BAB III
METODOLOGI PENELITIAN

Bab ini membahas deskripsi masalah, tahapan penelitian, model optimisasi, dan teknik penyelesaian dengan menggunakan Algoritma *Migrating Birds Optimization* (MBO) dalam masalah HFS di suatu pabrik tekstil.

3.1 Deskripsi Masalah

Penelitian ini membahas masalah penjadwalan HFS di suatu pabrik tekstil. HFS merupakan masalah penjadwalan produksi yang kompleks dan sering ditemui dalam industri manufaktur. Tujuan utama dari penyelesaian HFS adalah untuk mengoptimalkan urutan pekerjaan (*job*) pada setiap mesin dengan meminimumkan waktu penyelesaian total (*makespan*). Terdapat *n* buah *job* di mana setiap *job* harus diproses dalam *m stage*. Pada setiap *stage k* terdapat i_k mesin dengan tipe yang sama. Masalah dalam penjadwalan HFS adalah bagaimana menjadwalkan *job* sedemikian sehingga setiap *job* harus melewati setiap *stage* yang sama dan pada setiap *stage*, setiap mesin hanya akan mengerjakan satu *job* saja. *Job* selanjutnya akan dikerjakan apabila *job* sebelumnya telah selesai dikerjakan. Setiap *stage* memungkinkan memproses beberapa *job* dalam satu waktu, namun pada mesin yang berbeda. Algoritma MBO akan digunakan untuk menyelesaikan masalah penjadwalan HFS ini.

3.2 Tahapan Penelitian

Tahapan penelitian yang dilakukan adalah sebagai berikut:

1. Studi Pustaka

   Tahap ini dilakukan studi pustaka dengan cara mempelajari teori-teori yang berkaitan dengan teori tentang HFS dan Algoritma MBO dari berbagai sumber yang berupa jurnal nasional, jurnal internasional, *proceeding*, buku, dan skripsi.

2. Pengumpulan Data

   Data yang akan digunakan dalam penelitian ini bersifat sekunder yang diperoleh dari jurnal dan sebagian data merupakan data *dummy* yang

menyerupai data asli. Data yang digunakan dalam penelitian ini, yaitu data tahapan produksi, data jumlah *job*, data jumlah mesin dan data *job* yang terdiri dari waktu proses pengerjaan *job* pada tiap-tiap mesin yang digunakan.

3. Pembangunan Model Optimisasi

   Tahapan ini akan dibangun model optimisasi dengan terlebih dahulu mendefinisikan himpunan, indeks, dan parameter yang digunakan dalam model optimisasi.

4. Penyelesaian Masalah

   Tahapan ini akan dibangun model masalah HFS yang akan diselesaikan dengan Algoritma MBO.

5. Validasi

   Sebelum dilakukan implementasi pada masalah HFS di suatu pabrik tekstil, perlu dilakukan validasi terlebih dahulu. Validasi merupakan proses yang dilakukan untuk menguji apakah model optimisasi dan Algoritma MBO yang digunakan sudah benar atau tidak. Validasi dilakukan dengan cara membandingkan solusi yang diperoleh dari metode penyelesaian dengan menggunakan Python 3.11 dan solusi dari perhitungan manual dari suatu kasus HFS berukuran kecil. Jika solusi yang diperoleh sama, maka tahapan akan dilanjutkan ke implementasi. Jika solusi yang dihasilkan berbeda, maka tahapan akan diulang dari pemodelan.

6. Implementasi

   Model optimisasi HFS yang sudah dibangun akan diimplementasikan untuk menyelesaiakan masalah HFS di suatu pabrik tekstil dengan menggunakan Algoritma MBO.

7. Penarikan Kesimpulan

   Tahapan ini akan dilakukan penarikan kesimpulan sebagai ulasan dari hasil penelitian yang telah dilakukan. Hasil penelitian tersebut meliputi kesesuaian hasil penelitian dengan rumusan masalah serta hasil implementasi Algoritma MBO terhadap masalah HFS.

3.3 Model Optimisasi *Hybrid Flowshop Scheduling* (HFS)

Model optimisasi masalah HFS dibangun dengan terlebih dahulu mendefinisikan himpunan, parameter, variabel keputusan, fungsi tujuan, dan fungsi kendala. Adapun asumsi yang diambil dalam penelitian ini, yaitu:

1. Bahan baku selalu tersedia dalam jumlah cukup untuk produksi.
2. Seluruh mesin yang dioperasikan untuk produksi berada dalam kondisi operasional yang prima.
3. Waktu persiapan mesin sebelum operasi telah termasuk dalam kalkulasi waktu proses.
4. Kinerja operator sesuai dengan standar operasional prosedur yang telah ditetapkan.
5. Proses produksi berjalan tanpa adanya kegagalan atau kebutuhan akan perbaikan produk yang cacat.

Berikut adalah himpunan, indeks, dan parameter yang digunakan pada model optimisasi:

- J : Himpunan *job*
- K : Himpunan *stage*
- I_k : Himpunan mesin, di mana *i* sebagai indeks mesin ($i = 1, \ldots, i_k$)
- j : indeks *job* ($j = 1,2, \ldots, n$)
- k : indeks *stage* ($k = 1,2, \ldots, m$)
- i : indeks J
- l : indeks J
- $P_{k,j}$: Waktu pemprosesan *job j* pada *stage k*
- $S_{k,j}$: Waktu mulai *job j* pada *stage k*
- C_{max} : Waktu penyelesaian maksimum dari semua *job*

Variabel keputusan model didefinisikan untuk menentukan mesin mana yang akan mengerjakan setiap *job* pada setiap *stage*. Variabel ini adalah sebagai berikut:

$$X_{k,j,i} = \begin{cases} 1, \text{jika } job\ j \text{ dikerjakan oleh mesin } i \text{ pada } stage\ k \\ 0, \text{yang lainnya.} \end{cases}$$

Karena mesin tersedia dalam jumlah terbatas, maka perlu ada pengurutan *job* yang harus dikerjakan terlebih dahulu. Oleh karena itu didefinisikan variabel keputusan berikut:

