

Algoritma Local Search pada Bin Packing

Hubert, Dennis
Sekolah Tinggi Teknik Elektro
Institut Teknologi Bandung
Bandung, Indonesia
13222018@std.stei.itb.ac.id

Abstract— Penelitian ini membandingkan kinerja berbagai algoritma optimasi dalam mencapai *global optimum*. Urutan terbaik hingga terburuk untuk *hill climbing* adalah $RRHC > SAHC = SMHC > SHC$, di mana $RRHC$ unggul karena menghasilkan *initial state* baru setiap iterasi, sedangkan SHC paling lambat konvergen. Pada *Simulated Annealing*, semakin tinggi temperatur, semakin mudah *worst state* terpilih sehingga peluang *global optimum* menurun. Dalam *Genetic Algorithm*, peningkatan populasi, generasi, mutasi, dan *threshold* memperbesar peluang mencapai *global optimum* dengan memperluas ruang solusi dan variasi. Berdasarkan waktu eksekusi: $SAHC$ tercepat, diikuti $SMHC$, SHC , *Simulated Annealing*, $RRHC$, dan *Genetic Algorithm*. Sementara itu, *First Fit* lebih buruk dari *Worst Fit* karena ruang pencariannya lebih sempit meski *initial state*-nya lebih baik.

Keywords—local search, first fit, bin packing, Hill climbing

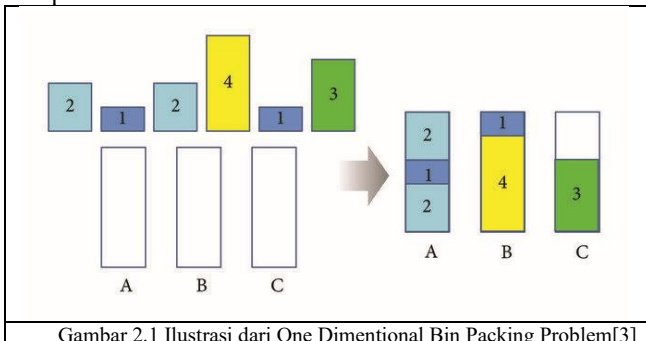
I. INTRODUCTION

Optimasi adalah proses mencari solusi terbaik dari sekumpulan kemungkinan solusi dalam suatu masalah. Dalam praktiknya, algoritma optimasi lokal dan evolusioner sering digunakan untuk menyelesaikan masalah yang kompleks seperti *8-queens*, penjadwalan, atau bin packing. Algoritma seperti *Hill Climbing*, *Simulated Annealing*, dan *Genetic Algorithm* memiliki mekanisme berbeda dalam mengeksplorasi ruang solusi dan mencapai *global optimum*.

II. STUDI LITERATUR

A. Bin Packing Problem

[1] Bin Packing Problem adalah masalah optimisasi di mana sejumlah item dengan ukuran tertentu harus dimasukkan ke dalam sejumlah wadah dengan kapasitas terbatas, dengan tujuan meminimalkan jumlah wadah yang digunakan. Masalah ini bersifat NP-hard dan banyak diterapkan dalam bidang logistik, manufaktur, serta ilmu komputer.



Gambar 2.1 Ilustrasi dari One Dimensional Bin Packing Problem[3]

Dalam masalah *one dimensional bin packing* klasik, diberikan sebuah urutan item $L = (a_1, a_2, \dots, a_n)$, di mana setiap item memiliki ukuran $s(a_i) \in (0, c]$. Tujuannya adalah untuk mengemas item-item tersebut ke dalam jumlah wadah (*bin*) berkapasitas satu seminimal mungkin,

yaitu membagi himpunan item tersebut ke dalam sejumlah minimum m_{subset} B_1, B_2, \dots, B_m sedemikian sehingga:

$$\sum_{a_i \in B_j} s(a_i) \leq c, 1 \leq j \leq m$$

Algoritma Worst case didapatkan pada saat:

$$RA = \inf\{r \geq 1 \mid RA(L) \leq r \text{ untuk semua } L\}$$

Dengan RA :

$$RA(L) = A(L)/OPT(L)$$

Keterangan :

RA : Rasio jumlah bin algoritma ke hasil optimal;

r : Batas atas algoritma

L : daftar weight

[4] Prosedur peningkatan solusi dalam *bin packing problem* umumnya terdiri dari tiga tahap utama:

1. Konstruksi solusi awal (Initial solution construction)

Algoritma Best Fit Decreasing (BFD) digunakan untuk membangun solusi awal yang layak — yaitu pembagian item ke dalam bin sedemikian rupa sehingga kapasitas tiap bin tidak terlampaui. Salah satu Konstruksi lainnya adalah Worst-Fit, yaitu membangun solusi dengan memasukkan masing-masing weight satu weight ke dalam masing-masing bin sehingga weight tersebar merata dan jumlah ruang kosong paling tinggi dan membuat worst case scenario.

2. Perbaikan solusi (Solution improvement)

Setiap bin dalam solusi saat ini dihapus satu per satu, dan item-item di dalamnya dicoba untuk didistribusikan kembali ke bin lain sesuai dengan aturan tertentu (misalnya *best-fit*, *first-fit*, atau *worst-fit*).

- o Jika hasil redistribusi menghasilkan solusi baru S' yang feasible, maka jumlah bin berkurang satu, dan proses diulang.
- o Jika S' tidak feasible (ada bin yang kelebihan kapasitas), maka algoritma local search diterapkan untuk mencoba mengurangi tingkat ketidaklayakan tersebut (*reduce infeasibility*).

3. Local search

search

Pada tahap ini, pasangan bin dalam solusi sementara yang tidak layak S' diperiksa secara berurutan. Item-item di antara kedua bin tersebut ditukar (redistributed) dengan harapan bahwa salah satu

kombinasi pertukaran menghasilkan solusi yang layak kembali. Bila solusi layak ditemukan, maka proses perbaikan solusi diulang dari tahap (b).

