Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) p-ISSN: 2355-7699
Vol. 3, No. 4, Desember 2016, hlm. 259-264 e-ISSN: 2528-6579

OPTIMASI PENJADWALAN PENGERJAAN *SOFTWARE* PADA *SOFTWARE HOUSE* DENGAN *FLOW-SHOP PROBLEM* MENGGUNAKAN *ARTIFICIAL BEE COLONY*

Muhammad Fhadli[1], Daneswara Jauhari[2], Dhimas Anjar Prabowo[3], Anang Hanafi[4], Aryeswara Sunaryo[5], Imam Cholissodin[6]

[1,2,3,4,5,6]Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]muhammadfhadli20@gmail.com, [2]daneswarajauhari@gmail.com, [3]dhimasanjar@gmail.com, [4]ananghanafi13@gmail.com, [5]ary3swara@gmail.com, [6]imamcs@ub.ac.id

(Naskah masuk: 1 Oktober 2016, diterima untuk diterbitkan: 26 Desember 2016)

Abstrak

Penelitian ini mengusulkan sebuah implementasi terkait optimasi penjadwalan pengerjaan *software* pada *software house* dengan *Flow-Shop Problem* (FSP) menggunakan algoritma *Artificial Bee Colony* (ABC). Dimana dalam FSP dibutuhkan suatu solusi untuk menyelesaikan suatu *job*/*task* dengan meminimalkan total *cost* yang dikeluarkan. Terdapat *constraint* yang perlu diperhatikan dalam objek permasalahan penelitian ini, yaitu lama waktu penyelesaian keseluruhan projek *software* yang tidak pasti. Dalam penelitian ini akan disusun sebuah representasi solusi yaitu berupa urutan pengerjaan projek dengan total waktu pengerjaan yang minimum. Pengujian akan dilakukan dengan tiga kali percobaan untuk setiap kondisi uji coba, yaitu uji coba batas parameter iterasi dan uji coba batas parameter limit. Dari hasil pengujian didapatkan bahwa penggunaan algoritma yang dibahas dalam penelitian ini bisa mengurangi waktu pengerjaan jika jumlah iterasi dan jumlah *colony* diperbesar.

Kata kunci: *optimasi, flow-shop problem, artificial bee colony, swarm intelligence, meta-heuristik.*

Abstract

This research proposed an implementation related to software execution scheduling process at a software house with Flow-Shop Problem (FSP) using Artificial Bee Colony (ABC) algorithm. Which in FSP required a solution to complete some job/task along with its overall cost at a minimum. There is a constraint that should be kept to note in this research, that is the uncertainty completion time of its jobs. In this research, we will present a solution that is a sequence order of project execution with its overall completion time at a minimum. An experiment will be performed with 3 attempts on each experiment conditions, that is an experiment of iteration parameter and experiment of limit parameter. From this experiment, we concluded that the use of this algorithm explained in this paper can reduce project execution time if we increase the value of total iteration and total colony.

Keywords: *optimization, flow-shop problem, artificial bee colony, swarm intelligence, meta-heuristic.*

1. PENDAHULUAN

Scheduling atau penjadwalan merupakan pekerjaan untuk mengalokasikan sumber daya yang terbatas dari suatu pekerjaan untuk mengefisiensikan waktu pengerjaan (Garrido dkk., 1994). *Job Shop Problem* merupakan suatu permasalahan dimana dibutuhkan suatu solusi untuk menyelesaikan suatu *job*/*task* dengan meminimalkan total *cost* yang dikeluarkan. Pada penelitian ini, terdapat *constraint* yang perlu diperhatikan yaitu ketidakpastian waktu proses penyelesaian suatu *job*.

Ketidakpastian waktu proses penyelesaian tentunya berhubungan erat dengan bidang pekerjaan tertentu seperti *software house*, dsb. Untuk *constraint* pertama berupa masalah ketidakpastian dalam waktu pemrosesan, dapat diselesaikan menggunakan algorima kombinatorial *Artificial Bee Colony*.