$Y_{k,j,l} = \begin{cases} 1, \text{jika } job\ j \text{ dikerjakan terlebih dahulu daripada } job\ l \text{ pada } stage \text{ k.} \\ 0, \text{yang lainnya.} \end{cases}$

Fungsi tujuan dari model optimisasi didefinisikan untuk mengatur jadwal penyelesaian *job* pada mesin sehingga waktu selesainya *job* terakhir seminimum mungkin. Waktu selesai *job* terakhir atau biasa disebut *makespan*, adalah waktu maksimum yang digunakan untuk menyelesaiakan suatu *job*. Dengan demikian, *makespan* dapat dihitung sebagai berikut:

$$C_{max} = max\left(S_{m,j} + P_{m,j}\right), \forall j = 1, \dots, n$$

Sehingga fungsi tujuan model optimisasi adalah sebagai berikut:

Meminimumkan:

$$C_{max}$$

Fungsi kendala pada model optimisasi adalah sebagai berikut:

1. Kendala bahwa setiap *job* harus melewati setiap *stage* dan hanya melewati 1 mesin untuk setiap tipe mesin yang sama. Setiap *job* yang dikerjakan perlu melintasi setiap *stage* yang ada pada jalur produksi, yaitu *m stage*. Setiap *job* tersebut harus dikerjakan pada setiap *stage* dengan urutan yang sama 1 sampai *m.* Selain itu, dalam setiap *stage* setiap *job* hanya melewati 1 mesin. Hal ini merupakan kendala yang nantinya akan digunakan dalam penentuan rute urutan *job* yang harus dikerjakan. Dengan demikian, *job j* harus melewati *stage k* dan mesin *i*. Kendala ini dituliskan sebagai berikut:

$$\sum_{i=1}^{i_k} X_{k,j,i} = 1, \forall j = 1, \dots, n\ ;\ \forall k = 1, \dots, m$$

2. Kendala yang memastikan bahwa waktu mulai *job* pada *stage* pertama lebih besar sama dengan 0. Pada *stage* pertama untuk *job j* mempunyai waktu mulai *job* lebih dari 0. Ini terjadi karena *job j* yang tidak menjadi urutan pertama akan memiliki *job* sebelumnya yang telah dikerjakan. Sehingga *job j* tersebut memiliki waktu awal untuk memulai *job* tersebut dikerjakan. Sehingga *job j* pada *stage* pertama lebih besar sama dengan nol. Kendala ini dituliskan sebagai berikut:

$$S_{1,j} \geq 0, \qquad \forall j = 1, \dots, n$$

3. Kendala untuk menjamin bahwa proses selanjutnya hanya dapat dikerjakan ketika proses pada *stage* sebelumnya untuk *job j* sudah selesai dikerjakan. Pengerjaan *job j* akan berlangsung secara berurutan, dimulai dari *stage* 1

hingga *stage m*. *Job j* hanya akan dilanjutkan ke *stage k+1* setelah semua proses *job j* pada *stage k* telah selesai dikerjakan. Sehingga proses berikutnya hanya dapat dikerjakan ketika proses yang mendahului telah diselesaikan. Kendala ini memastikan bahwa tidak ada *job* yang memulai *stage* berikutnya sebelum menyelesaikan *stage* sebelumnya, sehingga menghindari tumpang tindih dan memastikan kelancaran alur kerja. Kendala ini dituliskan sebagai berikut:

$$S_{k+1,j} - S_{k,j} \geq P_{k,j}, \forall j = 1, \dots, n;\ \forall k = 1, \dots, m$$

4. Kendala untuk menentukan urutan *job* mana yang akan dikerjakan terlebih dahulu. Suatu urutan *job* hanya dapat dijalankan ketika terpenuhi antara *job j* dan *job l*, mana *job* yang akan dikerjakan terlebih dahulu dan mana *job* yang akan dikerjakan setelahnya. Sehingga ketika *job j* dikerjakan terlebih dahulu sebelum *job l* pada *stage* yang sama maka tidak mungkin *job l* mendahului *job j*. Kendala ini memastikan bahwa pada setiap *stage k*, antara dua *job j* dan *job l*, hanya satu dari dua kondisi berikut yang dapat terjadi, yaitu *job j* dilakukan sebelum *job l* atau *job l* dilakukan sebelum *job j*. Bertujuan untuk menghindari kemungkinan terjadinya konflik penjadwalan di mana kedua *job* tersebut dijadwalkan pada posisi yang sama atau terjadi penjadwalan ganda. Kendala ini dituliskan sebagai berikut:

$$Y_{k,j,l} - Y_{k,l,j} \leq 1, \forall j = 1, \dots, n, \forall l = 1, \dots, n;\ \forall k = 1, \dots, m$$

Adapun batasan variabel keputusan model adalah sebagai berikut:

$$X_{k,j,i}, Y_{k,j,l} \in \{0,1\}, \forall j = 1, \dots, n;\ \forall l = 1, \dots, n;\ \forall k = 1, \dots, m;$$
$$\forall i = 1, \dots, k$$

Berdasarkan uraian fungsi tujuan dan fungsi kendala pada penjadwalan HFS tersebut, maka dapat disajikan formulasi model optimisasi secara lengkap sebagai berikut:

Meminimumkan:

$$C_{max}$$

Terhadap:

$$\sum_{i=1}^{i_k} X_{k,j,i} = 1, \forall j = 1, \dots, n\ ;\ \forall k = 1, \dots, m$$

$$S_{1,j} \geq 0, \forall j = 1, \dots, n$$

$$S_{k+1,j} - S_{k,j} \geq P_{k,j}, \forall j = 1, \dots, n;\ \forall k = 1, \dots, m$$

$$Y_{k,j,l} - Y_{k,l,j} \leq 1, \forall j = 1, \dots, n;\ \forall l = 1, \dots, n;\ \forall k = 1, \dots, m$$

$$X_{k,j,i}, Y_{k,j,l} \in \{0,1\}, \forall j = 1, \dots, n;\ \forall l = 1, \dots, n;\ \forall k = 1, \dots, m;\ \forall i = 1, \dots, k$$

Pada bagian selanjutnya akan dijelaskan penyelesaian model dengan menggunakan Algoritma MBO.