B. Local Search

[2]Local search adalah metode optimisasi heuristik yang bekerja dengan memodifikasi solusi saat ini secara bertahap (melalui tetangga/neighbor) untuk menemukan solusi yang lebih baik, sampai tidak ada perbaikan lebih lanjut yang dapat dilakukan dalam lingkungan lokalnya. Pada paper ini dilakukan penelitian dengan 3 metode Local search yaitu Hill Climbing, Simulated Annealing, dan Genetic Algorithm.

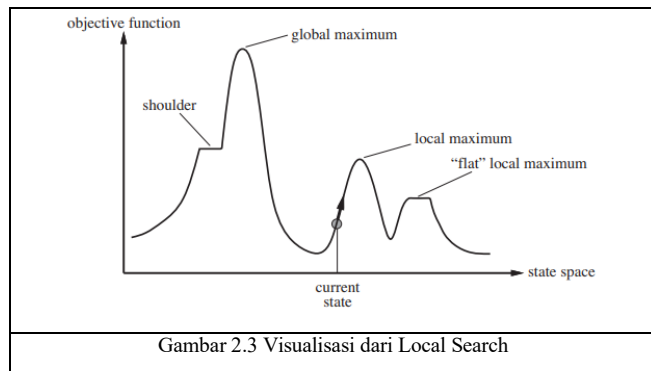
Pseudocode local search untuk Binary Packing :

```

procedure Local_Search( $S'$ )
1   $local\_improvement \leftarrow .TRUE.;$ 
2  while  $S'$  is infeasible and  $local\_improvement$  do
3     $local\_improvement \leftarrow .FALSE.;$ 
4    for  $j = z(S'), \dots, 2$  while  $.NOT.local\_improvement$ 
      and  $w_{S'}(id(j)) > b$  do
5      for  $i = 1, \dots, j - 1$  while  $.NOT.local\_improvement$  do
6        if Valid_Pair( $S', i, j$ )
7          then do
8             $dm(id(i), id(j)), dm(id(j), id(i)) \leftarrow iter;$ 
9            Obtain neighbor  $S'_{ij}$  by applying the differencing
            method to the  $i$ -th and  $j$ -th bins of solution  $S'$ ;
10           if  $|w_{S'_{ij}}(id(i)) - w_{S'_{ij}}(id(j))| < |w_{S'}(id(i)) - w_{S'}(id(j))|$ 
11             then do
12                $S' \leftarrow S'_{ij};$ 
13                $local\_improvement \leftarrow .TRUE.;$ 
14                $chd(id(i)), chd(id(j)) \leftarrow iter;$ 
15             end_then;
16           end_for;
17         end_for;
18       end_for;
19     end_while;
20   end_while;
21   return  $S' = \{B_1, \dots, B_{z(S')}\};$ 
end Local_Search;

```

Gambar 2.2 Local Search dari Binary Packing



Gambar 2.3 Visualisasi dari Local Search

III. METODOLOGI

A. Algoritma Inisiasi dan first fit

A.1 First Fit

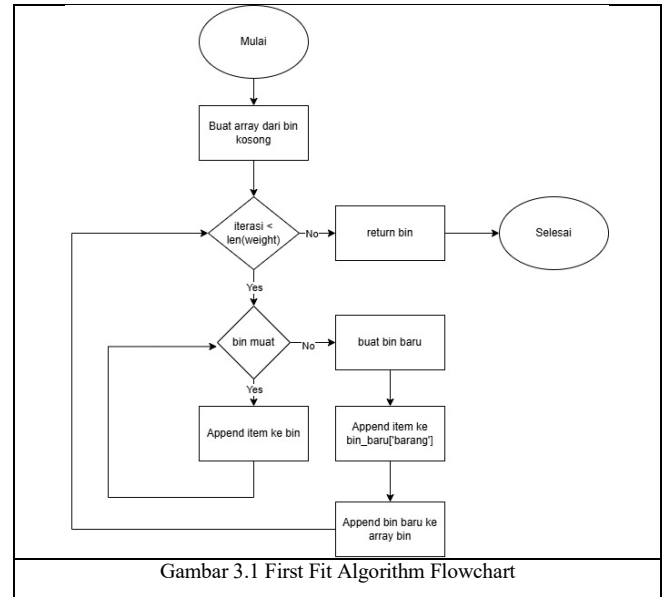
[5] Misalkan bin diberi indeks sebagai B_1, B_2, \dots , dengan masing-masing awalnya terisi pada level nol. Angka-angka a_1, a_2, \dots, a_n akan ditempatkan secara berurutan. Untuk menempatkan a_i , carilah indeks terkecil j sedemikian sehingga bin B_j terisi hingga level $f_j \leq c - a_i$, lalu tempatkan a_i ke dalam B_j . Setelah itu, bin B_j kini terisi hingga level $f_j + a_i$.

Keuntungan dari digunakannya Algoritma first fit adalah saat inisiasi terkadang dapat mendapatkan global optimal

sehingga mempercepat proses local search. [7] Selain itu Worst case dari first fit algorithm adalah :

$$FF(L) \leq c \times OPT(L)$$

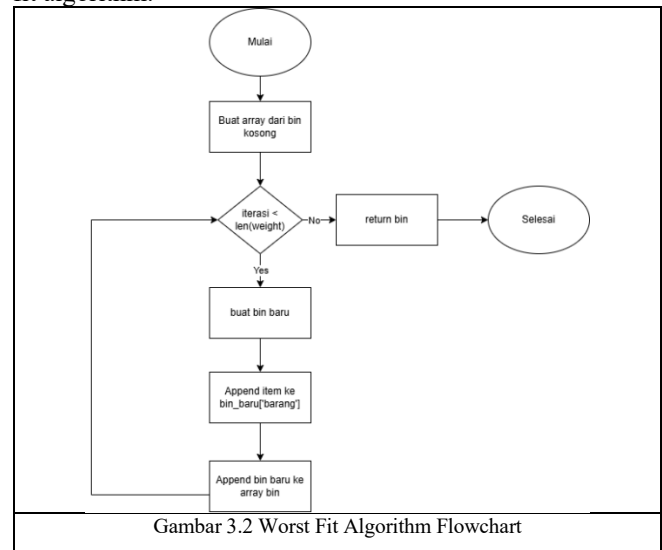
Berikut adalah flowchart dari first-fit algorithm :



Gambar 3.1 First Fit Algorithm Flowchart

A.2 Worst Fit

Sementara algoritma worst-fit membangun solusi dengan memasukkan masing masing weight satu weight ke dalam masing-masing bin sehingga weight tersebar merata dan jumlah ruang kosong paling tinggi dan selalu membuat worst case scenario. Gambar dibawah adalah Flowchart dari worst fit algorithm.