Penulis menemukan salah satu penelitian yang berhubungan dengan konsep dasar dari penjadwalan. Penelitian ini mendiskusikan permasalahan *Constraint Satisfaction Problem* (CSP) yang didasari oleh konsep *slack* untuk nilai dan variabel tertentu yang harus dilakukan oleh suatu operasi (Garrido dkk., 1994). Penelitian ini menyimpulkan bahwa metode heuristik dapat menyelesaikan beberapa jenis permasalahan yang sulit diselesaikan dengan menggunakan CSP tradisional.

Penelitian sebelumnya yang terkait dengan penjadwalan juga pernah dilakukan oleh Bruker dan Schile (Gao dkk., 2016) yang merupakan penggagas dalam penelitian tentang *Flexible Job Shop Problem* (FJSP) dengan mengajukan penelitian berupa algoritma *polynomial* untuk 2 *jobs* dan penggunaan mesin identik pada sebuah FJSP.

Penelitian lain dengan permasalahan FJSP dengan *uncertaity processing time* dan *new job insertion* pernah dilakukan oleh Gao, dkk. (2016) dengan menggunakan *Artificial Bee Colony* (ABC). Penelitian ini menggambarkan dan membandingkan

beberapa metode solusi terbaru dan membahas strategi pengembangan. Tujuannya adalah untuk meminimalkan nilai dari *fuzzy completion time.* Pada penelitian yang menjadi rujukan penulis, penelitian tersebut menggunakan *Two-stage Artificial Bee Colony* (TABC) dengan beberapa pengembangan dan hasilnya menunjukkan bahwa metode tersebut mampu bersaing dengan beberapa metode yang sudah ada sebelumnya.

Dari latar belakang tersebut, kami mencoba menerapkan prinsip-prinsip tersebut dengan merubah domain permasalahan menjadi penjadwalan pengerjaan *software.* Penelitan ini penulis beri judul Optimasi Penjadwalan Pengerjaan *Software* pada *Software House* dengan *Flow-Shop Problem* Menggunakan *Artificial Bee Colony*. Penelitian ini menggunakan data pegerjaakn *project* yang statis sehingga lebih efisien jika tidak menggunakan *two-stage artificial bee colony*, diharapkan penelitian yang penulis lakukan ini bisa mejadi rujukan pada penelitian-penelitian lain dengan topik yang berkaitan.

2. DASAR TEORI

2.1 Data Penelitian Projek

Penelitian ini menggunakan *dataset* yang didapat dari perusahaan *software house* GUMCODE, Malang, Jawa Timur. Data yang penulis peroleh berupa 10 daftar projek berserta *timeline* pengerjaan setiap projek. Data ini penulis konversikan kedalam satuan tabel dan bisa dilihat pada Tabel 2.1 berikut. Tabel ini menunjukkan lama pengerjaan setiap tahap pada suatu projek dalam satuan minggu.

Tabel 2.1 *Dataset* pengerjaan projek GUMCODE

Projek ke-*i*	Pengerjaan			
	Tahap 1	Tahap 2	Tahap 3	Tahap 4
1	2	2	1	2
2	1	3	3	4
3	6	9	2	3
4	1	4	3	2
5	4	4	1	2
6	3	1	0	0
7	1	2	7	3
8	1	3	4	4
9	1	2	8	1
10	1	1	5	2

2.2 Estimasi Pengerjaan *Software*

Estimasi Pengerjaan *Software* merupakan topik yang sulit. Steve McConnell menyebutnya *Black Art*, sehingga dia mengarang buku yang sangat bagus tentang topik ini, judulnya *Software Estimation, Demystifying the Black Art*. Menurut Steve, dalam membuat estimasi, ada 3 metode yang dilakukan, yaitu *count, compute, judge* (McConnell, 2006).