3.4 Teknik Penyelesaian Masalah HFS dengan Algoritma MBO

Algoritma MBO pertama kali dikenalkan oleh Duman dkk. (2011) terinspirasi dari pola migrasi burung yang membentuk formasi "V" di langit, yang mana formasi tersebut membantu burung terbang dengan lebih hemat energi. Manfaat dari formasi "V" tersebut adalah memungkinkan burung untuk menempuh perjalanan jarak jauh. Penelitian ini menggunakan Algoritma MBO untuk menyelesaikan masalah HFS. Hasil dari penelitian ini adalah berupa jadwal produksi dengan total waktu penyelesaian yang minimum. Tahapan-tahapan dari algortima MBO untuk menyelesaikan masalah HFS adalah sebagai berikut:

1. Inisialisasi Parameter

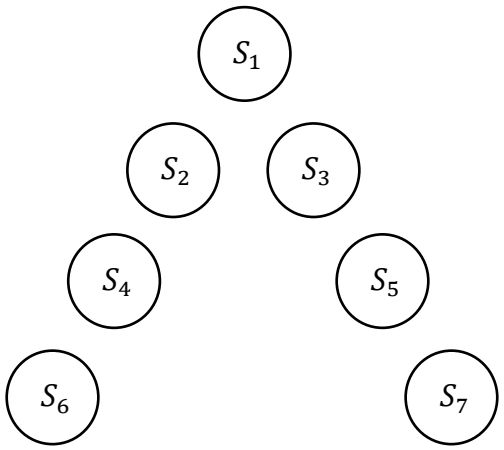
   Parameter yang digunakan pada Algoritma MBO terdiri dari jumlah populasi (n), banyaknya solusi tetangga (k), nilai *sharing* (x), dan iterasi maksimum (m). Penetapan jumlah populasi (n) pada Algoritma MBO dilakukan secara bebas dengan syarat jumlah populasi tersebut adalah ganjil (minimal 3) dan dapat dibentuk menjadi formasi "V".

2. Pembangkitan Populasi Awal Burung

   Proses pembangkitan populasi awal burung diawali dengan melakukan pembangkitan elemen-elemen burung yang terdiri atas bilangan *real* pada interval [0,1] secara acak. Banyaknya burung yang dibangkitkan adalah sebanyak jumlah *job* dikalikan dengan jumlah *stage*. Proses ini dilakukan sebanyak populasi awal burung.

3. Pembentukan Formasi Awal

Setelah populasi awal burung terbentuk, tahapan selanjutnya adalah pembentukan formasi awal dalam bentuk formasi "V". Nomor posisi satu mewakili solusi pemimpin, nomor posisi ganjil mewakili sisi kanan dan nomor posisi genap mewakili sisi kiri. Gambar 3.1 menunjukkan gambaran formasi "V" pada Algoritma MBO.



Gambar 3.1 Gambaran Formasi "V" Algoritma MBO

4. Evaluasi Awal

   Tahapan ini dilakukan evaluasi pada setiap burung dengan cara menghitung *makespan* (C_{max}). Kemudian mencari nilai fungsi tujuan, yaitu meminimumkan *makespan*.

5. Pembangkitan Solusi Tetangga

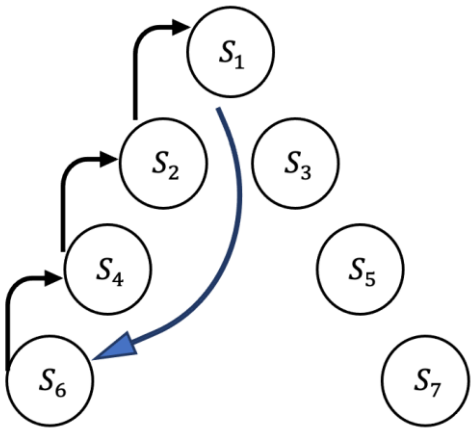
   Solusi burung yang dibangkitkan secara acak sebelumnya akan digunakan sebagai solusi awal. Solusi awal tersebut akan dicari solusi tetangganya dengan cara *swap* atau yang sering dikenal dengan *swapping permutation* sebanyak jumlah solusi tetangga (*k*) yang telah ditentukan. Proses *swapping* ini bekerja dengan menukar *job* di posisi *l* dengan *job* di posisi *j* secara acak.

6. Proses Migrasi

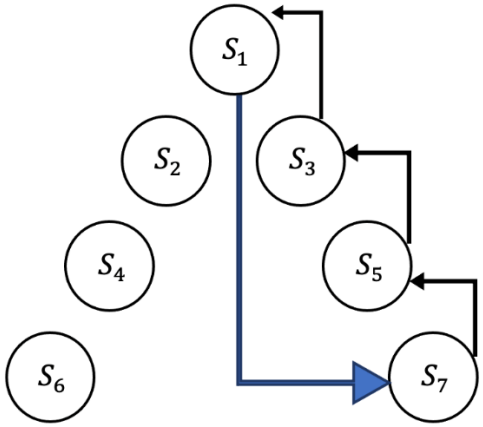
   Hitung kembali *makespan* dari semua solusi tetangga. *Makespan* dihitung dengan cara yang sama seperti Langkah 4. Solusi tetangga terbaik yang sudah didapatkan akan digunakan untuk memperbaiki solusi saat ini. Sisa *x* solusi tetangga dibagikan dengan burung berikutnya. Solusi tetangga terbaik pada burung selain burung pemimpin akan dipakai dan solusi yang tersisa akan dibagikan (*sharing*) ke burung di sekitarnya.

7. Evaluasi dan Pembaruan Pemimpin

Setelah migrasi, evaluasi kembali setiap burung. Jika burung mana pun memiliki nilai yang lebih baik daripada pemimpin saat ini, maka burung tersebut menjadi pemimpin baru. Pembaruan pemimpin dari sisi kanan atau sisi kiri dapat di lihat pada Gambar 3.2 dan Gambar 3.3.