Gambar 3.2 Worst Fit Algorithm Flowchart

Pada Analisis akan digunakan Worst Fit agar visualisasi proses dari local search lebih mudah untuk diamati. Selain itu Perbandingan Performa akhir akan dibandingkan di bagian analisis.

Pada pengolahan digunakan tipe data pada bin sebagai list of dictionary. Masing-masing dictionary memiliki keys dan Value sebagai berikut:

['container':(string)],

['remaining':(int)]

['total':(int)]

['barang':[['id':(string)],['ukuran':(int)]]]

B. Perhitungan Heuristic

[6] Heuristik adalah aturan praktis atau educated guess yang mengurangi usaha pencarian dengan mengarahkan proses pencarian ke solusi yang lebih menjanjikan. Pada program karena objective dari bin packing adalah menggunakan bin sesedikit mungkin, heuristic merupakan fungsi minimalisasi. Heuristic dihitung dengan menjumlahkan space kosong dari masing-masing bin, jika ada bin yang space berlebih, diberi penalty sejumlah a. Pada percobaan yang dilakukan a ditune sebesar 10. Angka diambil tidak terlalu besar agar memungkinkan overflow dipilih dalam mencari hasil optimum.

Pada awalnya, perhitungan local search diberi constrain agar selalu mematuhi constraint besar container, namun seiring berjalannya waktu hasil banyak yang terjebak di local optimum. Sehingga diberikan penalti untuk overfitting pada Heuristic. Sehingga Heuristic didefinisikan sebagai Persamaan dibawah dengan c adalah space container :

$$difference = c - space \text{ terisi}$$

$$H = \sum_{i=0}^n difference > 0 ? difference : a * |difference|$$

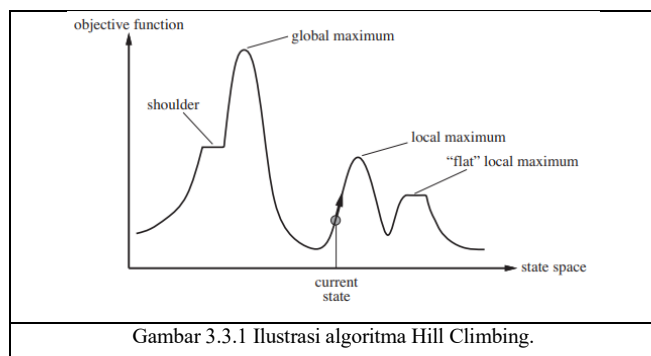
C. Hill climbing algorithms

[6] Hill climbing algorithm pada dasarnya adalah sebuah loop yang terus bergerak ke arah peningkatan atau penurunan nilai (sesuai fungsi objektif)—dengan kata lain, *hill climbing*. Algoritma ini berhenti ketika mencapai sebuah “puncak” atau local optimum dimana tidak ada *neighbor* yang memiliki nilai lebih tinggi. Algoritma ini tidak memelihara *search tree*, sehingga struktur data untuk *node* saat ini hanya perlu menyimpan *state* dan nilai dari fungsi objektif. Untuk mempermudah penggunaan istilah, karena pada bin packing heuristic didefinisikan sebagai minimum maka kata ascent di bawah dapat berarti juga optimalisasi dan Climbing = Descending, Naik = Turun (tergantung konteks).

Pada percobaan di paper ini dilakukan 4 jenis hill climbing, yaitu steepest ascent hill climbing (SAHC), sideway move hill climbing (SMHC), random restart hill climbing (RRHC), dan stochastic hill climbing (SHC).

Tipe data input : [['kontainer':(string)], ['remaining':(int)], ['total':(int)], ['barang':[['id':(string)], ['ukuran':(int)]]]

Tipe data output : [['kontainer':(string)], ['remaining':(int)], ['total':(int)], ['barang':[['id':(string)], ['ukuran':(int)]]]



C1. Steepest ascent hill climbing (SAHC)

[6] Pada Steepest Ascent hill climbing, pergeseran current dilakukan saat didapatkan hasil yang lebih baik. Hill climbing kadang disebut *greedy local search* karena algoritma ini langsung memilih *neighbor state* yang lebih baik tanpa memikirkan langkah berikutnya. Steepest Ascent Hill climbing biasanya membuat optimum solution yang cepat menuju solusi karena umumnya cukup mudah untuk memperbaiki keadaan yang buruk.

Namun sayangnya, SAHC sering terjebak karena beberapa alasan berikut:

- **Local maximum (maksimum lokal):** Sebuah *local maximum* adalah puncak yang lebih tinggi dari semua keadaan tetangganya, tetapi lebih rendah dari *global maximum*. Algoritma SAHC yang mencapai daerah sekitar *local maximum* akan terdorong naik ke puncak tersebut, namun kemudian terjebak karena tidak ada lagi arah untuk naik
- **Ridge:** Ridge menghasilkan serangkaian *local maximum* yang sangat sulit dilalui oleh greedy algorithm.
- **Plateau:** Plateau adalah area datar pada lanskap ruang keadaan (*state-space landscape*). Area ini bisa berupa *flat local maximum* — yaitu puncak datar tanpa jalan naik sama sekali — atau *shoulder*, yaitu daerah yang masih memungkinkan untuk naik lebih lanjut (Gambar 3.3.1). Pencarian SAHC bisa tersesat di *plateau* ini.

```

1: i = initial solution
2: While f(s) ≤ f(i) s ∈ Neighbours (i) do
3:   Generates an s ∈ Neighbours (i);
4:   If fitness (s) > fitness (i) then
5:     Replace s with the i;
6:   End If