Karakteristik perangkat lunak tentunya memberikan kendala yang tidak sedikit terhadap estimasi yang akan dilakukan. Kerumitan dari perangkat lunak tersebut dan hal lain yang tidak kasat mata, serta sumber daya manusia pengembang perangkat lunak tidak bisa diperkirakan secara mutlak secara mekanistik seperti mesin.

2.3 *Flow-Shop Problem*

Dalam *flow-shop scheduling problem*, terdapat sejumlah *n job* dan *m* mesin, dimana setiap *job* harus diproses pada setiap mesin mulai dari mesin 1 sampai mesin *m* secara berurutan. Dengan kondisi seperti ini, tentunya akan muncul *buffer* dengan kapasitas yang tidak terbatas diantara mesin (Khorasanian, 2017). Pada permasalahan FSP ini, dapat diartikan dengan bagaimana membuat urutan pekerjaan, sehingga waktu yang dibutuhkan untuk menyelesaikan semua pekerjaan dapat diminimalkan.

2.4 Algoritma *Artificial Bee Colony*

2.4.1 Inisialisasi Parameter dan Populasi

Parameter yang harus diinisialisasi yaitu:

- Data Kasus : merupakan data pekerjaan apa saja yang harus dilakukan.
- *Colony Size* : jumlah *Employee Bee* ditambah dengan *Onlooker Bee* (Cholissodin, 2016).
- Maksimum Iterasi : merupakan banyaknya iterasi yang dilakukan.
- *Limit* : batas jumlah populasi yang kualitasnya tidak meningkat pada suatu iterasi.
- *Number of sequence* (NSE) : merupakan batasan banyak *sequence* dalam *neighborhood operator*.

Untuk melakukan inisialisasi populasi dapat dengan cara merandom urutan *project* yang harus dikerjakan, kemudian menghitung nilai *makespan* dan nilai *fitness*. Nilai *makespan* dapat dihitung dengan menggambarkan *gantt-chart*.

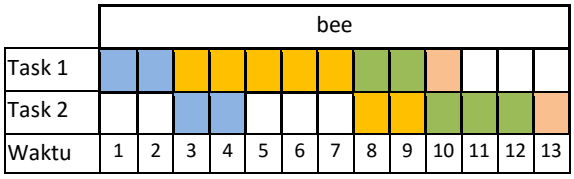
2.4.2 *Gantt Chart*

Perhitungan nilai *makespan* dilakukan menggunakan *gantt-chart.* Berikut ini penulis contohkan perhitungan *makespan* menggunakan *gantt-chart* dengan projek dan *task* yang ditunjukkan pada Tabel 2.2 dibawah ini, dengan urutan pengerjaan 2, 3, 1, 4:

Tabel 2.2 Projek dan *task*

Project	Task	
	1	2
1	2	3
2	2	2
3	5	2
4	1	1

Tabel di atas bisa kita gambarkan seperti *gantt-chart* pada Gambar 2.1 yang menunjukkan *makespan* dari *bee* tersebut adalah 13.



Gamber 2.1 *Gantt chart*

Berdasarkan contoh kasus, *project* yang pertama kali dikerjakan adalah *project* nomor 2 dimana pada *project* ini *task* 1 dan *task* 2 diselesaikan dalam 2 pekan dan *task* yang berbeda dari *project* yang sama tidak boleh dikerjakan dalam pekan yang sama sehingga pada kolom pekan pertama sampai pekan keempat di*block* dengan warna biru muda.

Hal yang sama dilakukan juga untuk *project* 3. Pada *task* 1, *project* 3 membutuhkan waktu 5 pekan untuk pengerjaannya. Oleh karena itu, *task* 2 dari *project* 3 baru dapat dijalankan pada pekan ke-8. Begitu seterusnya sampai *project* yang terakhir.