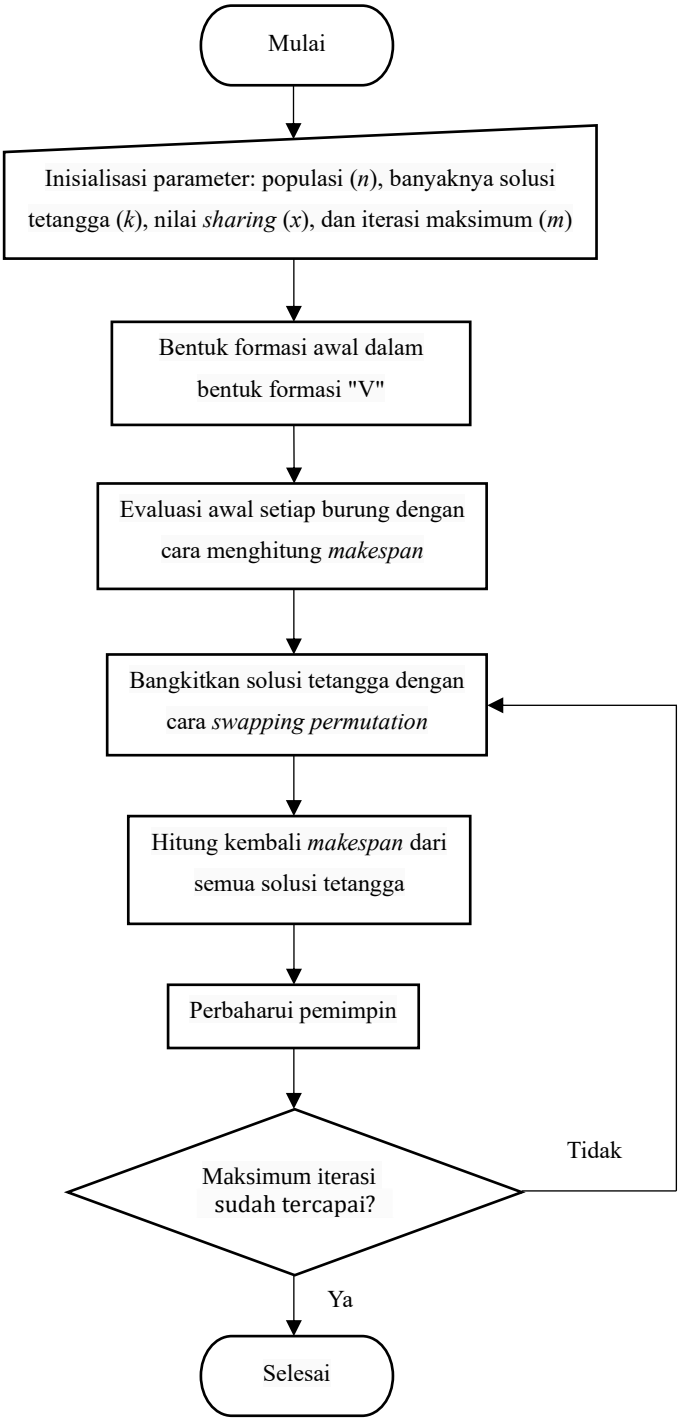
Gambar 3.2 Gambaran Pembaruan Pemimpin Sisi Kiri



Gambar 3.3 Gambaran Pembaruan Pemimpin Sisi Kanan

8. Kriteria Pemberhentian

Ulangi proses migrasi dan pembaruan solusi untuk sejumlah iterasi yang ditentukan atau sampai kriteria konvergensi terpenuhi. Apabila iterasi telah mencapai iterasi maksimum, maka proses dihentikan dan solusi terbaik dihasilkan. Berdasarkan pemaparan tersebut, maka langkah-langkah Algoritma MBO dalam menyelesaikan masalah HFS dapat digambarkan sebagai *flowchart* pada Gambar 3.4



Gambar 3.4 *Flowchart* Algoritma MBO.

3.5 Contoh Kasus

Untuk memperjelas pembahasan langkah-langkah dari Algoritma MBO, berikut ini diberikan contoh kasus sederhana masalah penjadwalan HFS yang akan diselesaikan dengan menggunakan Algoritma MBO. Misalkan terdapat 4 *job*, 3 *stage*, dan 2 mesin pada setiap *stage*. Akan dicari urutan pengerjaan *job* terbaik sehingga menghasilkan *makespan* atau total keseluruhan wartu pengerjaan *job* yang minimum, di mana waktu pengerjaan masing-masing *job* pada mesin di setiap *stage* yang disajikan pada Tabel 3.1.

Tabel 3.1 Data Contoh Penjadwalan HFS

	***Stage* 1**	***Stage* 2**	***Stage* 3**
***Job* 1**	2	4	1
***Job* 2**	5	4	2
***Job* 3**	2	1	5
***Job* 4**	2	2	3
Mesin	2	2	2

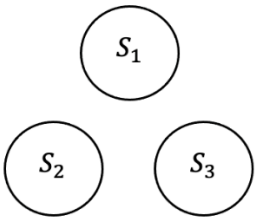
Sumber: Cui, Z. & Gu, X. (2015)

Tabel 3.2 Populasi Awal Burung

Bird	***Stage* 1**				***Stage* 2**				***Stage* 3**			
S_1	0,38	0,65	0,20	0,40	0,68	0,29	0,64	0,35	0,16	0,61	0,87	0,78
S_2	0,35	0,89	0,51	0,65	0,48	0,75	0,22	0,84	0,50	0,02	0,20	0,21
S_3	0,85	0,86	0,28	0,12	0,55	0,19	0,17	0,89	0,04	0,16	0,63	0,26

Selanjutnya, masalah tersebut akan diselesaikan dengan Algoritma MBO, dengan menggunakan langkah berikut:

1. Misalkan ditetapkan parameter-parameter yang akan digunakan dalam menyelesaikan contoh kasus, yaitu jumlah populasi (n) = 3, solusi tetangga (k) = 3, nilai *sharing* (x) = 1, dan iterasi maksimum (m) = 1.
2. Bangkitkan populasi awal burung dengan melakukan pembangkitan elemen-elemen burung yang terdiri atas bilangan *real* pada interval [0,1] secara acak. Hasil pembangkitan burung selengkapnya disajikan dalam Tabel 3.2.
3. Bentuk formasi awal dalam bentuk formasi "V". Populasi awal burung memiliki 3 burung, yaitu S_1, S_2, dan S_3. Gambar 3.5 menunjukkan gambaran formasi "V" pada Algoritma MBO untuk contoh kasus.



