UNPjoMath Vol.4 No 2 September 2021
ISSN:997 235516589 1-6

Algoritma Genetika pada Optimasi Persoalan *Knapsack* 0/1

Abdullah Husein[#1], Dewi Murni[*2], Meira Parma Dewi[*3]

[#]*Student of Mathematics Department, Universitas Negeri Padang, Indonesia*
**Lecturers of Mathematics Department, Universitas Negeri Padang, Indonesia*

[1]abdullah.husein9192@gmail.com
[2]dewimunp@gmail.com
[3] meira.daud@gmail.com

Abstract – The problem of 0/1 Knapsack is an issue in the selection of objects from the set of objects that each object have a decision "selected" or "not selected". The decision to choose a object is prioritized by the weight and profit of these objects, for example, to maximize profits or minimize costs. The main issue of this problem, it take to many processes and time to find the optimum solution. Therefore, we need a method and a program to find aproximate solutions to this problem so that decisions can be made quickly with fixed gain maximum profit. The purpose of this study is to obtain an efficient way to finding the optimum solution of this problem. Optimization method that used in this research is genetic algorithm, while the program is made in Python programming language. Based on this research, it is known that the genetic algorithm is able to obtain the optimum solution knapsack problem in a fairly short time.

Keywords – 0/1 knapsack problem, finding the optimal solution, genetic algorithm

Abstrak – Persoalan *Knapsack* 0/1 merupakan persoalan pemilihan objek dari suatu himpunan objek yang masing-masingnya memiliki keputusan “dipilih” atau “tidak dipilih”. Keputusan untuk memilih suatu objek disesuaikan dengan prioritas yang diinginkan berdasarkan nilai bobot dan profit objek tersebut, misalnya untuk memaksimalkan keuntungan atau meminimalkan biaya. Masalah utama dari persoalan ini ada pada banyaknya proses dan lama waktu yang dibutuhkan untuk mencari solusi optimumnya. Oleh karena itu, diperlukan suatu metode dan program untuk mencari solusi persoalan ini sehingga pengambilan keputusan bisa dilakukan dengan cepat tetapi tetap memperoleh keuntungan yang maksimum atau mendekati maksimum. Tujuan dari penelitian ini adalah untuk memperoleh suatu cara yang efisien untuk menyelesaikan solusi optimum pada persoalan *Knapsack* 0/1. Metode optimasi yang digunakan adalah algoritma genetika, sedangkan program dibuat dengan bahasa pemrograman Python. Berdasarkan hasil penelitian, diketahui bahwa algoritma genetika mampu memperoleh solusi optimum persoalan *Knapsack* dalam waktu yang cukup singkat.

Kata kunci – persoalan knapsack 0/1, pencarian solusi optimum, algoritma genetika

PENDAHULUAN

Program linier atau persoalan linier merupakan metode matematika dalam mengalokasikan sumber daya yang terbatas untuk mencapai suatu tujuan tertentu, seperti memaksimalkan keuntungan atau meminimalkan biaya. Penerapan persoalan ini banyak ditemukan dalam berbagai aspek, mulai dari industri, ekonomi, militer, dan lain-lain. Persoalan tersebut memiliki ciri khas dalam memodelkan permasalahan ke bentuk linier yaitu adanya fungsi tujuan dan fungsi kendala.

Salah satu bentuk persoalan linier yaitu persoalan pengambilan keputusan yang dikenal sebagai *linear decisions problem*. Sebagai contoh pada penyusunan peti-peti kemas ke dalam kapal kontainer, pada kontainer tersebut terdapat batasan ruang yang tersedia sehingga untuk memasukkan objek-objek yang memiliki ukuran yang bervariasi mesti dipilih (diputuskan “angkut” atau “tidak”) secara bijak agar tidak memboroskan ruang maupun biaya.

Penyeleksian objek-objek tersebut dapat dirunut berdasarkan prioritas yang diinginkan, bisa saja berdasarkan nilai harga (*profit*) ataupun bobotnya (*weight*). Persoalan pemilihan objek seperti ini dikenal sebagai Persoalan *Knapsack*.