```

Gambar 3.3.2 Pseudocode dari SAHC, arah fitness function dibalik pada Bin Packing[8]

C2. Sideway move hill climbing (SMHC)

Karena untuk menyelesaikan masalah Plateau ini, SAHC hanya dapat menyelesaikan 86% problem SAHC[6]. Salah Satu solusi dari problem ini adalah memperbolehkan sideways move. Akibatnya dilakukan sideway move agar state dapat bergerak dari plateau ke global maximum. Perbedaan pseudocode sari SMHC dan SAHC adalah jika $fitness_baru \geq fitness_lama$, dapat terjadi pertukaran state. Pertukaran state kemudian diconstraint dengan $Max_no_improve$.

C3. Stochastic hill climbing (SHC)

Stochastic hill climbing memilih secara acak langkah-langkah yang menuju ke arah naik, namun akan selalu dipilih arah terbaik. Probabilitas pemilihannya dapat bervariasi tergantung pada tingkat *steepness* dari langkah tersebut. Metode ini biasanya berkonvergensi lebih lambat dibandingkan dengan steepest ascent hill climbing, tetapi pada

beberapa bentuk *state landscape*, metode ini dapat menemukan solusi yang lebih baik.

```

i = initial solution
j = 0
While (j < max_iter)
    Generates neighbours(i);
    s_rand = choose randomly(s);
    if (fitness(s_rand) > fitness(i) ) then
        replace s with i;
    end if
j++

```

Gambar 3.3.2 Pseudocode dari SHC, arah fitness function dibalik pada Bin Packing.

C4. Random restart hill climbing (RRHC)

Algoritma ini melakukan serangkaian pencarian *hill climbing* yang dimulai dari *initial state* yang dihasilkan secara acak, hingga akhirnya ditemukan *goal state*. Algoritma ini dapat dikatakan *trivially complete* dengan probabilitas mendekati 1, karena pada akhirnya algoritma akan menghasilkan keadaan tujuan sebagai *initial state* setelah cukup banyak percobaan acak dilakukan.

```

i = initial solution
j = 0
while (j < max_restart) do
    While (f(s) < f(i)) do
        Generates neighbours(i);
        if fitness(s) > fitness(i) then
            replace s with i;
        end if
        random_restart(bins)
    j++

```

Gambar 3.3.3 Pseudocode dari RRHC, arah fitness function dibalik pada Bin Packing.

D. Simulated Annealing

Sebuah algoritma hill climbing yang tidak pernah melakukan langkah “menurun” menuju keadaan dengan nilai lebih rendah (atau biaya lebih tinggi) dijamin akan tidak lengkap, karena dapat terjebak pada maksimum lokal.

Sebaliknya, random walk murni — yaitu bergerak menuju keadaan penerus yang dipilih secara acak uniform dari seluruh himpunan penerus — bersifat lengkap, tetapi sangat tidak efisien.

Oleh karena itu, masuk akal untuk mencoba menggabungkan hill climbing dengan random walk agar diperoleh algoritma yang memiliki efisiensi sekaligus

kelengkapan. Salah satu algoritma yang melakukan hal ini adalah simulated annealing. Algoritma dari simulated annealing ditampilkan di gambar 3.4.1 dibawah.

Dalam metalurgi, *annealing* adalah proses untuk mengeraskan atau memperkuat logam dan kaca dengan cara memanaskannya hingga suhu tinggi, kemudian mendinginkannya secara bertahap, sehingga material dapat mencapai keadaan kristal dengan energi rendah.

```

function SIMULATED-ANNEALING(problem, schedule) returns a solution state
inputs: problem, a problem
         schedule, a mapping from time to “temperature”

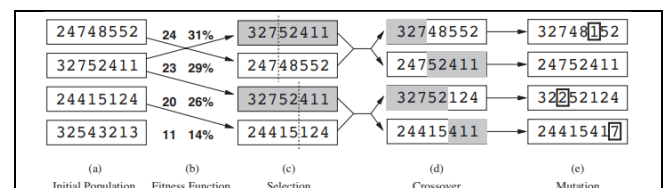
current ← MAKE-NODE(problem.INITIAL-STATE)
for t = 1 to ∞ do
    T ← schedule(t)
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← next.VALUE – current.VALUE
    if ΔE > 0 then current ← next
    else current ← next only with probability  $e^{\Delta E/T}$ 

```

Gambar 3.4.1 Pseudocode dari Simulated Annealing[6][page 126]

E. Genetic Algorithms

Algoritma Genetika (Genetic Algorithms, GAs) dimulai dengan sekumpulan *k* keadaan yang dihasilkan secara acak, yang disebut populasi (population). Setiap keadaan dievaluasi menggunakan fungsi objektif atau dalam terminologi GA, fitness function. Dalam varian algoritma genetika ini, probabilitas terpilih untuk bereproduksi diambil dari fitness threshold.



Gambar 3.5.1 Ilustrasi dari Genetic Algorithm pada [6][page 127]

Selanjutnya dua pasangan dipilih secara acak untuk melakukan reproduksi. Individu baru / child dibuat dengan menyilangkan (crossover) input dari kedua induk pada titik crossover tersebut. Setiap posisi dalam string dapat mengalami mutasi acak dengan probabilitas kecil dan bersifat independen.