Gambar 3.5 Formasi Solusi Awal

Setiap burung memiliki elemen sebanyak jumlah *job* = 4 dikalikan dengan jumlah *stage* = 3. Sehingga setiap burung memiliki 12 elemen. Hal ini dikarenakan setiap burung terbagi atas 3 solusi penjadwalan. Solusi yang pertama terdiri dari 4 elemen yang mewakili urutan dari 4 *job* yang dikerjakan pada *stage* 1 dan begitu juga dengan elemen-elemen yang terdapat pada solusi kedua dan ketiga. Setelah populasi awal diperoleh, langkah selanjutnya adalah evaluasi setiap burung.

4. Hitung *makespan* (C_{max}) kemudian cari nilai fungsi tujuan, yaitu meminimumkan *makespan*. Namun sebelumnya dilakukan proses pengurutan bilangan acak mulai dari yang terkecil hingga yang terbesar pada setiap *stage* tanpa mengubah letak tiap elemen burung tersebut. Kemudian ditransformasi menjadi bilangan bulat sesuai dengan pengurutan yang telah dilakukan, sehingga diperoleh suatu urutan penjadwalan pengerjaan *job*. Pengurutan dan transformasi burung dari populasi awal pada Tabel 3.2 disajikan pada Tabel 3.3.

Tabel 3.3 Pengurutan dan Transformasi Bilangan Acak

Bird	***Stage* 1**				***Stage* 2**				***Stage* 3**			
S_1	0,38	0,65	0,20	0,40	0,68	0,29	0,64	0,35	0,16	0,61	0,87	0,78
	2	4	1	3	4	1	3	2	1	2	4	3
S_2	0,35	0,89	0,51	0,65	0,48	0,75	0,22	0,84	0,50	0,02	0,20	0,21
	1	4	2	3	2	3	1	4	4	1	2	3
S_3	0,85	0,86	0,28	0,12	0,55	0,19	0,17	0,89	0,04	0,16	0,63	0,26
	3	4	2	1	3	2	1	4	1	2	4	3

Selanjutnya, dilakukan evaluasi terhadap masing-masing *bird* (burung) dengan menghitung nilai fungsi tujuan, yaitu meminimumkan *makespan*. Dari Tabel 3.3 diketahui bahwa untuk burung pertama (S_1) diperoleh urutan penjadwalan

pengerjaan *job* adalah 2 4 1 3 - 4 1 3 2 - 1 2 4 3. Berikut ini proses perhitungan *makespan* secara manual untuk setiap *stage* dari setiap burung pada Tabel 3.3:

1. *Stage* pertama adalah 2 4 1 3
    a. *Job* pertama adalah *job* 2 yang dikerjakan pada mesin 1. Waktu pengerjaan *job* 2 di mesin 1, yaitu 5.
    b. *Job* kedua adalah *job* 4 yang dikerjakan pada mesin 2. Waktu pengerjaan *job* 4 di mesin 2, yaitu 2.
    c. *Job* ketiga adalah *job* 1. Untuk menentukan mesin yang mengerjakan *job* 1, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 2, maka *job* 1 dikerjakan pada mesin 1. Kemudian nilai tersebut dijumlahkan dengan waktu pengerjaan *job* 1, yaitu 2 + 2 = 4.
    d. *Job* keempat adalah *job* 3. Untuk menentukan mesin yang mengerjakan *job* 3, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 4, maka *job* 3 dikerjakan pada mesin 2. Kemudian nilai tersebut dijumlahkan dengan waktu pengerjaan *job* 3 pada saat ini. Jadi, waktu selesai *job* 3, yaitu 4 + 2 = 6.
2. *Stage* kedua yaitu 4 1 3 2
    a. *Job* pertama adalah *job* 4 yang dikerjakan pada mesin 1. Waktu pengerjaan diperoleh dari penjumlahan waktu pengerjaan *job* 4 pada *stage* sebelumnya dengan waktu pengerjaan *job* 4 saat ini, yaitu 2 + 2 = 4.
    b. *Job* kedua adalah *job* 1 yang dikerjakan pada mesin 2. Waktu pengerjaan diperoleh dari penjumlahan waktu pengerjaan *job* 1 pada *stage* sebelumnya dengan waktu pengerjaan *job* 1 saat ini, yaitu 4 + 4 = 8.
    c. *Job* ketiga adalah *job* 3. Untuk menentukan mesin yang mengerjakan *job* 3, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 4, maka *job* 3 dikerjakan pada mesin 2. Waktu pengerjaan *job* ketiga diperoleh dari perbandingan waktu

pengerjaan *job* 3 pada *stage* sebelumnya dengan *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} dijumlahkan dengan waktu pengerjaan *job* 3 pada *stage* saat ini, yaitu 6 + 1 = 7.

d. *Job* keempat adalah *job* 2. Untuk menentukan mesin yang mengerjakan *job* 2, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu keteresediaan mesin 2} = 7, maka *job* 2 dikerjakan pada mesin 2. Waktu pengerjaan *job* keempat diperoleh dari perbandingan waktu pengerjaan *job* 2 pada *stage* sebelumnya dengan *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2}, pilih nilai yang lebih besar lalu ditambah dengan waktu pengerjaan *job* 2 pada saat ini, yaitu 7 + 4 = 11.