Definisi persoalan *Knapsack* [1] yaitu; Diberikan suatu himpunan dari n jenis benda dan sebuah ransel (*knapsack*), dengan p_j sebagai profit benda, w_j sebagai bobot benda, dan W sebagai kapasitas dari ransel tersebut. Persoalannya, pilihlah subset benda-benda tersebut untuk memaksimalkan fungsi tujuan:

$$z = \sum_{j=1}^{n} p_j x_j \qquad (1)$$

dengan kendala (*constrains*):

$$\sum_{j-1}^{n} w_j x_j \leq W \qquad (2)$$

dimana pilihan tersebut (x_j) terletak antara 0 yang berarti "tidak diambil" dan 1 yang berarti "ambil". Berdasarkan variabel keputusannya yang terdiri dari 0 dan 1, persoalan *Knapsack* ini dikenal sebagai persoalan *Knapsack* 0/1.

Pendekatan secara langsung untuk mencari solusi optimasi dari persoalan *Knapsack* 0/1 dapat dilakukan dengan cara mengevaluasi fungsi tujuan dan fungsi kendala dari semua kemungkinan vektor solusi $X = \{x_1, x_2, x_3, \dots, x_n\}$ kemudian memilih vektor solusi yang memiliki fungsi tujuan tertinggi dan memenuhi fungsi kendalanya. Persoalan utama dari pedekatan ini adalah lamanya waktu yang dibutuhkan untuk mencari solusi, yaitu dibutuhkan pengevaluasian nilai vektor sebanyak 2^n [1]. Penggunaan metode pencarian solusi optimum secara eksak seperti teknik *brute force* memiliki kompleksitas waktu $O(n2^n)$, dimana waktu eksekusi meningkat secara eksponensial [2]. Persoalan seperti ini dalam istilah komputasi dikenal sebagai persoalan yang intractable, yaitu suatu persoalan yang secara eksak tidak dapat diselesaikan dalam batasan waktu polinomial [3]. Karena itu dibutuhkan suatu pendekatan lain untuk mencari keoptimalan solusi dengan waktu yang rasional.

Salah satu pendekatan yang bisa diterapkan adalah dengan menggunakan algoritma genetika. Algoritma genetika merupakan algoritma optimasi yang diadaptasi dari proses genetika dan proses seleksi alam dalam teori evolusi [4]. Algoritma ini termasuk ke dalam metode optimasi metaheuristik, yaitu suatu teknik untuk mencari penyelesaian aproksimasi suatu persoalan [5]. Algoritma ini memiliki suatu kelebihan, khususnya pada persoalan *Knapsack* dengan banyak ketersediaan objek yang besar dimana algoritma ini tidak bekerja dengan mengevaluasi semua kemungkinan vektor solusi (2^n), melainkan hanya beberapa vektor solusi yang menjadi tebakan awal dikali dengan banyaknya iterasi. Hal ini tentu saja dapat memangkas banyaknya proses yang dikerjakan dan waktu pemrosesan secara signifikan.

Penelitian ini bertujuan untuk menerapkan algoritma genetika pada bentuk khusus persoalan *Knapsack*, yaitu persoalan *Knapsack* 0/1. Selain itu pada penelitian ini juga akan dibuat program optimasi persoalan *Knapsac*k 0/1 menggunakan bahasa Python [6].

METODE

Penelitian ini merupakan penelitian dasar (teoritis). Metode yang digunakan adalah metode deskriptif dengan cara menganalisis teori-teori yang relevan dengan permasalahan yang dibahas dan berlandaskan pada studi kepustakaan. Adapun langkah-langkah yang dilakukan adalah sebagai berikut:

1. Menelaah dan memahami teori mengenai Persoalan *Knapsack* 0/1.
2. Merancang algoritma genetika untuk menyelesaikan Persoalan *Knapsack* 0/1.
3. Membuat program menggunakan bahasa Python berdasarkan prinsip algoritma genetika untuk menyelesaikan optimasi pada Persoalan *Knapsack* 0/1.
4. Mengujikan program yang telah dibuat terhadap penerapan contoh soal Persoalan *Knapsack* 0/1.
5. Menyimpulkan hasil penelitian.