Pada saat crossover pada bin packing, karena tiap bin yang ada harus berbeda dilakukan restoration dengan melakukan append weight awal ketika ada weight yang hilang dan pop pada duplikat. Selanjutnya dilakukan SAHC pada child tersebut sehingga hasil yang didapatkan optimal. Adanya SAHC tersebut membuat Genetic Algorithm sangat efektif mencapai global minimum pada generasi dan populasi yang sedikit. Namun tradeoff yang didapatkan adalah waktu run yang lumayan lama (10 second per run). Pada program yang digunakan untuk mengubah menjadi global max digunakan fitness function $F(h) = 1/(1-h)$

Gambar dibawah adalah pseudocode dari Genetic Algorithm :

```

function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
inputs: population, a set of individuals
        FITNESS-FN, a function that measures the fitness of an individual

repeat
    new_population ← empty set
    for i = 1 to SIZE(population) do
        x ← RANDOM-SELECTION(population, FITNESS-FN)
        y ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE(x, y)
        if (small random probability) then child ← MUTATE(child)
        add child to new_population
    population ← new_population
until some individual is fit enough, or enough time has elapsed
return the best individual in population, according to FITNESS-FN

function REPRODUCE(x, y) returns an individual
inputs: x, y, parent individuals
    n ← LENGTH(x); c ← random number from 1 to n
    return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))

```

Gambar 3.5.2 Pseudocode dari Genetic Algorithm[6][page 129]

F. Perhitungan Global Minimum

[9] Mencari global minimum pada Bin Packing Algorithm Secara merupakan NP-Hard Problem. Sebagai contoh pada Algoritma Integer Linear Programming didapatkan time complexity sebesar $2^{O(1/e \log(1/e))}$ Dengan $e = 1/(OPT(I)+1)$. Namun batas bawah dari Bin Packing dengan mudah didapatkan dengan :

$$OPT(i) \leq c - (\sum \text{weight}) \bmod c$$

Sehingga jika ada salah satu nilai heuristic pada percobaan mencapai batas bawah, sudah dipastikan hasil tersebut adalah global minimum berdasarkan *squeeze theorem*.

IV. HASIL DAN DISKUSI

Pada experiment digunakan 3 testcase, 3 weight yang didefinisikan sebagai list of integer berikut :

weights_1 = [2, 5, 4, 7, 1, 3, 6, 8, 9, 2, 5, 3, 7, 4, 6, 1, 8, 9, 2, 5]

weights_2 = [1, 2, 1, 3, 2, 4, 3, 1, 2, 3, 4, 2, 1, 3, 2, 4, 3, 1, 2, 3]

weights_3 = [2, 5, 4, 7, 1, 3, 6, 8, 9, 2, 5, 3, 7, 4, 6, 1, 8, 9, 2, 5, 3, 7, 2, 4, 5]

Berdasarkan metodologi yang digunakan, lower bound ditentukan sebagai berikut :

Tabel 4.1 Lower bound dari masing-masing weights

weights 1	weights 2	weights 3
3	3	2

Berikut adalah variable control yang digunakan :

- Testcase : weights_3
- Iterasi-HC : 100
- Max_sidestep : 10
- Max_restart : 10
- Iterasi Simulated Annealing : 200
- Simulated Annealing Temperature : 100
- Penurunan Temperature : $T * = 0.95$
- Populasi : 10
- Generasi : 10
- Fitness threshold : 10% (diambil 10% terbaik) agar populasi tidak punah
- Mutation rate : 10%

A. Hill Climbing Algorithm

A.1. Steepest ascent hill climbing

Tabel 4.2 plot dari SAHC

TC	Plot	Final value	Bin Used
1		13	11
2		3	5
3		12	13

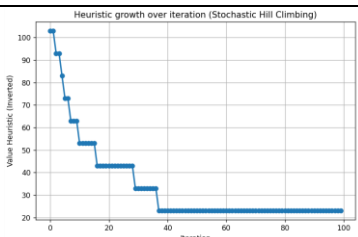
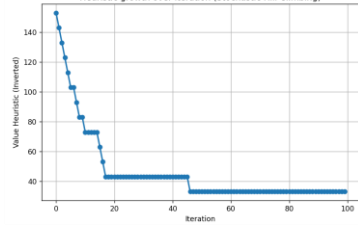
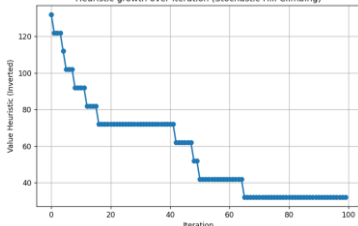
A.2. Sideways Move Hill Climbing

Tabel 4.3 plot dari SMHC

TC	Plot	Final value	Bin Used
1		13	11
2		3	5
3		12	13

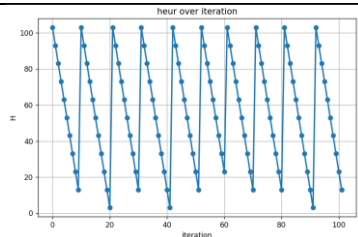
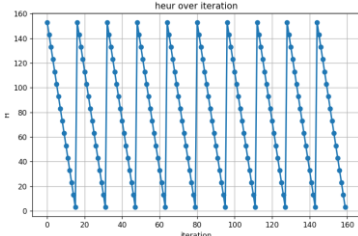
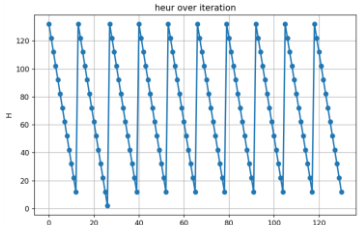
A.3. Stochastic Hill Climbing