3. *Stage* ketiga adalah 1 2 4 3

a. *Job* pertama adalah *job* 1 yang dikerjakan pada mesin 1. Waktu pengerjaan diperoleh dari penjumlahan waktu pengerjaan *job* 1 pada *stage* sebelumnya dengan waktu pengerjaan *job* 1 saat ini, yaitu 8 + 1 = 9.

b. *Job* kedua adalah *job* 2 yang dikerjakan pada mesin 2. Waktu pengerjaan diperoleh dari penjumlahan waktu pengerjaan *job* 2 pada saat *stage* sebelumnya dengan waktu pengerjaan *job* 2 saat ini, yaitu 11 + 2 = 13.

c. *Job* ketiga adalah *job* 4. Untuk menentukan mesin mana yang mengerjakan *job* 4, carilah nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 9, maka *job* 4 dikerjakan pada mesin 2. Waktu pengerjaan *job* ketiga diperoleh dari perbandingan waktu pengerjaan *job* 4 pada *stage* sebelumnya dengan *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} dijumlahkan dengan waktu pengerjaan *job* 4 pada *stage* saat ini, yaitu 9 + 3 = 12.

d. *Job* keempat adalah *job* 3. Untuk menentukan mesin yang mengerjakan *job* 3, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1,

waktu ketersediaan mesin 2} = 12, maka *job* 3 dikerjakan pada mesin 2. Waktu pengerjaan *job* keempat diperoleh dari perbandingan waktu pengerjaan *job* 3 pada *stage* sebelumnya dengan *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2}, pilih nilai yang lebih besar lalu dijumlahkan dengan waktu pengerjaan *job* 3 pada saat ini, yaitu 12 + 7 = 19.

Berdasarkan perhitungan tersebut, diperoleh *makespan* untuk burung, yaitu $C_{max} = 19$ satuan waktu. Langkah perhitungan tersebut juga dilakukan untuk S_2. Dari Tabel 3.3 diketahui bahwa untuk burung kedua (S_2) diperoleh urutan penjadwalan pengerjaan *job* adalah 1 4 2 3 - 2 3 1 4 - 4 1 2 3. Berikut ini proses perhitungan *makespan* secara manual untuk setiap *stage* dari setiap burung pada Tabel 3.3:

1. *Stage* pertama adalah 1 4 2 3
   a. *Job* pertama adalah *job* 1 yang dikerjakan pada mesin 1. Waktu pengerjaan *job* 1 di mesin 1, yaitu 2.
   b. *Job* kedua adalah *job* 4 yang dikerjakan pada mesin 2. Waktu pengerjaan *job* 4 di mesin 2, yaitu 2.
   c. *Job* ketiga adalah *job* 2. Untuk menentukan mesin yang mengerjakan *job* 2, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 2, maka *job* 2 dikerjakan pada mesin 1. Kemudian nilai tersebut dijumlahkan dengan waktu pengerjaan *job* 2, yaitu 2 + 5 = 7.
   d. *Job* keempat adalah *job* 3. Untuk menentukan mesin yang mengerjakan *job* 3, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 2, maka *job* 3 dikerjakan pada mesin 2. Kemudian nilai tersebut dijumlahkan dengan waktu pengerjaan *job* 3 pada saat ini. Jadi, waktu selesai *job* 3, yaitu 2 + 2 = 4.
2. *Stage* kedua yaitu 2 3 1 4
   a. *Job* pertama adalah *job* 2 yang dikerjakan pada mesin 1. Waktu pengerjaan diperoleh dari penjumlahan waktu pengerjaan *job* 2 pada

stage sebelumnya dengan waktu pengerjaan *job* 2 saat ini, yaitu 7 + 4 = 11.

b. *Job* kedua adalah *job* 3 yang dikerjakan pada mesin 2. Waktu pengerjaan diperoleh dari penjumlahan waktu pengerjaan *job* 3 pada *stage* sebelumnya dengan waktu pengerjaan *job* 3 saat ini, yaitu 4 + 1 = 5.

c. *Job* ketiga adalah *job* 1. Untuk menentukan mesin yang mengerjakan *job* 1, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 5, maka *job* 1 dikerjakan pada mesin 2. Waktu pengerjaan *job* ketiga diperoleh dari perbandingan waktu pengerjaan *job* 1 pada *stage* sebelumnya dengan *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} dijumlahkan dengan waktu pengerjaan *job* 1 pada *stage* saat ini, yaitu 5 + 4 = 9.

d. *Job* keempat adalah *job* 4. Untuk menentukan mesin yang mengerjakan *job* 4, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu keteresediaan mesin 2} = 9, maka *job* 2 dikerjakan pada mesin 2. Waktu pengerjaan *job* keempat diperoleh dari perbandingan waktu pengerjaan *job* 4 pada *stage* sebelumnya dengan *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2}, pilih nilai yang lebih besar lalu ditambah dengan waktu pengerjaan *job* 2 pada saat ini, yaitu 9 + 2 = 11.

3. *Stage* ketiga adalah 4 1 2 3

a. *Job* pertama adalah job 4 yang dikerjakan pada mesin 1. Waktu pengerjaan diperoleh dari penjumlahan waktu pengerjaan *job* 4 pada *stage* sebelumnya dengan waktu pengerjaan *job* 4 saat ini, yaitu 11 + 3 = 14.

b. *Job* kedua adalah *job* 1 yang dikerjakan pada mesin 2. Waktu pengerjaan diperoleh dari penjumlahan waktu pengerjaan *job* 1 pada saat *stage* sebelumnya dengan waktu pengerjaan *job* 2 saat ini, yaitu 9 + 1 = 10.

c. *Job* keempat adalah *job* 3. Untuk menentukan mesin mana yang mengerjakan *job* 2, carilah nilai minimum antara waktu ketersediaan

mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 10, maka *job* 2 dikerjakan pada mesin 2. Waktu pengerjaan *job* ketiga diperoleh dari perbandingan waktu pengerjaan *job* 2 pada *stage* sebelumnya dengan *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} dijumlahkan dengan waktu pengerjaan *job* 1 pada *stage* saat ini, yaitu 10 + 2 = 12.

d. *Job* keempat adalah *job* 3. Untuk menentukan mesin yang mengerjakan *job* 3, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 12, maka *job* 3 dikerjakan pada mesin 2. Waktu pengerjaan *job* keempat diperoleh dari perbandingan waktu pengerjaan *job* 3 pada *stage* sebelumnya dengan *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2}, pilih nilai yang lebih besar lalu dijumlahkan dengan waktu pengerjaan *job* 3 pada saat ini, yaitu 12 + 5 = 17.