HASIL DAN PEMBAHASAN

A. *Peninjauan Persoalan Knapsack 0/1*

Persoalan *Knapsack* 0/1 secara matematis dinyatakan sebagai berikut: Diberikan n unit objek dengan bobot masing-masing $w_1, w_2, w_3, \dots, w_n$ dan profit masing-masing objek $p_1, p_2, p_3, \dots, p_n$. Objek-objek tersebut akan dimasukkan ke dalam ransel (*knapsack*) yang memiliki kapasitas bobot sebesar c. Solusi persoalan tersebut dapat dinyatakan dalam vektor X:

$$X = \{x_1, x_2, x_3, \dots, x_n\}$$

dengan x_i bernilai 1 jika objek ke-i diambil dan bernilai 0 jika tidak diambil. Misalnya $X = \{1, 0, 0\}$ merupakan solusi dimana objek yang diambil adalah objek pertama, sedangkan objek kedua dan objek ketiga tidak diambil.

Solusi persoalan *Knapsack* 0/1 dipandang sebagai berikut:

Maksimalkan $z = p_1x_1 + p_2x_2 + p_3x_3 + \cdots + p_nx_n$

Dengan kendala:

$$\sum_{i-1}^{n} w_i x_i \leq c$$

$x_i = 0$ atau 1, $i = 1, 2, 3, \dots, n$

p_i, w_i, dan c merupakan konstanta

Dengan:

z = fungsi tujuan
p_i = profit objek ke-i
w_i = bobot objek ke-i
x_i = variabel keputusan objek ke-i
c = kapasitas maksimum knapsack

Setelah memformulasikan persoalan, penyelesaian yang dilakukan adalah memilih sejumlah objek dengan aturan total bobot objek-objek tersebut tidak melebihi kapasitas *knapsack* tapi memiliki keuntungan yang maksimum. Objek yang dipilih akan masuk ke dalam keputusan "ambil", sedangkan yang tidak dipilih masuk keputusan "tidak diambil".

Sebelum menyelesaikan persoalan *Knapsack* 0/1, bentuk persoalan haruslah *feasible* dalam artian memenuhi asumsi-asumsi berikut:

1. $\sum_{j=1}^{n} w_j > c$
2. $w_j \leq c$ untuk setiap $j \in N$

Berdasarkan asumsi di atas dapat diinterpretasikan, pada asumsi pertama jika semua objek terpilih maka bobotnya akan melebihi kapasitas *knapsack* sehingga objek yang dipilih haruslah himpunan bagian dari keseluruhan ketersediaan objek. Jika asumsi ini tidak terpenuhi maka persoalan akan menghasilkan solusi yang trivial, $x_j = 1$ untuk setiap $j \in N$. Asumsi kedua, setiap objek harus dapat dimasukkan ke dalam *knapsack*.

B. *Pencarian Solusi Optimum Persoalan Knapsack 0/1 Menggunakan Algoritma Genetika*

Proses pada algoritma genetika dipandang sebagai proses perbaikan yang berkelanjutan yang dikenal dengan istilah evolusi. Solusi yang dihasilkan oleh algoritma genetika pada persoalan *knapsack* 0/1 berasal dari keputusan optimum dari sejumlah tebakan awal yang terus disempurnakan melalui operator genetika (seleksi, penyilangan, dan mutasi) pada setiap tahapan iterasi yang dikenal dengan istilah generasi.

Berikut langkah-langkah untuk mencari solusi persoalan *Knapsack* 0/1 dengan algoritma genetika:

1) Uji kefeasibelan: Lakukan pengujian kefeasibelan data berdasarkan kedua asumsi sebelumnya. Jika asumsi terpenuhi, lanjut ke langkah berikutnya.

2) Pengkodean kromosom: Pada tahap ini, solusi dikodekan dalam bentuk kromosom. Pada Persoalan *Knapsack* 0/1, kromosom dapat direpresentasikan dalam *array* yang memuat elemen biner sesuai banyak objek yang tersedia. Setiap elemen *array* menyatakan status objek, diangkut (‘1’) atau tidak(‘0’).

