MBPSO	HBPSO
Solusi Maks	10355	9977	10588
PMS	0%	0%	0%

Dari Gambar 3.7 dapat dilihat dengan iterasi yang semakin besar grafik dari algoritma HBPSO akan semakin semakin monoton naik dibandingkan dengan algoritma DBPSO dan MBPSO. Dari Gambar 3.7 dapat diketahui bahwa algoritma HBPSO mempunyai lebih banyak peningkatan solusi daripada kedua algoritma lainnya.

Dari Tabel 3.9 diketahui bahwa tidak ada algoritma yang tepat mendapatkan solusi eksak. Meskipun demikian dapat dilihat algoritma yang paling mendekati solusi eksak adalah algoritma HBPSO dengan solusi maksimalnya 10588. Nilai PMS (Persentase Mendekati Solusi) dari ketiga algoritma adalah 0%. Artinya tidak ada algoritma yang mendapatkan solusi eksak.

Secara garis besar, dari keempat pengujian di atas dapat diambil beberapa poin penting. Pertama, nilai solusi pada iterasi awal ketiga algoritma berbeda-beda. Hal ini disebabkan karena nilai awal posisi dan kecepatan ketiga algoritma diinisialisasi secara acak oleh komputer. Kedua, algoritma HBPSO mempunyai kemampuan yang lebih baik untuk meningkatkan solusi ditiap iterasinya dibandingkan dengan algoritma lainnya. Kelebihan algoritma HBPSO ini disebabkan karena terdapat operator genetika didalam algoritma HBPSO yaitu operator *crossover* dan mutasi. Keuntungan yang dapat diambil adalah kemampuan algoritma HBPSO untuk menemukan nilai solusi baru menjadi lebih luas (*search space*-nya lebih luas) sehingga algoritma tidak mudah terjebak pada nilai optimum lokal. Ketiga, diperlukan jumlah iterasi yang besar untuk data dan kendala dengan jumlah yang besar untuk mengetahui tingkat keakuratan algoritma HBPSO. Semakin besar data dan kendala pada permasalahan *multidimensional knapsack 0/1*, maka semakin banyak pula kemungkinan-kemungkinan calon solusinya. Oleh karena itu diperlukan iterasi yang lebih banyak bagi ketiga algoritma tersebut untuk menemukan calon-calon solusi. Akan tetapi ketika algoritma DBPSO dan MBPSO terjebak pada optimum lokal, algoritma HBPSO mampu melewati titik optimum lokal tersebut dengan kelebihan yang dimiliki. Keempat, meskipun belum menemukan solusi eksak, algoritma HBPSO lebih baik dalam menyelesaikan permasalahan *multidimensional knapsack 0/1* dengan jumlah data dan kendala yang rumit dari kedua kedua algoritma lainnya. Dengan iterasi yang cukup banyak, algoritma HBPSO memiliki tingkat eksplorasi yang tinggi terhadap nilai solusi yang didapatkan. Berbeda dengan algoritma DBPSO dan MBPSO yang mempunyai nilai yang cenderung sama dengan nilai solusi sebelumnya sehingga keberagaman nilai solusinya menjadi lebih sedikit dan peluang untuk mendapatkan nilai solusi baru yang sama dengan nilai solusi eksak menjadi lebih kecil. Pada uji data keempat dicukupkan iterasi pada nilai 8000 karena diperlukan waktu yang lama, ± 6 jam untuk menyelesaikan pengerjaan masalah *multidimensional knapsack 0/1* tersebut. Akan tetapi dapat dilihat bahwa algoritma HBPSO mempunyai solusi yang lebih baik dari algoritma DBPSO dan MBPSO.