2.4.3 Rumus *Fitness*

Rumus *fitness* merupakan suatu rumus yang digunakan sebagai tolak ukur baik atau tidaknya urutan pengerjaan projek yang sedang diteliti, rumus *fitness* dapat dilihat pada persamaan 2.1 sebagai berikut:

$$fitness = {}^{1}/_{C} \quad (2.\ 1)$$

C = *cost* atau dimana dalam penelitian ini *cost* yang digunakan adalah total *makespan* dari *gantt-chart*.

2.4.4 Fase *Employeed Bee*

Pada fase ini dilakukan *update* populasi untuk setiap *Employeed Bee* dengan menggunakan *neighborhood operator*, yang terdiri dari *swap operator* (SO) dan *swap sequence* (SS). Kemudian hitung *fitness* nya, jika solusi yang baru lebih baik dari pada yang lama maka gantikan solusi lama dengan solusi baru, jika tidak tambahkan nilai trial dengan 1. Langkah-langkahnya sebagai berikut:

Swap Operator dapat dihitung dengan cara melakukan pergantian posisi antara 2 operasi. Solusi baru dapat dihitung dengan rumus (Cholissodin, 2016):

$$x_i = x_i + SO \quad (2.\ 2)$$

Swap Squences dapat dihitung dengan cara melakukan perhitungan SO sejumlah *NSE*, perhitungan SS dilakukan pada semua *Employeed Bee*. Solusi didapatkan dengan mencari nilai *fitness* terbaik pada setiap perhitungan SS, solusi baru dapat dihitung dengan rumus:

$$x_i = x_i + SS \quad (2.\ 3)$$

$$x_i = x_i + (SO_1, SO_2, SO_3, \dots, SO_n)$$

n = banyaknya SS, kemudian hitung nilai probabilitas setiap *Employeed Bee* dengan rumus (Cholissodin, 2016):

$$Prob_i = \frac{fitness(x_i)}{\sum_{k=1}^{S} fitness(x_k)} \quad (2.\ 4)$$

2.4.5 Fase *Onlooker Bee*

Untuk setiap *Onlooker Bee* didapatkan dengan cara memilih solusi yang ada pada *Employeed Bee*, dengan teknik seleksi *roulette wheel*.

Untuk menentukan solusi yang baru, dapat dilakukan dengan menggunakan *neighborhood operator*, yang terdiri dari *insert operator* (IO) dan *insert sequence* (IS). Kemudian hitung *fitness*-nya, jika solusi yang baru lebih baik dari pada yang lama maka gantikan solusi lama dengan solusi baru, jika tidak tambahkan nilai *trial* dengan 1. Langkah-langkahnya sebagai berikut:

Insert Operator dapat dihitung dengan cara melakukan *insert*, misalnya IO(1,3) artinya melakukan *insert* operasi ke 3 kedepan operasi 1. Solusi baru dapat dihitung dengan rumus (Cholissodin, 2016):

$$xi = xi + IO \quad (2.\ 5)$$

Insert Squences dapat dihitung dengan cara melakukan perhitungan IO sejumlah NSE, perhitungan IS dilakukan pada semua *Onlooker Bee*. Solusi didapatkan dengan mencari nilai *fitness* terbaik pada setiap perhitungan IS, Solusi baru dapat dihitung dengan rumus:

$$x_i = x_i + IS \quad (2.\ 6)$$

$$x_i = x_i + (IO_1, IO_2, IO_3, \dots, IO_n)$$

n = banyaknya IS

2.4.6 Fase *Scout Bee*

Setelah melalui dua fase sebelumnya, yaitu *Employeed Bee* dan *Onlooker Bee*, maka langkah selanjutnya yaitu perhitungan kualitas dari masing-masing *Employeed Bee*. Jumlah *Scout Bee* dalam fase