Tabel 4.4 plot dari SHC

TC	Plot	Final value	Bin Used
1		23	12
2		33	8
3		32	15

A.4. Random Restart Hill Climbing

Tabel 4.5 plot dari SAHC

TC	Plot	Best val	Bin Used
1		3	10
2		3	5
3		2	12

Tabel 4.6 Hasil Global optimum pada TC_3

Kontainer 1 (total: 10)
 * BRG009 (9)
 * BRG005 (1)
 Kontainer 2 (total: 10)
 * BRG008 (8)
 * BRG001 (2)
 Kontainer 3 (total: 10)
 * BRG013 (7)
 * BRG021 (3)
 Kontainer 4 (total: 10)
 * BRG015 (6)
 * BRG003 (4)
 Kontainer 5 (total: 9)
 * BRG017 (8)
 * BRG016 (1)
 Kontainer 6 (total: 10)
 * BRG007 (6)
 * BRG019 (2)
 * BRG023 (2)
 Kontainer 7 (total: 10)
 * BRG004 (7)
 * BRG012 (3)
 Kontainer 8 (total: 10)
 * BRG022 (7)
 * BRG006 (3)
 Kontainer 9 (total: 10)
 * BRG011 (5)
 * BRG002 (5)
 Kontainer 10 (total: 9)
 * BRG018 (9)
 Kontainer 11 (total: 10)
 * BRG024 (4)
 * BRG010 (2)
 * BRG014 (4)
 Kontainer 12 (total: 10)
 * BRG020 (5)
 * BRG025 (5)

Berdasarkan hasil yang didapatkan diatas urutan dari algoritma yang paling optimal untuk mencapai local minimum dari terbaik ke terburuk adalah :

$$RRHC > SMHC = SAHC > SHC$$

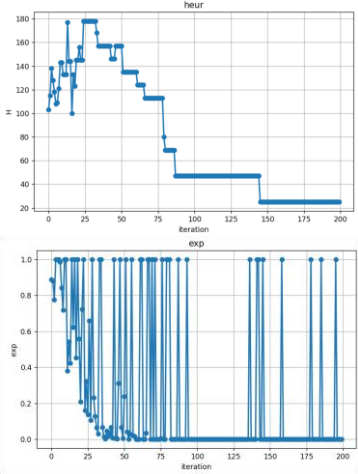
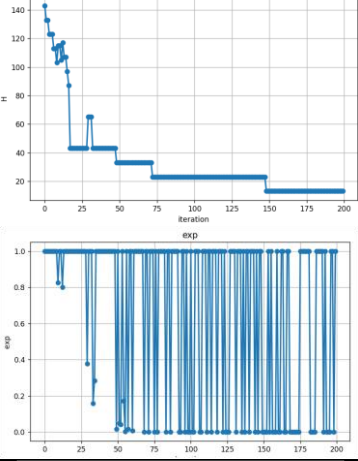
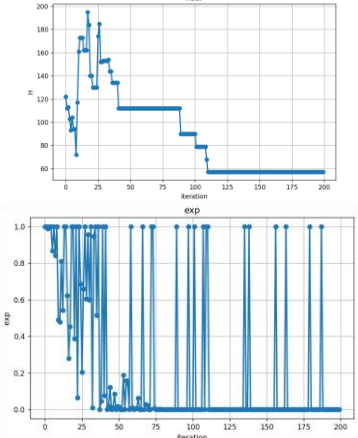
Jika RRHC dipotong ke 100 iterasi Pertama pun RRHC mendapatkan hasil terbaik. Hal ini karena RRHC menggunakan SAHC karena setiap Initial state yang berbeda. Karena SAHC adalah Algoritma greedy, initial state sangat memengaruhi Final state, Hal ini yang mengakibatkan RRHC sangat efektif sehingga menghasilkan local optimal di setiap test case. didapatkan local optimal.

Selain itu SHC merupakan Algoritma terburuk, karena dibutuhkan sangat banyak iterasi untuk mendapatkan hasil yang baik. Sebagai contoh pada Testcase 3 diatas terdapat pada iterasi 65.

Untuk SMHC tidak terdapat problem solving pada Plateaux, hal yang sama juga terjadi saat max_sidewaymove dimaksimalkan. Hal ini karena Greedy algorithm sangat berpengaruh pada initial condition.

B. Simulated Annealing

Tabel 4.8 plot dari Simulated Annealing dan probabilitas Worse move dipilih

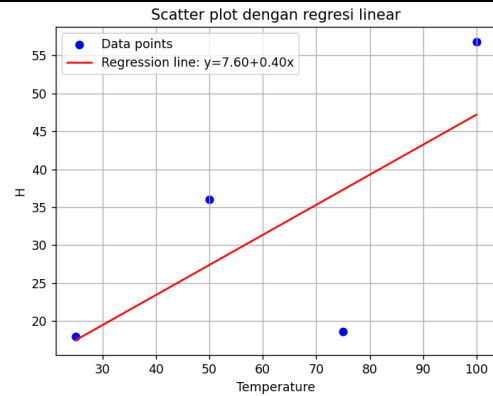
TC	plot	Best val	Bin
1		25	10
2		13	6
3		57	12

Pengaruh temperature

Tabel 4.9 Pengaruh Temperature pada Simulated Annealing

T	Iter 1-5	mean	Std
100	101, 57, 46, 23, 57	56.8	25.35
75	23, 34, 12, 12, 12	18.6	8.8
50	34, 33, 56, 23, 34	36	10.83

25	12, 22, 22, 22, 12	18	4.9
----	--------------------	----	-----



Gambar 4.1 Pengaruh Temperature pada Simulated Annealing

Untuk mendapatkan grafik eksponen yang baik dilakukan limiting pada output (hasil exponen adalah probabilitas sehingga maksimum ada di 1).

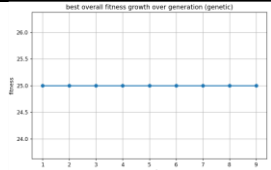
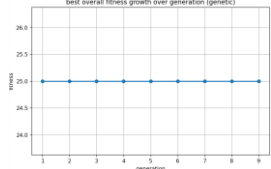
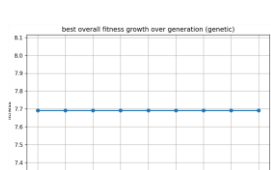
Dari Tabel 4.9 diatas tidak ada satupun hasil dari simulated annealing yang mencapai local Optima. Hal ini dikarenakan terjadi jitter tinggi pada awal algoritma dijalankan sesuai dengan grafik eksponen yang ada. Hal ini karena saat temperature mendekati 0, simulated annealing terjebak pada plateaux (kemungkinan worse dipilih 0). Dibanding RRHC ataupun SHC Simulated Annealing memiliki final value yang lebih buruk.