BAB IV
PENUTUP

4.1 Kesimpulan

Berdasarkan hasil dan pembahasan, dapat disimpulkan :

1. Algoritma PSO dapat diterapkan ke dalam permasalahan *multidimensional knapsack 0/1* dengan merubah posisi partikel manjadi tegas (0/1) dan merubah fungsi *fitness* algoritma PSO sesuai dengan persamaan (3.4) dengan memperhatikan nilai kendala sesuai dengan persamaan (3.5).
2. Algoritma HBPSO mampu menyelesaikan permasalahan *multidimensional knapsack 0/1* lebih baik dalam hal mendapatkan nilai solusi yang sama dengan solusi eksak dibandingkan dengan algoritma DBPSO dan MBPSO. Data hasil dari percobaan dapat disajikan sebagai berikut :

PMS	DBPSO	MBPSO	HBPSO
Data 1	78%	85%	**94%**
Data 2	0%	0%	**76,4%**
Data 3	0%	0%	**51,16%**
Data 4	0%	0%	0%

4.2 Saran

Pada penulisan skripsi terdapat kekurangan pada waktu komputasi untuk mendapatkan solusi akhir yang sama dengan solusi eksak. Oleh karena itu diperlukan pengembangan perbaikan kecepatan dan posisi sehingga iterasi yang digunakan untuk memperoleh solusi eksak tidak terlalu banyak (mempercepat waktu komputasi).

DAFTAR PUSTAKA

Abdelhalim, M. B. dan Habib, S. E. 2009. *Particle Swarm Optimization for HW/SW Partitioning*. Cairo University. Egypt. Halaman 51

Apostol, T. M. 1967. *Calculus Volume I : One-Variable Calculus, with an Introduction to Linear Algebra*. John Wiley & Sons, Inc. USA. Halaman 130

Bronson, R. 1996. *Teori dan Soal-Soal Operation Research, Alih Bahasa : Hans J. Waspakrik*. Erlangga. Jakarta.

Dianto, I. N. 2009. Skripsi "*Fuzzy Adaptive Turbulence Particle Swarm Optimization (Fatpso) Untuk Masalah Optimasi Fungsi Non Linier*". Universitas Brawijaya Malang. Malang. Halaman 17

Engelbretch, A. P. 2007. *Computational Intelligence An Introduce*. University of Pretoria. South Africa. Hal.293-294, 298 dan 312

Glazer, V. 2005. *Introduction to Optimizaton Problems and Mathematical Programming*. http://www.cs.toronto.edu/. Halaman 1

Karpovsky, M. G., Stankovic, R. S., dan Astola, J. T. 2008. *Spectral Logic and Its Applications for the Design of Digital Devices*. John Wiley & Sons, Inc. Halaman 2

Ling, W., Xiuting, W., Jingqi, F., dan Lanlan, Z. *A Novel Probability Binary Particle Swarm Optimization Algorithm and Its Application*. Shanghai University. China. Halaman 28-29

Martello, S., dan Toth, P. 1990. *Knapsack Problem : Algorithm and Computer Implementations*. John Wiley & Sons Ltd. England. Halaman 1-3

Menhas, M. I., MinRui, F., Ling, W., dan Xiping, F. 2011. *A Novel Hybrid Binary PSO Algorithm*. Springer. Berlin-Heidelberg. Halaman 96-97

Parsopaulus, K. E. 2002. *Recent Approach to Global Optimization Problem through Particle Swarm Optimization.* Natural Computing No.1, pp.235-306. Halaman 240

Premalatha, K., dan Natarajan, A. M. 2009. *Hybrid PSO and GA for Global Maximization.* Int. J. Open Problem Comt. Math., Vol. 2, No. 4. Erode. India. Halaman 599

Siang, J. J. 2002. *Matematika Diskrit dan Aplikasinya pada Ilmu Komputer*. Andi Offset. Yogyakarta.

Stroppa, N. 2006. *Algorithm & Complexity – Introduction*. Dublin City University. Irlandia. Halaman 4 dan halaman 9

Wigderson, A. 2006. *P, NP and mathematics – a computational complexity perspective*. Halaman 8 dan Halaman 11

Weise, T. 2009. *Global Optimization Algorithms -Theory and Application-*. http://www.it-weise.de/projects/book.pdf .