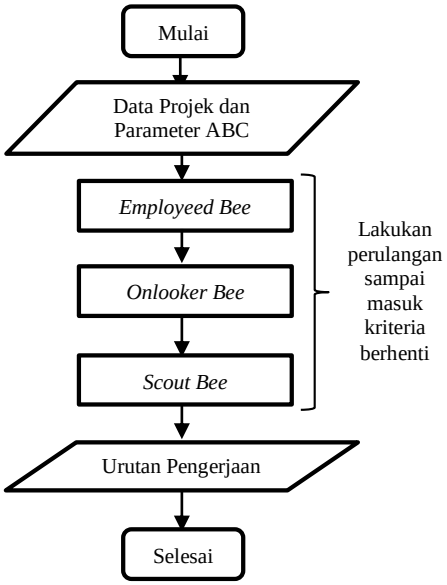
ini bersifat dinamis, tergantung pada jumlah *employeed* yang telah melebihi *limit*.

Apabila *limit* dari *Bee* yang melakukan *Improvement Solution* melebihi maksimum *limit* yang ditetapkan, maka solusi pada *Bee* tersebut dihilangkan dan digantikan dengan solusi baru yang dibuat secara *random*, kemudian memperbarui *fitness* yang dihasilkan, dan menyetel ulang *limit* menjadi 0.

3. PERANCANGAN DAN IMPLEMENTASI

3.1 Perancangan Alur Proses Algoritma

Perancangan alur proses algoritma ini kami adaptasi dari penjelasan yang telah dijelaskan pada bagian dasar teori, dimana rancangannya dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Alur Proses Optimasi Penjadwalan

Dimana representasi solusinya adalah indeks urutan pengerjaan projek, agar lebih jelasnya dapat dilihat contoh representasi solusi individu pada Gambar 3.2.

	<----------*representasi solusi*---------->			
P1 =	[Projek 1,	Projek 2,	…,	Projek *i*]

Gambar 3.2 Contoh representasi solusi individu

3.2 Perancangan Uji Coba

Perancangan uji coba ini dilakukan untuk mengevaluasi dan menganalisis hasil yang dihasilkan, dimana akan dilakukan 2 proses uji coba yaitu sebagai berikut:

3.2.1 Uji Coba Parameter Iterasi

Pada uji coba ini, penulis mencari nilai *fitness* yang optimal dengan melakukan perubahan sebanyak 5 kali terhadap jumlah iterasi. Percobaan ini dilakukan untuk melihat pengaruh jumlah iterasi terhadap peningkatan nilai *fitness.*

3.2.2 Uji Coba Parameter Limit

Pada uji coba ini, penulis mencari nilai *fitness* yang optimal dengan melakukan perubahan sebanyak 5 kali terhadap nilai limit. Percobaan ini dilakukan untuk melihat pengaruh nilai *limit* terhadap peningkatan nilai *fitness.*

3.3 Implementasi

Implementasi pada penelitian ini dilakukan dengan mengikuti dasar teori serta perancangan yang telah dijelaskan pada poin 2 dan 3.

Tabel 3.3 Prosedur pengerjaan FSP-ABC

Tahap 1: Membuat inisialisasi *bee* untuk tahap *employeed bee* secara acak sebanyak populasi

Tahap 2: Lakukan evaluasi *fitness* pada setiap *bee* dalam populasi

Tahap 3: Lakukan *improvement* dengan menggunakan *swap operator* (SO) pada *bee* inisialisasi, kemudian evaluasi *fitness*-nya kembali

Tahap 4: Lakukan *improvement* dengan menggunakan *swap sequence* (SS) pada *bee* hasil SO, kemudian evaluasi *fitness*-nya kembali

Tahap 5: Lakukan *selection* dengan menggunakan *roulette wheel selection* (RWS) pada *bee* hasil SS hingga terbentuk *bee* baru hasil seleksi sebanyak populasi sebagai *bee* untuk tahap *onlooker bee*

Tahap 6: Lakukan *improvement* dengan menggunakan *insertion operator* (IO) pada *bee* hasil RWS, kemudian evaluasi *fitness*-nya kembali

Tahap 7: Lakukan *improvement* dengan menggunakan *insertion sequence* (IS) pada *bee* hasil IO, kemudian evaluasi *fitness*-nya kembali