Berdasarkan grafik 4.1, heuristic hampir berbanding lurus dengan temperatur, sehingga saat temperature naik, H cenderung lebih buruk. Hal ini dikarenakan semakin tinggi temperature, semakin memungkinkan untuk worst neighbor terpilih, sehingga jika diakumulasikan didapatkan hasil yang lebih buruk. Selain itu hasil yang didapatkan juga memiliki Std yang sangat tinggi terutama, terjadi pada temperature tinggi sehingga hasil tidak stabil.

C. Genetic Algorithm

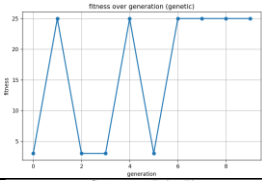
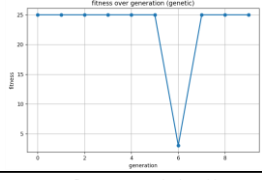
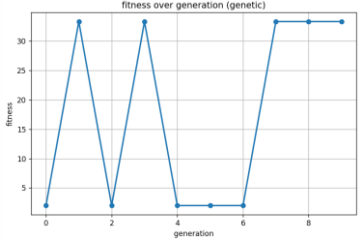
Best H over generation:

Tabel 4.10 Pengaruh testcase pada best H

TC	plot	Best val	Bin
1		3	10
2		3	5
3		12	13

Fitness over generation :

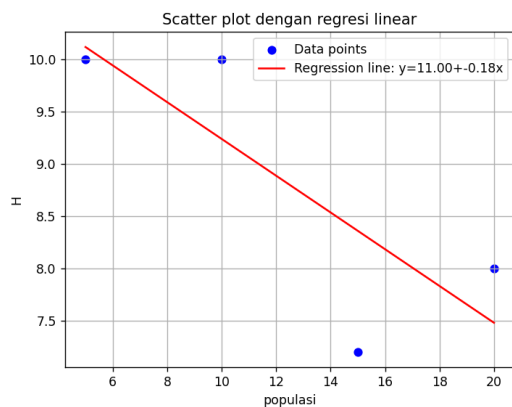
Tabel 4.11 Pengaruh testcase pada Fitness

TC	plot	Best val
1		3
2		3
3		2

Pengaruh Populasi :

Tabel 4.12 Pengaruh populasi pada Heuristic

P	Iterasi 1-5	mean	std
5	12, 2, 12, 12, 12	10	10
10	2, 12, 12, 12, 12	10	4
15	2, 12, 2, 12, 8	7.2	4.49
20	7, 8, 2, 11, 12	8	3.52



Gambar 4.2 Pengaruh populasi pada H Genetic Algorithm

Koefisien korelasi = -0.798

Pengaruh mutation rate :

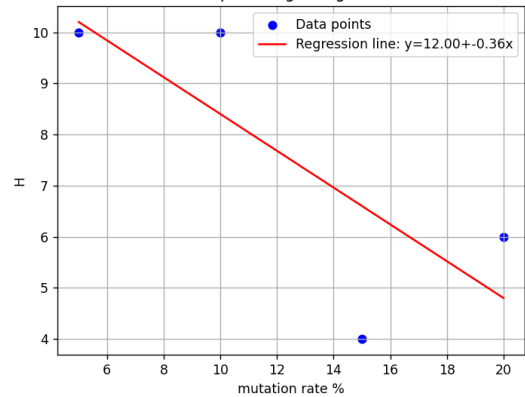
Tabel 4.13 Pengaruh Mutation Rate pada Heuristic

M	Iterasi 1-5	Mean	std
5	12, 12, 2, 12, 12	10	4
10	2, 12, 12, 12, 12	10	4
15	2, 2, 12, 2, 2	4	4

20	2, 2, 12, 12, 2	6	4.9
----	-----------------	---	-----

Koefisien korelasi = -0.775

Scatter plot dengan regresi linear

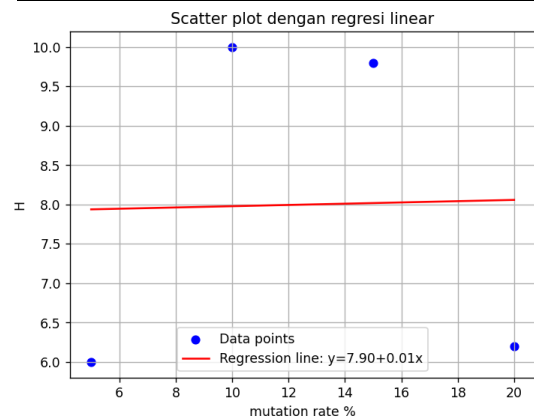


Gambar 4.3 Pengaruh Mutation Rate pada H Genetic Algorithm

Pengaruh Fitness threshold :

Tabel 4.14 Pengaruh Fitness threshold pada Heuristic

T	Iterasi 1-5	mean	std
5	2, 12, 2, 12, 2	6	4.9
10	2, 12, 12, 12, 12	10	4
15	12, 11, 12, 12, 2	9.80	3.92
20	2, 3, 2, 12, 12	6.2	4.75

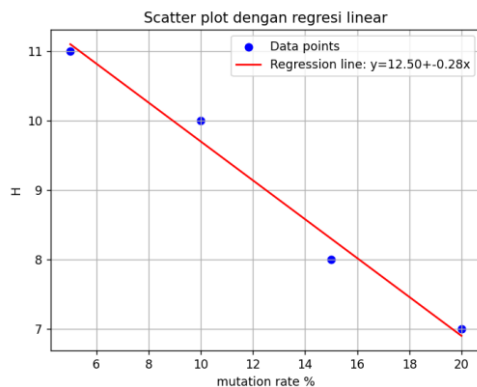


Gambar 4.4 Pengaruh Fitness Threshold pada H Genetic Algorithm

Koefisien korelasi $r = 0.024$ **Pengaruh Generasi :**

Tabel 4.15 Pengaruh Fitness threshold pada Heuristic

G	Iterasi 1-5	mean	std
5	12, 7, 12, 12, 12	11	2
10	2, 12, 12, 12, 12	10	4
15	12, 12, 2, 7, 7	8.0	3.74
20	12, 12, 7, 2, 2	7	4.47



Gambar 4.5 Pengaruh Generasi pada H Genetic Algorithm
Koefisien korelasi $r = -0.990$

Genetic hampir memiliki local optimum untuk setiap testcase, hal ini sesuai dengan metodologi di atas. Selain itu hasil yang didapatkan pada genetic algorithm memiliki nilai yang cukup stabil untuk setiap generasi bahkan pada tabel 4.11 semua testcase sampai pada local optimum pada 1 generasi.