Yanbing, L, Linlin, L., Dayang, W., dan Ruijuan, W. 2010. *Optimizing Particle Swarm Optimization to Solve Knapsack Problem*. Hebei Polytechnic University. China. Halaman 437

LAMPIRAN 1
Source Code Program

1. Program DBPSO

No	*Source code*
1	clc
2	clear global
3	%input knapsack
4	a=[600.1 310.5 1800 3850 18.6 198.7 882 4200 402.5 327];
5	c=[20 5 100 200 2 4 60 150 80 40
6	20 7 130 280 2 8 110 210 100 40
7	60 3 50 100 4 2 20 40 6 12
8	60 8 70 200 4 6 40 70 16 20
9	60 13 70 250 4 10 60 90 20 24
10	60 13 70 280 4 10 70 105 22 28
11	5 2 20 100 2 5 10 60 0 0
12	45 14 80 180 6 10 40 100 20 0
13	55 14 80 200 6 10 50 140 30 40
14	65 14 80 220 6 10 50 180 30 50];
15	b=[450
16	540
17	200
18	360
19	440
20	480
21	200
22	360
23	440
24	480];
25	%inisialisasi PSO
26	dim=length(a);
27	const=length(b);
28	m=30;

No	*Source code*
29	maxiter=100;
30	c1=2;
31	c2=2;
32	w=1;
33	sukses=0;
34	%posisi dan kecepatan awal
35	for i=1:m
36	for j=1:dim
37	x(i,j)=round(random('unif',0,1));
38	v(i,j)=random('unif',0,1);
39	end
40	end
41	pbest=zeros(size(v));
42	gbest=zeros(1,dim);
43	nilaimax=0;
44	t=0;
45	while t<maxiter
46	t=t+1
47	for i=1:m
48	fx(i)=0;
49	cek=0;
50	z=0;
51	for j=1:const
52	fc(j)=0;
53	nilai=0;
54	for k=1:dim
55	nilai=nilai+a(1,k)*x(i,k);
56	fc(j)=fc(j)+c(j,k)*x(i,k);
57	end
58	if fc(j)>b(j)
59	cek=1;
60	end

No	*Source code*
61	end
62	if cek==1
63	fx(i)=0;
64	else
65	fx(i)=nilai;
66	end
67	end
68	%menentukan pbest dan gbest
69	[nilai,posisipbest]=max(fx);
70	pbest(posisipbest,:)=x(posisipbest,:);
71	if nilai>nilaimax
72	nilaimax=nilai
73	gbest(1,:)=x(posisipbest,:);
74	end
75	%update kecepatan dan posisi
76	for i=1:m
77	if i~=posisipbest
78	for j=1:dim
79	r1=random('unif',0,1);
80	r2=random('unif',0,1);
81	v(i,j)=w*v(i,j)+c1*r1*(pbest(i,j)-x(i,j))+c2*r2*(gbest(1,j)-x(i,j));
82	s(i,j)=1/(1+exp(-v(i,j)));
83	r=random('unif',0,1);
84	if r<=s(i,j)
85	x(i,j)=1;
86	else
87	x(i,j)=0;
88	end
89	end
90	end
91	end
92	pDBPSO(t)=nilai;

No	*Source code*
93	if nilai==8706.1
94	sukses=sukses+1;
95	end
96	end
97	%prosentase mendapatkan solusi eksak
98	pmsDBPSO=(sukses/maxiter)*100;

2. Program MBPSO

No	*Source code*
1	clc
2	clear global
3	%input knapsack
4	a=[600.1 310.5 1800 3850 18.6 198.7 882 4200 402.5 327];
5	c=[20 5 100 200 2 4 60 150 80 40
6	20 7 130 280 2 8 110 210 100 40
7	60 3 50 100 4 2 20 40 6 12
8	60 8 70 200 4 6 40 70 16 20
9	60 13 70 250 4 10 60 90 20 24
10	60 13 70 280 4 10 70 105 22 28
11	5 2 20 100 2 5 10 60 0 0
12	45 14 80 180 6 10 40 100 20 0
13	55 14 80 200 6 10 50 140 30 40
14	65 14 80 220 6 10 50 180 30 50];
15	b=[450
16	540
17	200
18	360
19	440
20	480
21	200
22	360
23	440

No	*Source code*
24	480];
25	%inisialisasi PSO
26	dim=length(a);
27	const=length(b);
28	m=30;
29	maxiter=100;
30	c1=2;
31	c2=2;
32	w=1;
33	alpha=0.5;
34	%posisi dan kecepatan awal
35	for i=1:m
36	for j=1:dim
37	x(i,j)=round(random('unif',0,1));
38	v(i,j)=random('unif',0,1);
39	end
40	end
41	pbest=zeros(size(v));
42	gbest=zeros(1,dim);
43	nilaimax=0;
44	t=0;
45	sukses=0;
46	while t<maxiter
47	t=t+1
48	for i=1:m
49	fx(i)=0;
50	cek=0;
51	z=0;
52	for j=1:const
53	fc(j)=0;
54	nilai=0;
55	for k=1:dim