Berdasarkan perhitungan tersebut, diperoleh *makespan* untuk burung, yaitu $C_{max} = 17$ satuan waktu. Langkah perhitungan tersebut juga dilakukan untuk S_3. Dari Tabel 3.3 diketahui bahwa untuk burung ketiga (S_3) diperoleh urutan penjadwalan pengerjaan *job* adalah 3 4 2 1 - 3 2 1 4 - 1 2 4 3. Berikut ini proses perhitungan *makespan* secara manual untuk setiap *stage* dari setiap burung pada Tabel 3.3:

1. *Stage* pertama adalah 3 4 2 1
   a. *Job* pertama adalah *job* 3 yang dikerjakan pada mesin 1. Waktu pengerjaan *job* 3 di mesin 1, yaitu 2.
   b. *Job* kedua adalah *job* 4 yang dikerjakan pada mesin 2. Waktu pengerjaan *job* 4 di mesin 2, yaitu 2.
   c. *Job* ketiga adalah *job* 2. Untuk menentukan mesin yang mengerjakan *job* 2, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 2, maka *job* 2 dikerjakan pada mesin 1.

Kemudian nilai tersebut dijumlahkan dengan waktu pengerjaan *job* 2, yaitu 2 + 5 = 7.

d. *Job* keempat adalah *job* 1. Untuk menentukan mesin yang mengerjakan *job* 1, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 2, maka *job* 1 dikerjakan pada mesin 2. Kemudian nilai tersebut dijumlahkan dengan waktu pengerjaan *job* 1 pada saat ini. Jadi, waktu selesai *job* 1, yaitu 2 + 2 = 4.

2. *Stage* kedua yaitu 3 2 1 4

a. *Job* pertama adalah *job* 3 yang dikerjakan pada mesin 1. Waktu pengerjaan diperoleh dari penjumlahan waktu pengerjaan *job* 3 pada *stage* sebelumnya dengan waktu pengerjaan *job* 3 saat ini, yaitu 2 + 1 = 3.

b. *Job* kedua adalah *job* 2 yang dikerjakan pada mesin 2. Waktu pengerjaan diperoleh dari penjumlahan waktu pengerjaan *job* 2 pada *stage* sebelumnya dengan waktu pengerjaan *job* 2 saat ini, yaitu 7 + 4 = 11.

c. *Job* ketiga adalah *job* 1. Untuk menentukan mesin yang mengerjakan *job* 1, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 3 maka *job* 1 dikerjakan pada mesin 2. Waktu pengerjaan *job* ketiga diperoleh dari perbandingan waktu pengerjaan *job* 1 pada *stage* sebelumnya dengan *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} dijumlahkan dengan waktu pengerjaan *job* 1 pada *stage* saat ini, yaitu 4 + 4 = 8.

d. *Job* keempat adalah *job* 4. Untuk menentukan mesin yang mengerjakan *job* 4, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu keteresediaan mesin 2} = 8, maka *job* 4 dikerjakan pada mesin 1. Waktu pengerjaan *job* keempat diperoleh dari perbandingan waktu pengerjaan *job* 4 pada *stage* sebelumnya dengan *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2}, pilih nilai yang

lebih besar lalu ditambah dengan waktu pengerjaan *job* 4 pada saat ini, yaitu 8 + 2 = 10.

3. *Stage* ketiga adalah 1 2 4 3
   a. *Job* pertama adalah *job* 1 yang dikerjakan pada mesin 1. Waktu pengerjaan diperoleh dari penjumlahan waktu pengerjaan *job* 1 pada *stage* sebelumnya dengan waktu pengerjaan *job* 1 saat ini, yaitu 8 + 1 = 9.
   b. *Job* kedua adalah *job* 2 yang dikerjakan pada mesin 2. Waktu pengerjaan diperoleh dari penjumlahan waktu pengerjaan *job* 2 pada saat *stage* sebelumnya dengan waktu pengerjaan *job* 2 saat ini, yaitu 11 + 2 = 13.
   c. *Job* ketiga adalah *job* 4. Untuk menentukan mesin mana yang mengerjakan *job* 4, carilah nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 9, maka *job* 4 dikerjakan pada mesin 1. Waktu pengerjaan *job* ketiga diperoleh dari perbandingan waktu pengerjaan *job* 4 pada *stage* sebelumnya dengan *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} dijumlahkan dengan waktu pengerjaan *job* 4 pada *stage* saat ini, yaitu 10 + 3 = 13.
   d. *Job* keempat adalah *job* 3. Untuk menentukan mesin yang mengerjakan *job* 3, maka dicari nilai minimum antara waktu ketersediaan mesin 1 dan waktu ketersediaan mesin 2. Karena *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2} = 13, maka *job* 3 dikerjakan pada mesin 2. Waktu pengerjaan *job* keempat diperoleh dari perbandingan waktu pengerjaan *job* 3 pada *stage* sebelumnya dengan *min*{waktu ketersediaan mesin 1, waktu ketersediaan mesin 2}, pilih nilai yang lebih besar lalu dijumlahkan dengan waktu pengerjaan *job* 3 pada saat ini, yaitu 13 + 5 = 18.

Berdasarkan perhitungan tersebut, diperoleh *makespan* untuk burung, yaitu $C_{max} = 18$ satuan waktu. Hasil perhitungan nilai *makespan* untuk seluruh burung disajikan pada Tabel 3.4.

Tabel 3.4 Nilai *Makespan* Setiap Burung

Bird	**Urutan *Job***	***Makespan* (detik)**
S_1	2 4 1 3 - 4 1 3 2 - 1 2 4 3	19
S_2	1 4 2 3 - 2 3 1 4 - 4 1 2 3	17
S_3	3 4 2 1 - 3 2 1 4 - 1 2 4 3	18

5. Bangkitkan solusi tetangga sebanyak jumlah solusi tetangga (k) = 3 dengan cara *swapping permutation* secara acak. Hasil dari pembangkitan solusi tetangga dari contoh kasus dapat dilihat pada Tabel 3.5.
6. Hitung kembali *makespan* dari semua solusi tetangga dengan cara yang sama seperti Langkah 4, selengkapnya dapat dilihat pada Lampiran 2. Solusi tetangga terbaik pada burung selain burung pemimpin akan dipakai dan solusi yang tersisa akan dibagikan (*sharing*) ke burung berikutnya, yaitu N_{12} dibagikan ke N_{33} dan N_{13} dibagikan ke N_{23}. Hasil perhitungan nilai *makespan* untuk seluruh solusi tetangga dari contoh kasus disajikan pada Tabel 3.6.