3) *Pembangkitan populasi awal:* Pembangkitan populasi awal bisa diartikan proses memberikan sejumlah tebakan awal sebagai solusi persoalan. Proses ini dilakukan dengan membangkitkan kromosom secara acak sebanyak ukuran populasi.

4) *Perbaikan Kromosom:* Perbaikan diterapkan pada kromosom dalam populasi yang tidak memenuhi fungsi kendala. Fungsi kendala pada Persoalan *Knapsack* 0/1 dinyatakan sebagai:

$$\sum_{i=1}^{n} w_i x_i \leq c \qquad (3)$$

Dengan,
$x_i = 0$ atau 1, $i = 1, 2, 3, \dots, n$
w_i dan c merupakan konstanta
n = banyaknya objek
w_i = bobot objek ke-i
x_i = variabel keputusan objek ke-i
c = kapasitas maksimum *knapsack*

atau dengan kata lain bobot total objek yang direpresentasikan oleh kromosom tidak boleh melebihi kapasitas *knapsack*.

Karena kromosom dibangkitkan secara acak, solusi yang direpresentasikan oleh kromosom tersebut tidak selalu berupa solusi yang memenuhi kendala. Kromosom yang tidak memenuhi fungsi kendala disebut sebagai kromosom tidak layak (*infeasible*). Agar diperoleh solusi sesuai yang diharapkan, diperlukan suatu strategi untuk memperbaiki kromosom *infeasible* ini menjadi kromosom yang memuhi kendala (*feasible*).

Strategi perbaikan yang diterapkan yaitu dengan cara mengubah secara acak elemen ‘1’ pada kromosom menjadi elemen ‘0’ sampai diperoleh kromosom yang *feasible*. Cara ini dapat diartikan dengan mengganti status objek dari ‘dipilih’ menjadi ‘tidak dipilih’.

5) *Evaluasi Nilai Fitness:* Pada tahap ini setiap kromosom dalam populasi akan dihitung nilai fitnessnya. Nilai fitness inilah yang menyatakan kualitas kromosom, semakin tinggi nilai fitness semakin baik solusi yang direpresentasikan. Pada Persoalan *Knapsack* 0/1 fungsi fitness dapat dinyatakan langsung dari fungsi tujuannya, yaitu:

$$f = \sum_{j=1}^{n} p_j x_j \qquad (4)$$

Dengan,
$x_j = 0$ atau 1, $j = 1, 2, 3, \dots, n$
n = banyak objek yang tersedia
p_j = profit objek ke-j
x_j = variabel keputusan objek ke-j

Dengan pemilihan fungsi fitness seperti ini, nilai fitness sekaligus menyatakan total profit dari solusi yang direpresentasikan oleh kromosom.

6) *Pembentukan Populasi Baru:* Populasi baru dibentuk melalui serangkaian proses evolusi, yaitu elitisme, seleksi, penyilangan, dan mutasi.

7) *Ulangi dari langkah 4 hingga kondisi berhenti terpenuhi:* Pada populasi baru yang telah dibentuk akan diterapkan strategi perbaikan untuk memastikan semua kromosomnya *feasible* (langkah 4), diperiksa nilai fitnessnya untuk memperoleh kromosom terbaik (langkah 5), kemudian dari populasi ini akan dibentuk populasi baru melalui proses elitisme, seleksi, penyilangan, dan mutasi (langkah 6). Proses ini terus diulang hingga kriteria berhenti terpenuhi, biasanya setelah sejumlah generasi tertentu tercapai atau telah terjadi konvergensi sehingga diharapkan pada generasi terakhir diperoleh solusi terbaik.

C. *Implementasi Algoritma Genetika pada Program Optimasi Persoalan Knapsack 0/1 Menggunakan Bahasa Python*

Dalam menerapkan algoritma genetika ke dalam pemrograman bahasa Python, dilakukan tahapan-tahapan sebagai berikut:

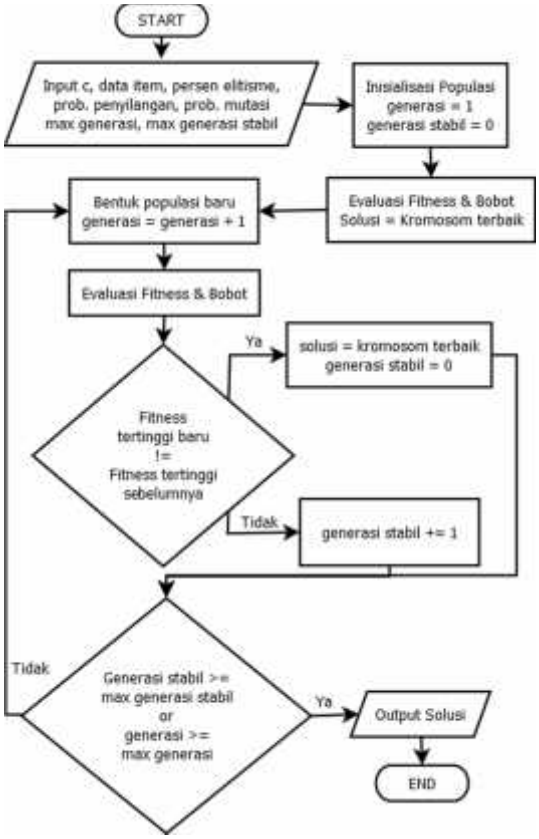
1) *Representasi Item:* Untuk merepresentasikan item yang akan dimuat, digunakan struktur data berupa *list* yang menampung sekumpulan *list* lainnya yang berisi informasi pada setiap item; yaitu nama item, bobot, dan profit seperti ditunjukkan berikut ini:

```
[[ ‘item 0’ ,  Bobot item 0 ,  Profit item 0 ],
 [ ‘item 1’ ,  Bobot item 1 ,  Profit item 1 ],
 [ ‘item 2’ ,  Bobot item 2 ,  Profit item 2 ],
 [‘item n-1’, Bobot item n-1, Profit item n-1]]
```

2) *Pengkodean kromosom:* Kromosom dinyatakan dengan sebuah *list* yang ukurannya sesuai dengan banyak item yang tersedia. Setiap elemen dalam *list* menyatakan apakah item tertentu dimuat (‘1’) atau tidak (‘0’). Jika terdapat n buah item, representasi kromosomnya dapat dinyatakan sebagai berikut:

$[x_0, x_1, x_2, \dots, x_{n-1}]$
$x_i = 0$ atau 1, untuk $i = 0, 1, 2, \dots, n-1$

3) *Flowchart Program Utama*: Berikut ini flowchart dari program utama.



Gambar 1. Flowchart Program

Program menggunakan beberapa fungsi, namun proses mengikuti langkah pada *flowchart*. Fungsi yang digunakan diantaranya:

a. Fungsi untuk mengevaluasi fitness dan bobot suatu kromosom, jika total bobot bobot kromosom melebihi kapasitas *knapsack*, maka nilai fitnessnya diset menjadi 0. Kode dalam bahasa Python:

```
def hitungFitness(krom):
    fitness = .0
    total_bobot = .0
    for i, (item, bobot, profit) in
        enumerate(ITEM_KNAPSACK):
       if krom[i] == 1:
           fitness += float(profit)
           total_bobot += float(bobot)
       if total_bobot > KAP_KNAPSACK:
           return 0
        return fitness
```

b. Fungsi untuk memperbaiki populasi. Fungsi ini akan melakukan pemeriksaan kelayakan setiap kromosom dalam populasi. Jika ditemukan kromosom yang fitnessnya bernilai 0 (total bobotnya melebihi kapasitas *knapsack*), fungsi ini akan mengubah secara acak bit '1' pada kromosom menjadi bit '0' hingga diperoleh kromosom yang *feasible*. Berikut kode dalam bahasa Python:

```
def perbaikiKromosom(pop):
    for krom in pop:
        while hitungFitness(krom) == 0:
            # ubah bit '1' menjadi '0'
            bit_1 = [i for i, bit in
                enumerate(krom) if bit == 1]
            krom[random.choice(bit_1)] = 0
```