No	*Source code*
56	nilai=nilai+a(1,k)*x(i,k);
57	fc(j)=fc(j)+c(j,k)*x(i,k);
58	end
59	if fc(j)>b(j)
60	cek=1;
61	end
62	end
63	if cek==1
64	fx(i)=0;
65	else
66	fx(i)=nilai;
67	end
68	end
69	%menentukan pbest dan gbest
70	[nilai,posisipbest]=max(fx);
71	pbest(posisipbest,:)=x(posisipbest,:);
72	if nilai>nilaimax
73	nilaimax=nilai
74	gbest(1,:)=x(posisipbest,:);
75	end
76	%update kecepatan dan posisi
77	for i=1:m
78	if i~=posisipbest
79	for j=1:dim
80	r1=random('unif',0,1);
81	r2=random('unif',0,1);
82	v(i,j)=w*v(i,j)+c1*r1*(pbest(i,j)-x(i,j))+c2*r2*(gbest(1,j)-x(i,j));
83	if (0<v(i,j)) & (v(i,j)<=alpha)
84	x(i,j)=x(i,j);
85	else if (alpha<v(i,j)) & (v(i,j)<=(0.5*(1+alpha)))
86	x(i,j)=pbest(i,j);
87	else if ((0.5*(1+alpha))<v(i,j)) & (v(i,j)<=1)

No	*Source code*
88	x(i,j)=gbest(1,j);
89	end
90	end
91	end
92	alpha=random('unif',0,1);
93	s(i,j)=1/(1+exp(-v(i,j)));
94	r=random('unif',0,1);
95	if r<=s(i,j)
96	x(i,j)=1;
97	else
98	x(i,j)=0;
99	end
100	end
101	end
102	end
103	pMBPSO(t)=nilai;
104	if nilai==8706.1
105	sukses=sukses+1;
106	end
107	end
108	%prosentase mendapatkan solusi eksak
109	pmsMBPSO=(sukses/maxiter)*100;

3. Program HBPSO

No	*Source code*
1	clc
2	clear global
3	%input knapsack
4	a=[600.1 310.5 1800 3850 18.6 198.7 882 4200 402.5 327];
5	c=[20 5 100 200 2 4 60 150 80 40
6	20 7 130 280 2 8 110 210 100 40
7	60 3 50 100 4 2 20 40 6 12

No	*Source code*
8	60 8 70 200 4 6 40 70 16 20
9	60 13 70 250 4 10 60 90 20 24
10	60 13 70 280 4 10 70 105 22 28
11	5 2 20 100 2 5 10 60 0 0
12	45 14 80 180 6 10 40 100 20 0
13	55 14 80 200 6 10 50 140 30 40
14	65 14 80 220 6 10 50 180 30 50];
15	b=[450
16	540
17	200
18	360
19	440
20	480
21	200
22	360
23	440
24	480];
25	%inisialisasi PSO
26	dim=length(a);
27	const=length(b);
28	m=30;
29	maxiter=100;
30	c1=2;
31	c2=2;
32	w=1;
33	alpha=0.333;
34	mut=0.05;
35	%posisi dan kecepatan awal
36	for i=1:m
37	for j=1:dim
38	x(i,j)=round(random('unif',0,1));
39	v(i,j)=random('unif',0,1);

No	*Source code*
40	end
41	end
42	pbest=zeros(size(v));
43	gbest=zeros(1,dim);
44	nilaimax=0;
45	t=0;
46	sukses=0;
47	while t<maxiter
48	t=t+1
49	for i=1:m
50	fx(i)=0;
51	cek=0;
52	for j=1:const
53	fc(j)=0;
54	nilai=0;
55	for k=1:dim
56	nilai=nilai+a(1,k)*x(i,k);
57	fc(j)=fc(j)+c(j,k)*x(i,k);
58	end
59	if fc(j)>b(j)
60	cek=1;
61	end
62	end
63	if cek==1
64	fx(i)=0;
65	else
66	fx(i)=nilai;
67	end
68	end
69	%menentukan pbest dan gbest
70	[nilai,posisipbest]=max(fx);
71	pbest(posisipbest,:)=x(posisipbest,:);