Hampir semua parameter berbanding terbalik dengan jumlah heuristic, kecuali fitness threshold karena. Hal ini diduga karena fitness threshold memperkecil sample sehingga memperkecil kemungkinan child.

Populasi Berbanding terbalik karena, Semakin banyak populasi, semakin banyak searching dilakukan. Jumlah generasi memiliki korelasi terbaik karena hasil Children dari parents selalu memiliki H yang lebih baik. Mutation rate menambahkan mean yang membaik namun memperburuk standard deviation, karena mutation rate menambah ruang pencarian namun memperbanyak sifat stokastik pada algoritma.

D. Time Analysis

Dengan menggunakan variable control diatas, berikut adalah waktu run masing-masing algoritma :

Algoritma	Time Elapsed
SAHC	1.154
SMHC	1.772
SHC	3.971
RRHC	8.375
Simulated Annealing	6.778
Genetic Algorithm	10.399

Pada tabel diatas sesuai dengan prediksi, SAHC sampai ke local optimum tercepat, disusul dengan SMHC karena adanya fungsi max_no_improve. Selanjutnya adalah SHC karena selalu dijalankan searching sesuai max_iterasi. Selanjutnya adalah simulated anealing karena jumlah iterasi yang digunakan 2 kali lipat. RRHC adalah algoritma terlambat kedua karena tidak ditentukan jumlah iterasi, tetapi menjalankan SAHC sejumlah max_restart. Terakhir adalah Genetic Algorithm karena menjalankan SAHC dengan jumlah yang sangat banyak pada fungsi repair dan dilakukan fungsi crossover setiap generasi. Namun Genetic Algorithm merupakan algoritma tercepat untuk mencapai Global optimum.

E. Initial State Analysis

Dengan menggunakan variable control diatas, berikut adalah waktu hasil masing-masing algoritma :

Algoritma	Worst (Final value)	First fit (Final value)
SAHC	12	12
RRHC	2	12
Simulated Annealing	12	113
Genetic Algorithm	2	2

Berdasarkan hasil yang didapatkan, Worst cenderung memiliki hasil yang lebih baik dibanding First-Fit. Berdasarkan analisis yang dilakukan sebelumnya

V. KESIMPULAN

1. Algoritma hill climbing jika diurutkan dari yang terbaik ke terburuk adalah $RRHC > SAHC = SMHC > SHC$. Hal ini dikarenakan SAHC bergantung pada initial state, RRHC membuat initial state yang baru setiap iterasi yang mengakibatkan global optimum mudah tercapai. Adapun algoritma terburuk dimiliki SHC karena waktu konvergen yang dibutuhkan sangat lama.
2. Temperatur berbanding terbalik dengan global optimum pada Simulated annealing karena semakin tinggi temperature, semakin besar entalpi akibatnya semakin mudah worst state terpilih.
3. Populasi, Generasi, %Mutasi, dan %Threshold berbanding Lurus dengan global optimum. Hal ini karena Populasi, %Threshold dan Generasi menambah sample space yang diperiksa. Mutasi menambah variasi/entalpi namun menambah deviasi.
4. Berdasarkan waktu jika diurutkan adalah SAHC, SMHC, SHC, Simulated annealing, RRHC, dan Genetic.
5. First fit cenderung lebih buruk dari Worst karena First fit mempersempit ruang sample walaupun initial state lebih baik dan memiliki kemungkinan mencapai global optimum lebih cepat.

REFERENSI

- [1] Coffman, E. G., Garey, M. R., & Johnson, D. S. (1997). Approximation Algorithms for Bin Packing: A Survey. In D. S. Hochbaum (Ed.), Approximation Algorithms for NP-Hard Problems (pp. 46–93). PWS Publishing Company.
- [2] Aarts, E., & Lenstra, J. K. (Eds.). (2003). Local Search in Combinatorial Optimization. Princeton University Press.
- [3] Kang, Seokchan & Lee, Jiyeong. (2017). Developing a Tile-Based Rendering Method to Improve Rendering Speed of 3D Geospatial Data with HTML5 and WebGL. Journal of Sensors. 2017. 1-11. 10.1155/2017/9781307.
- [4] Alvim, Adriana & Glover, Fred & Ribeiro, Celso. (1999). Local Search For The Bin Packing Problem.
- [5] Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R., & Graham, R. L. (1974). Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM Journal on Computing, 3(4), 299–325. <https://doi.org/10.1137/0203025>.
- [6] Russell, S., & Norvig, P. (2010). Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall.

- [7] Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R., & Graham, R. L. (1974). *Worst-case performance bounds for simple one-dimensional packing algorithms*. SIAM Journal on Computing, 3(4), 299–325. <https://doi.org/10.1137/0203025>.
- [8] Shehab, Mohammad & Khader, Ahamad Tajudin & Makhoulf, Laouchedi. (2018). A hybrid method based on Cuckoo search algorithm for global optimization problems. Journal of Information and Communication Technology. 17. 10.32890/jict2018.17.3.4.
- [9] Jansen, Klaus & Kratsch, Stefan & Marx, Dániel & Schlotter, Ildikó. (2010). Bin Packing with Fixed Number of Bins Revisited. Journal of Computer and System Sciences. 79. 260-272. 10.1007/978-3-642-13731-0_25.