c. Fungsi untuk mendapatkan solusi terbaik. Fungsi ini digunakan untuk mendapatkan kromosom terbaik dan nilai fitnessnya berikut representasi solusinya. Kode bahasa Python:

```
def solusiTerbaik(pop):
    fitness_tertinggi, krom_terbaik =
        max(((hitungFitness(krom), krom)
        for krom in pop))
    item_terpilih, bobot_item, profit_item=
        [], [], []
    for i, (item, bobot,profit) in
        enumerate(ITEM_KNAPSACK):
        if krom_terbaik[i] == 1:
            item_terpilih.append(item)
            bobot_item.append(bobot)
            profit_item.append(profit)
    return krom_terbaik, fitness_tertinggi,
        item_terpilih, bobot_item,
        profit_item
```

d. Fungsi seleksi, digunakan untuk menyeleksi kromosom yang akan dipertahankan dalam proses selanjutnya. Penyeleksian dilakukan dengan cara membagi kromosom ke dalam beberapa grup kemudian memilih kromosom terbaik pada setiap grup. Kode Python:

```
def seleksi(pop):
    # Grup dengan n kromosom
    grup_krom = random.sample(pop, UK_GRUP)
    fitness_krom_terpilih, krom_terpilih =
        max(((hitungFitness(krom), krom)
        for krom in grup_krom))
    return krom_terpilih
```

e. Fungsi untuk melakukan penyilangan kromosom. Penyilangan dilakukan dengan mempertukarkan bit kedua kromosom. Kode program:

```
def crossover(krom1, krom2):
    # Pilih titik potong penyilangan acak
    tipot = random.randint(1, BANYAK_ITEM-1)
    # Bentuk kromosom baru
    krom_anak1= krom1[:tipot] + krom2[tipot:]
    krom_anak2= krom2[:tipot] + krom1[tipot:]
    return krom_anak1, krom_anak2
```

f. Fungsi untuk melakukan mutasi. Fungsi ini digunakan untuk melakukan mutasi kromosom sekaligus memperbaiki kromosom hasil mutasi. Mutasi dilakukan dengan cara menukar nilai bit dengan inversnya (0 menjadi 1 dan sebaliknya). Kode bahasa Python:

```
def mutasi(krom):
    # inversi bit terpilih
    bit_mut= random.randint(0, BANYAK_ITEM-1)
    krom[bit_mut] = int(not krom[bit_mut])
    # Perbaiki kromosom hasil mutasi
    while hitungFitness(krom) == 0:
        # ubah bit '1' menjadi '0'
        bit_1 = [i for i, bit in
            enumerate(krom) if bit == 1]
        krom[random.choice(bit_1)] = 0
```

g. Fungsi untuk membentuk populasi baru. Populasi baru dibentuk dengan cara mengambil kromosom elite dari populasi sekarang sebagai anggota populasi, kemudian anggota lainnya ditambahkan melalui proses seleksi, penyilangan, dan mutasi. Kode program dalam bahasa Python:

```
def bentukPopulasiBaru(pop):
    # Banyaknya kromosom elit
    krom_elit = int(UK_POP * PERS_ELIT/100)
    # Urutkan populasi berdasarkan fitness
```

```
    pop_terurut = [krom for fitness, krom in
        sorted(((hitungFitness(krom), krom)
        for krom in pop), reverse=True)]
    # Menambahkan kromosom elit ke dalam
      populasi baru
    pop_baru = pop_terurut[:krom_elit]
    # Sisa populasi dibentuk dari hasil
      penyilangan dan mutasi
    While len(pop_baru) < UK_POPULASI:
        # Seleksi induk kromosom yg akan
          disilangkan
        induk1 = seleksi(pop)
        induk2 = seleksi(pop)
        # Silangkan sesuai prob. Crossover
        if random.random() < PROB_CROSSOVER:
            anak_krom1, anak_krom2 =
                crossover(induk1, induk2)
        else:
            anak_krom1, anak_krom2 =
                induk1, induk2
        # Mutasi anak kromosom berdasarkan
          prob. Mutasi
        if random.random() < PROB_MUTASI:
            mutasi(anak_krom1)
        if random.random() < PROB_MUTASI:
            mutasi(anak_krom2)
        # Tambahkan kromosom ke dalam
          populasi baru
        pop_baru.append(anak_krom1)
        pop_baru.append(anak_krom2)
    return pop_baru[:UK_POP]
```