No	*Source code*
72	if nilai>nilaimax
73	nilaimax=nilai
74	gbest(1,:)=x(posisipbest,:);
75	end
76	%update kecepatan dan posisi
77	for i=1:m
78	if i~=posisipbest
79	for j=1:dim
80	r1=random('unif',0,1);
81	r2=random('unif',0,1);
82	v(i,j)=w*v(i,j)+c1*r1*(pbest(i,j)-x(i,j))+c2*r2*(gbest(1,j)-x(i,j));
83	s(i,j)=1/(1+exp(-v(i,j)));
84	r=random('unif',0,1);
85	if r<=s(i,j)
86	x(i,j)=1;
87	else
88	x(i,j)=0;
89	end
90	%crossover
91	if (0<=r) & (r<=alpha)
92	x(i,j)=x(i,j);
93	else if (alpha<r) & (r<=(2*alpha))
94	x(i,j)=pbest(i,j);
95	else if ((2*alpha)<r) & (r<=1)
96	x(i,j)=gbest(1,j);
97	end
98	end
99	end
100	%mutation
101	if r<=mut
102	if x(i,j)==0
103	x(i,j)=1;

No	*Source code*
104	else if x(i,j)==1
105	x(i,j)=0;
106	end
107	end
108	else
109	x(i,j)=x(i,j);
110	end
111	end
112	end
113	end
114	pHBPSO(t)=nilai;
115	if nilai==8706.1
116	sukses=sukses+1;
117	end
118	end
119	%prosentase mendapatkan solusi eksak
120	pmsHBPSO=(sukses/maxiter)*100;

LAMPIRAN 2
Data Uji

Data 1
Jumlah data : 10
Jumlah kendala : 10
Solusi eksak : 8706.1

Nilai Data ke-									
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
600.1	310.5	1800	3850	18.6	198.7	882	4200	402.5	327

	Kendala									
Data	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	20	20	60	60	60	60	5	45	55	65
2	5	7	3	8	13	13	2	14	14	14
3	100	130	50	70	70	70	20	80	80	80
4	200	280	100	200	250	280	100	180	200	220
5	2	2	4	4	4	4	2	6	6	6
6	4	8	2	6	10	10	5	10	10	10
7	60	110	20	40	60	70	10	40	50	50
8	150	210	40	70	90	105	60	100	140	180
9	80	100	6	16	20	22	0	20	30	30
10	40	40	12	20	24	28	0	0	40	50

Batas Kendala ke-									
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
450	540	200	360	440	480	200	360	440	480

Keterangan :

1. Nilai data ke- (warna hijau) adalah keuntungan tiap data yang akan dimaksimalkan.
2. Kendala (warna biru) adalah kendala pada masing-masing data.
3. Batas kendala ke- (warna biru) adalah batas maksimal tiap-tiap kendala, dengan kata lain solusi apabila dikalikan dengan kendala, tidak boleh melebihi batas kendala.

Data 2

Jumlah data : 20
Jumlah kendala : 10
Solusi eksak : 6120

Nilai Data ke-									
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
100	220	90	400	300	400	205	120	160	580
Nilai Data ke-									
11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
400	140	100	1300	650	320	480	80	60	2550

	Kendala									
Data	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	8	8	3	5	5	5	0	3	3	3
2	24	44	6	9	11	11	0	4	6	8
3	13	13	4	6	7	7	1	5	9	9
4	80	100	20	40	50	55	10	20	30	35
5	70	100	20	30	40	40	4	14	29	29
6	80	90	30	40	40	40	10	20	20	20
7	45	75	8	16	19	21	0	6	12	16
8	15	25	3	5	7	9	6	12	12	15
9	28	28	12	18	18	18	0	10	10	10
10	90	120	14	24	29	29	6	18	30	30
11	130	130	40	60	70	70	32	42	42	42
12	32	32	6	16	21	21	3	9	18	20
13	20	40	3	11	17	17	0	12	18	18
14	120	160	20	30	30	35	70	100	110	120
15	40	40	5	25	25	25	10	20	20	20
16	30	60	0	10	15	20	0	5	15	20
17	20	55	5	13	25	25	0	6	18	22
18	6	10	3	5	5	5	0	4	7	7
19	3	6	0	1	1	2	0	1	2	3
20	180	240	20	8	100	110	0	20	40	50