Tahap 8: Ambil individu dengan nilai *fitness* terbaik pada *bee* hasil IS sebagai *global best*/solusi terbaik

Tahap 9: Bandingkan nilai *fitness* antara *bee* inisialisasi dengan *bee* hasil IS, dengan kondisi:

- Jika maksimum nilai *trial bee* melebihi *limit* dan *bee* tidak mengalami perbaikan maka *reset* nilai *trial* menjadi 0, dan acak kembali urutan pengerjaan projeknya secara acak seperti pada tahap inisialisasi
- Jika maksimum nilai *trial bee* melebihi *limit* dan *bee* mengalami perbaikan maka *reset* nilai *trial* menjadi 0, dan biarkan urutan pengerjaan projek pada *bee* tersebut seperti pada hasil IS
- Jika maksimum nilai *trial bee* belum melebihi *limit* maka nilai *trial* tidak perlu di-*reset* dan biarkan urutan pengerjaan

projek pada *bee* tersebut seperti pada hasil IS

Kriteria berhenti dari sistem ini akan dilakukan sebanyak maksimum iterasi. Iterasi akan dilakukan sampai kriteria berhenti terpenuhi, dan selama belum terpenuhi, maka akan mengulang langkah Tahap 3.

4. PENGUJIAN DAN ANALISIS

4.1 Pengujian Parameter Iterasi

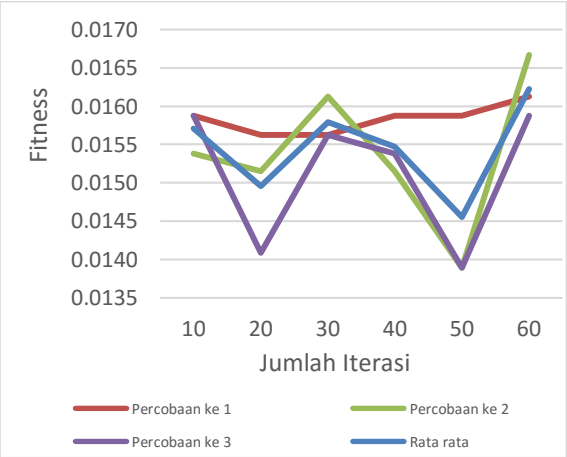
Pengujian parameter iterasi ini menggunakan jumlah *colony* sebesar 3 dan nilai *limit* diambil dari nilai terkecil pada pengujian parameter *limit*, yaitu 5. Berdasarkan hasil pengujian pada Tabel 4.1, nilai rata-rata *fitness* terkecil didapat pada iterasi 50.

Sementara itu, rata-rata nilai *fitness* terbesar bisa ditemukan pada saat iterasi 60. Sehingga jumlah iterasi bisa ditetapkan 60 agar mendapatkan nilai *fitness* maksimal.

Tabel 4.1 Pengujian Parameter Iterasi

Uji Coba Batas Parameter Iterasi				
Iterasi	**Nilai Fitness Percobaan ke-*i***			**Rata - Rata *Fitness***
	1	**2**	**3**	
10	0.0159	0.0154	**0.0159**	0.0157
20	0.0156	0.0152	0.0141	0.0150
30	0.0156	0.0161	0.0156	0.0158
40	0.0159	0.0152	0.0154	0.0155
50	0.0159	0.0139	0.0139	0.0146
60	**0.0161**	**0.0167**	**0.0159**	**0.0162**

Berikut ini adalah grafik yang menggambarkan tabel di atas.