Sesuai *flowchart*, proses yang dijalankan pada program utama yaitu:

a. Input kapasitas knapsack (`KAP_KNAPSACK`), ukuran populasi (`UK_POP`), ukuran grup (`UK_GRUP`), list data item (`ITEM_KNAPSACK`), banyak item (`BANYAK_ITEM`), persentase kromosom elit (`PERS_ELIT`), probabilitas penyilangan (`PROB_CROSSOVER`), probabilitas mutasi (`PROB_MUTASI`), maksimal banyaknya generasi (`MAX_BANYAK_GEN`), dan maksimal kekonvergenan generasi (`MAX_GEN_STABIL`).

b. Membangkitkan populasi awal secara acak sesuai banyaknya kromosom dalam populasi, kemudian memulai *counter* generasi.

```
populasi = [[random.choice((0, 1)) for i in
    range(BANYAK_ITEM)] for j in
    range(UK_POPULASI)]
generasi = 1
generasi_stabil = 0
```

c. Memperbaiki kromosom dalam populasi untuk memastikan kromosom selalu *feasible*. Proses dilakukan dengan memanggil fungsi perbaikan yang telah didefinisikan sebelumnya.

```
perbaikiPopulasi(populasi)
```

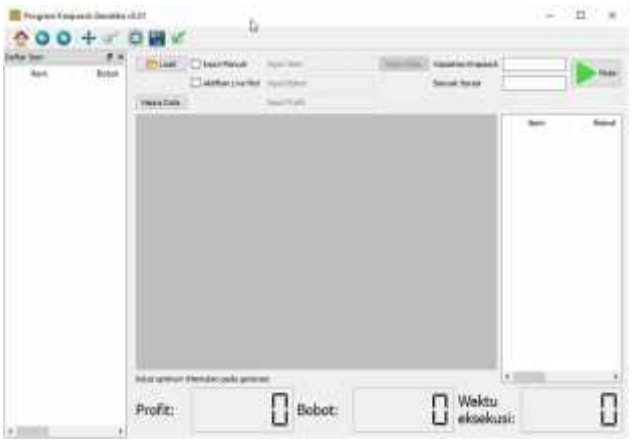
d. Memperoleh kromosom terbaik dan representasinya.

```
krom_terbaik, fitness_tertinggi,
    item_terpilih, bobot_item, profit_item =
    solusiTerbaik(populasi)
```

e. Membentuk populasi baru, memperbaiki kromosom, memperoleh kromosom terbaik dan representasinya, dan mengulangi proses sampai kondisi berhenti dipenuhi. Kondisi berhenti yang diterapkan yaitu proses akan dihentikan jika nilai fitness kromosom terbaik tidak mengalami peningkatan setelah beberapa generasi tercapai (telah tercapai konvergensi) atau generasi telah mencapai batas banyak generasi yang ditetapkan. Kode dalam bahasa Python:

```
while True:
    populasi = bentukPopulasiBaru(populasi)
    generasi += 1
    krom_terbaik_baru,fitness_tertinggi_baru,
        item_terpilih_baru, bobot_item_baru,
        profit_item_baru =
        solusiTerbaik(populasi)
    if fitness_tertinggi_baru !=
        fitness_tertinggi:
        # Solusi yang lain ditemukan
        krom_terbaik, fitness_tertinggi,
            item_terpilih, bobot_item,
            profit_item = krom_terbaik_baru,
                fitness_tertinggi_baru,
                item_terpilih_baru,
                bobot_item_baru,
                profit_item_baru
        generasi_stabil = 0
    else:
        generasi_stabil += 1
    # Cek kondisi berhenti
    if generasi_stabil >= MAX_GEN_STABIL:
        # fitness tertinggi tidak berubah
          setelah MAX_GEN_STABIL generasi
          tercapai => hentikan proses
        break
    if generasi >= MAX_BANYAK_GEN:
        # maksimum generasi tercapai =>
          hentikan proses
        break
```

Tampilan program (*Graphical User Interface*) dibuat dengan *module* PyQt versi 4, sementara untuk menampilkan plot grafik nilai fitness pada setiap generasi digunakan module *matplotlib*. Berikut *screenshoot* tampilan program:



Gambar2. Screenshoot Tampilan Program

D. *Pengujian dan Penerapan Program*

Program diujikan dengan parameter genetika yang digunakan yaitu ukuran populasi sebesar $5n$ (n merupakan banyak item yang tersedia), probabilitas penyilangan 0,7, dan probabilitas mutasi sebesar 0,4.