Batas Kendala ke-									
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
550	700	130	240	280	310	110	205	260	275

Data 3

Jumlah data : 28
Jumlah kendala : 10
Solusi eksak : 12400

Nilai Data ke-									
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
100	220	90	400	300	400	205	120	160	580
Nilai Data ke-									
11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
400	140	100	1300	650	320	480	80	60	2550
Nilai Data ke-									
21	**22**	**23**	**24**	**25**	**26**	**27**	**28**		
3100	1100	950	450	300	220	200	520		

	Kendala									
Data	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	8	8	3	5	5	5	0	3	3	3
2	24	44	6	9	11	11	0	4	6	8
3	13	13	4	6	7	7	1	5	9	9
4	80	100	20	40	50	55	10	20	30	35
5	70	100	20	30	40	40	4	14	29	29
6	80	90	30	40	40	40	10	20	20	20
7	45	75	8	16	19	21	0	6	12	16
8	15	25	3	5	7	9	6	12	12	15
9	28	28	12	18	18	18	0	10	10	10
10	90	120	14	24	29	29	6	18	30	30
11	130	130	40	60	70	70	32	42	42	42
12	32	32	6	16	21	21	3	9	18	20
13	20	40	3	11	17	17	0	12	18	18
14	120	160	20	30	30	35	70	100	110	120
15	40	40	5	25	25	25	10	20	20	20
16	30	60	0	10	15	20	0	5	15	20
17	20	55	5	13	25	25	0	6	18	22
18	6	10	3	5	5	5	0	4	7	7
19	3	6	0	1	1	2	0	1	2	3
20	180	240	20	80	100	110	0	20	40	50
21	220	290	30	60	70	70	30	50	60	60
22	50	80	40	50	55	55	10	30	50	55
23	30	90	10	20	20	20	0	5	25	25
24	50	70	0	30	50	50	10	20	25	30
25	12	27	5	10	15	20	10	20	25	25

	Kendala									
Data	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
26	5	17	0	5	15	15	5	10	15	15
27	8	8	0	3	6	6	0	10	10	10
28	18	28	10	20	20	20	10	20	28	28

Batas Kendala ke-									
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
930	1210	272	462	532	572	240	400	470	490

Data 4

Jumlah data : 39
Jumlah kendala : 5
Solusi eksak : 10618

Nilai Data ke-									
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
560	1125	300	620	2100	431	68	328	47	122
Nilai Data ke-									
11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
322	196	41	25	425	4260	416	115	82	22
Nilai Data ke-									
21	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
631	132	420	86	42	103	215	81	91	26
Nilai Data ke-									
31	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	
49	420	316	72	71	49	108	116	90	

	Kendala				
Data	**1**	**2**	**3**	**4**	**5**
1	40	16	38	8	38
2	91	92	39	71	52
3	10	41	32	30	30
4	30	16	71	60	42
5	160	150	80	200	170
6	20	23	26	18	9
7	3	4	5	6	7
8	12	18	40	30	20
9	3	6	8	4	0
10	18	0	12	8	3
11	9	12	30	31	21

	Kendala				
Data	**1**	**2**	**3**	**4**	**5**
12	25	8	15	6	4
13	1	2	0	3	1
14	1	1	1	0	2
15	10	0	23	18	14
16	280	200	100	60	310
17	10	20	0	21	8
18	8	6	20	4	4
19	1	2	3	0	6
20	1	1	0	2	1
21	49	70	40	35	18
22	8	9	6	15	15

	Kendala				
Data	1	2	3	4	5
23	21	22	8	31	38
24	6	4	0	2	10
25	1	1	6	2	4
26	5	5	4	7	8
27	10	10	22	8	6
28	8	6	4	2	0
29	2	4	6	8	0
30	1	0	1	0	3
31	0	4	5	2	0
32	10	12	14	8	10
33	42	8	8	6	6
34	6	4	2	7	1

	Kendala				
Data	1	2	3	4	5
35	4	3	8	1	3
36	8	0	0	0	0
37	0	10	20	0	3
38	10	0	0	20	5
39	1	6	0	8	4

Batas Kendala ke-				
1	2	3	4	5
600	500	500	500	600