Gambar 4.1 Pengujian Parameter Iterasi

4.2 Pengujian Parameter Limit

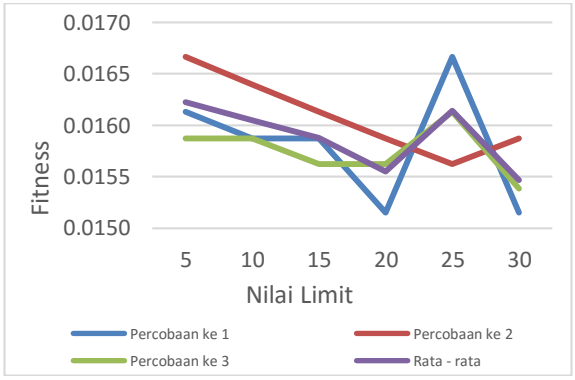
Pengujian parameter *limit* ini menggunakan jumlah *colony* sebesar 3 dan nilai iterasi diambil dari iterasi dengan *fitness* terbesar ada pengujian paramter iterasi, yaitu 60. Berdasarkan hasil pengujian pada Tabel 4.2, nilai rata-rata *fitness* terkecil didapat pada percobaan dengan limit 30.

Sementara itu, rata-rata nilai *fitness* terbesar bisa ditemukan pada saat nilai *limit* 5. Sehingga nilai iterasi bisa ditetapkan 5 agar mendapatkan nilai *fitness* maksimal.

Tabel 4.2 Pengujian Parameter *Limit*

Uji Coba Batas Parameter limit				
limit	**Nilai Fitness Percobaan ke-*i***			**Rata - Rata *Fitness***
	1	**2**	**3**	
5	0.0161	**0.0167**	0.0159	**0.0162**
10	0.0159	0.0164	0.0159	0.0160
15	0.0159	0.0161	0.0156	0.0159
20	0.0152	0.0159	0.0156	0.0155
25	**0.0167**	0.0156	**0.0161**	0.0161
30	0.0152	0.0159	0.0154	0.0155

Berikut ini adalah diagram yang menggambarkan tabel di atas.



Gambar 4.2 Pengujian Parameter *Limit*

Oleh karena itu, untuk mendapatkan nilai *fitness* yang maksimal, kita bisa mengecilkan nilai *limit* dan memperbesar nilai iterasi.

5. KESIMPULAN DAN SARAN

Artificial bee colony dengan *flow shop problem* bisa digunakan untuk menyelesaikan permasalahan penjadwalan pengerjaan *project* pada *software house* GUMCODE. Penjadwalan tersebut diuji dengan asumsi terdapat 4 mesin atau 4 tahap dalam pengerjaan setiap projek.

Pengujian ini menyimpulkan bahwa parameter yang optimal agar dapat menghasilkan nilai *fitness* terbaik adalah parameter dengan nilai iterasi 60 dan

nilai *limit* 5. Dengan kata lain, semakin besar nilai iterasi dan semakin kecil nilai *limit*, maka nilai *fitness* akan semakin baik.

Penelitian ini dapat dikembangkan dengan *job shop scheduling* yang lain ataupun bisa ditambahkan dengan pengunaan *fuzzy* seperti penelitian yang dilakukan oleh (Gao dkk., 2016).

6. DAFTAR PUSTAKA

CHOLISSODIN, I. (2016). Modul Swarm Intelligence – Semester Ganjil 2016-2017.

GAO, KAI ZHOU. dkk., 2016. Artificial bee colony algorithm for scheduling and rescheduling fuzzy flexible job shop problem with new job insertion. *Knowledge-Based Systems 109 (2016) 1-16.*

KARTHIKEYAN, A., MANIKANDAN, K. & SOMASUNDARAM, P., 2016. Economic Dispatch of Microgrid with Smart Energy Storage Systems using Particle Swarm Optimization. 2016 *International Conference on Computation of Power, Energy Information and Communication (ICCPEIC)*.

KHORASANIAN, D. & MOSLEHI, G., 2017. Two-machine flow shop scheduling problem with blocking, multi-task flexibility of the first machine, and preemption. *Computers and Operation Research*, 79(August 2016), pp.94–108. Available at: http://dx.doi.org/10.1016/j.cor.2016.09.023.

MCCONNELL, S., *Software Estimation*, Demystifying the Black Art. Washington, 2006.