Dari hasil pengujian program pada contoh penerapan persoalan *Knapsack* 0/1 dengan objek yang tersedia sebanyak 81 item, program dapat menyelesaikan persoalan ini dengan waktu rata-rata 13,58 detik. Meskipun demikian, solusi yang dihasilkan program tidak dijamin berupa solusi eksak, melainkan suatu aproksimasi

dari solusi optimum. Selain itu, solusi yang diperoleh pada setiap pengujian bisa bervariasi meskipun perbedaannya tidak signifikan. Pada kasus uji yang digunakan dalam penelitian ini diperoleh variasi output solusi dengan profit sebesar 136,8, 136,9, 137,1, dan 137,2.

SIMPULAN

Berdasarkan hasil pembahasan dari penelitian ini, dapat disimpulkan hal-hal sebagai berikut:

1. Untuk mencari solusi aproksimasi pada optimasi Persoalan *Knapsack* 0/1 dapat diselesaikan dengan menggunakan algoritma genetika. Langkah-langkah untuk menyelesaikan persoalan tersebut dengan algoritma genetika adalah:
   a. Mengidentifikasi permasalahan dan melakukan pengujian ke-*feasible*-an permasalahan.
   b. Mengkodekan kandidat solusi ke dalam bentuk kromosom biner, kemudian membangkitkan kromosom secara acak sebanyak ukuran populasi.
   c. Menerapkan strategi perbaikan pada kromosom yang tidak memenuhi fungsi kendala dengan cara mengubah secara acak elemen ‘1’ pada kromosom menjadi elemen ‘0’ sampai diperoleh kromosom yang memenuhi fungsi kendala.
   d. Menetapkan fungsi tujuan sebagai fungsi fitness, kemudian menghitung semua nilai fitness kromosom dan menetapkan kromosom dengan nilai fitness tertinggi sebagai representasi solusi.
   e. Memilih beberapa kromosom dengan nilai fitness teratas untuk ditambahkan ke dalam populasi baru (*elitisme*), kemudian melakukan proses seleksi, *crossover*, dan mutasi untuk memenuhi kuota populasi baru.
   f. Mengulangi proses dari poin c sampai kriteria berhenti dipenuhi.
2. Penerapan algoritma genetika untuk mengoptimasi Persoalan *Knapsack* 0/1 ke dalam program bahasa Python dilakukan dengan cara membagi langkah-langkah algoritma ke dalam beberapa fungsi, kemudian memanggil fungsi tersebut melalui program utama sesuai urutan algoritma.
3. Algoritma genetika dapat menyelesaikan persoalan *Knapsack* 0/1 dengan ketersediaan objek yang cukup besar dalam hitungan detik.

REFERENSI

[1] Martello, Silvano & Paolo Toth. 1990. *Knapsack Problems Algorithms and Computer Implementations.* England: John Willey & Sons Ltd.

[2] Hristakeva, Maya & Dipti Shrestha. 2004. “Different Approaches to Solve the 0/1 Knapsack Problem”. Paper. Computer Science Department, Simpson College.

[3] Hristakeva, Maya & Dipti Shrestha. 2004. “Solving the 0/1 Knapsack Problem with Genetic Algorithm”. Paper. Computer Science Department, Simpson College.

[4] Zukhri, Zainudin. 2014. *Algoritma Genetika: Metode Komputasi Evolusioner untuk Menyelesaikan Masalah Optimasi*. Yogyakarta: Penerbit Andi.

[5] Talbi, El-Ghazali. 2009. *Metaheuristic From Design to Implementation*. New Jersey: John Willey & Sons, Inc.

[6] Husein, Abdullah. 2016. “Algoritma Genetika pada Optimasi Persoalan Knapsack 0/1”. Skripsi. Padang: UNP.