Tabel 3.5 Solusi Tetangga dari Setiap Burung untuk k = 3 dan x = 1.

Bird	**Permutasi**	***Makespan* (detik)**	**Ketetanggaan**	**Permutasi**
S_1	2 4 1 3 - 4 1 3 2 - 1 2 4 3	19	N_{11}	1 3 2 4 - 4 3 1 2- 4 2 1 3
			N_{12}	2 3 1 4 - 4 2 3 1 - 1 3 4 2
			N_{13}	1 2 3 4 - 4 2 1 3 - 2 4 1 3
S_2	1 4 2 3 - 2 3 1 4 - 4 1 2 3	17	N_{21}	3 2 1 4 - 1 3 2 4 - 3 4 2 1
			N_{22}	2 1 3 4 - 4 3 1 2 - 2 3 4 1
			N_{23}	4 1 2 3 - 2 3 1 4 - 4 1 2 3
S_3	3 4 2 1 - 3 2 1 4 - 1 2 4 3	18	N_{31}	3 1 2 4 - 2 3 1 4 - 3 2 4 1
			N_{32}	1 3 2 4 - 3 2 4 1 - 4 3 1 2
			N_{33}	2 4 3 1 - 4 2 3 1 - 1 3 4 2

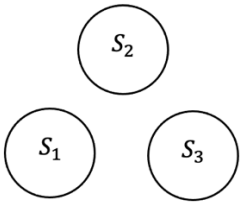
Tabel 3.6 *Makespan* yang Dihasilkan dari Solusi Tetangga

Bird	**Permutasi**	***Makespan* (detik)**	**Ketetanggaan**	**Permutasi**	***Makespan* (detik)**
S_1	2 4 1 3 - 4 1 3 2 - 1 2 4 3	19	N_{11}	1 3 2 4 - 4 3 1 2- 4 2 1 3	15
			N_{12}	2 3 1 4 - 4 2 3 1 - 1 3 4 2	15
			N_{13}	1 2 3 4 - 4 2 1 3 - 2 4 1 3	15
S_2	1 4 2 3 - 2 3 1 4 - 4 1 2 3	17	N_{21}	3 2 1 4 - 1 3 2 4 - 3 4 2 1	13
			N_{22}	2 1 3 4 - 4 3 1 2 - 2 3 4 1.	14
			N_{23} (*Shared* = N_{13})	1 2 3 4 - 4 2 1 3 - 2 4 1 3	15
S_3	3 4 2 1 - 3 2 1 4 - 1 2 4 3	18	N_{31}	3 1 2 4 - 2 3 1 4 - 3 2 4 1	14
			N_{32}	1 3 2 4 - 3 2 4 1 - 4 3 1 2	14
			N_{33} (*Shared* = N_{12})	2 3 1 4 - 4 2 3 1 - 1 3 4 2	15

S_1 adalah pemimpin, S_3 berada di sisi kanan kawanan, S_2 berada di sisi kiri kawanan. Pada Tabel 3.6, *makespan* dari solusi tetangga yang dihasilkan diurutkan dari yang terbaik hingga yang terburuk. Kemudian, solusi tetangga pertama (terbaik) digunakan untuk meningkatkan solusi sendiri. Setelah itu, solusi tetangga kedua dibagikan dengan burung berikutnya di sisi kiri kawanan. Solusi tetangga ketiga dibagikan dengan sisi kanan kawanan. Jika $f(S_i) > f(N_{ij})$ maka $S_i = N_{ij}$, jika lainnya, S_i tidak diperbarui. N_{ij} disimbolkan sebagai *neighbor* (solusi tetangga) pada S_i dengan tetangga ke-*j*. Dengan demikian, *makespan* baru dari setiap burung, yaitu $S_1 = 15$, $S_2 = 13$, dan $S_3 = 14$.

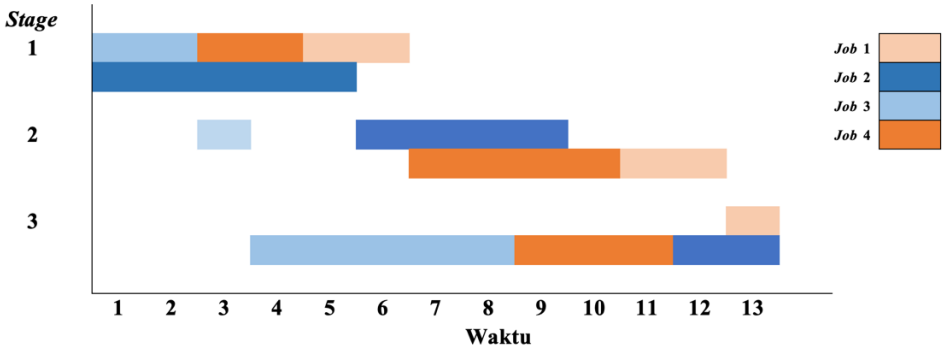
7. Dalam contoh kasus, terdapat burung yang lebih baik daripada pemimpin sebelumnya, yaitu S_2 yang setelah dievaluasi mempunyai *makespan* sebesar 13

satuan waktu maka pada iterasi pertama dilakukan pembaruan pemimpin dengan memindahkan pemimpin ke ujung kaki "V" sehingga terbentuk seperti pada Gambar 3.6.



Gambar 3.6 Formasi Baru Setelah 1 Iterasi

8. Karena dalam contoh kasus telah ditentukan maksimum iterasi 1, maka proses iterasi dihentikan. Sehingga diperoleh penjadwalan *job* terbaik, yaitu 3 2 1 4 - 1 3 2 4 - 3 4 2 1 dengan nilai *makespan*, yaitu 13 satuan waktu. Solusi ini dapat disajikan dalam bentuk diagram Gantt pada Gambar 3.7, di mana setiap *job* digambarkan sebagai sebuah blok yang panjangnya sesuai dengan waktu operasinya.



Gambar 3.7 Diagram Gantt untuk Contoh Kasus