ALGORITMA *DIFFERENTIAL EVOLUTION* UNTUK PENJADWALAN *FLOW SHOP* BANYAK MESIN DENGAN MULTI OBYEKTIF

STEFANUS EKO WIRATNO[1], RUDI NURDIANSYAH[2], DAN **BUDI SANTOSA[3]**
Teknik Industri, Fakultas Teknologi Industri, Institut Teknologi Sepuluh Nopember, Kampus ITS, Sukolilo, Surabaya, 60111

E-mail: eko_w@ie.its.ac.id

ABSTRAK

Penelitian ini mengembangkan algoritma Differential Evolution (DE) untuk menyelesaikan permasalahan penjadwalan flow shop m-mesin dengan mempertimbangkan dua obyektif yaitu makespan dan total flow time. Pengembangan algoritma DE dilakukan dengan menambahkan adaptive parameters pada tiap generasi, menggunakan strategi local search untuk meningkatkan kualitas solusi yang dihasilkan dan memodifikasi crossover untuk mengurangi waktu komputasi. Hasil penelitian ini menunjukkan bahwa algoritma DE yang diusulkan memiliki kinerja yang lebih baik dibandingkan dengan algoritma DE murni, algoritma Genetika (GA), dan pada kasus tertentu juga memiliki kinerja yang lebih baik dibandingkan algoritma Multi-Objective Ant Colony System (MOCSA).

Kata kunci: *flow shop scheduling, multi-objective, makespan, total flow time, differential evolution*

ABSTRACT

This research focuses on the development of Differential Evolution(DE) algorithmto solve m-machine flow shop scheduling problems with respect to both makespan and total flow time. Development of DE algorithm is done by modifying the adaptive parameter determination procedure in order to change the value of adaptive parameters in each generation, adding local search strategy to the algorithm in order to improve the quality of the resulting solutions, as ewell as modifying the crossover in order to reduce computation time. The result indicates that the proposed DE algorithm has proven to be better than the original DE algorithm, Genetic Algorithm (GA), and for certain cases it also out performs Multi-Objective Ant Colony System Algorithm (MOCSA).

Key words: *flow shop scheduling, multi-objective, makespan, total flow time, differential evolution*

PENDAHULUAN

Permasalahan penjadwalan *flow shop* biasanya ditetapkan memiliki obyektif tunggal, walaupun pada kenyataannya diinginkan untuk dapat mencapai beberapa obyektif seperti minimasi *makespan* dan minimasi total *flow time*. Minimasi *makespan* mengarah pada utilisasi dalam menjalankan produksi, sedangkan minimasi *total flow time* menghasilkan konsumsi yang stabil terhadap sumber daya, perputaran *job* yang cepat, dan meminimalkan *work in process inventory*. Total *flow time* merupakan ukuran kinerja sangat penting dalam meminimalkan total ongkos penjadwalan(Yagmahan dan Yenisey, 2010).

Penjadwalan *flow shop* menjadi permasalahan optimasi kombinatorial seiring dengan bertambahnya jumlah *job* dan jumlah mesin (Taillard, 1993). Permasalahan optimasi kombinatorial merupakan NP-hard dan pendekatan yang lebih menjadi pilihan dari permasalahan ini adalah teknik solusi yang mendekati optimal (Yagmahan dan Yenisey, 2008). Beberapa tahun terakhir, pendekatan metaheuristik seperti *Simulated Annealing*, *Tabu Search, Ant Colony Optimization, Genetic Algorithm, Particle Swarm Optimization, Differential Evolution* dan *Artificial Immune Systems* banyak digunakan untuk memecahkan permasalahan optimasi kombinatorial karena terbukti memiliki kinerja komputasi yang baik (Yagmahan dan Yenisey, 2008).

Salah satu algoritma yang mempunyai reputasi sebagai metoda optimasi global optima yang efektif adalah *Differential Evolution* (DE). Keunggulan dari DE adalah konsep yang sederhana, implementasi yang mudah dan cepat konvergen, namun kinerja DE sangat tergantung dari parameternya (Qian *et al.,* 2008). Tvrdik (2006) menyatakan bahwa efisiensi pencarian dari DE sangat sensitif terhadap penentuan nilai parameter F yang berfungsi untuk mengendalikan tingkat pertumbuhan populasi dan nilai parameter Cr yang berfungsi untuk

mengendalikan fraksi nilai variabel yang disalin dari vektor mutan.

Penelitian ini merancang suatu algoritma berbasis DE yang secara khusus digunakan untuk menyelesaikan permasalahan penjadwalan *flow shop* banyak mesin dengan obyektif untuk meminimasi *makespan* dan *total flow time*. Algoritma yang dirancang terdiri dari algoritma DE murni dan algoritma DE_plus yang merupakan pengembangan dari algoritma DE murni dengan cara melakukan tiga modifikasi. Pertama, memodifikasi pengendalian nilai parameter F dan Cr (disebut *adaptive parameters*) dengan cara menghitung nilai parameter F dan Cr pada tiap generasi dengan formula tertentu sehingga nilai parameter F dan Cr tiap generasi berubah-ubah. Kedua, merujuk pada hasil penelitian Pan *et al*., (2008) yang menyarankan untuk menambahkan prosedur *local search* pada algoritma DE untuk meningkatkan kualitas solusi yang dihasilkan. Ketiga, *crossover* yang pada mulanya merupakan langkah untuk menyilangkan populasi target dengan populasi mutan, dimodifikasi menjadi langkah untuk memilih populasi target yang akan dimutasi, sehingga dilakukan sebelum proses mutasi. Langkah ini bertujuan untuk mengurangi waktu komputasi. Hasil algoritma DE_plus akan dibandingkan dengan hasil yang diperoleh pada algoritma DE murni, *Genetic Algorithm* (GA), dan *Multi-Objective Ant Colony System Algorithm* (MOACSA).

MODEL PERMASALAHAN

Penjadwalan *flow shop* dicirikan oleh adanya aliran proses satu arah dan mesin disusun secara seri. Setiap *job* yang ada harus melalui semua mesin dengan urutan mesin yang sama di dalam prosesnya. Deskripsi model penjadwalan *flow shop* menurut Hejazi dan Saghafian (2005) adalah sebagai berikut: Seperangkat M mesin, $M = \{1,2,..,m\}$ yang digunakan untuk memproses sekelompok N *job*, $N = \{1,2,..,n\}$. Pada waktu yang sama, setiap mesin hanya dapat memproses satu *job* dan setiap *job* hanya diproses di satu mesin. Setiap *job* di tahap operasi i, hanya diproses sekali saja di salah satu mesin. Keseluruhan *job* dikerjakan dalam arah aliran operasi yang sama.

Notasi yang digunakan:

J : jumlah *job* (1,2,...,n)
M : jumlah mesin (1,2,...,m)
O : jumlah operasi (1,2,...,m)
J_i : *job* ke-i
π_i : *job* yang ditempatkan pada urutan ke-i
P_{ik} : waktu proses *job* ke-i pada mesin k
v_{ik} : *idle time* mesin k sebelum memproses *job* ke-i
w_{ik} : *waiting time job* ke-i setelah dikerjakan pada mesin k sebelum dikerjakan pada mesin k+1

Variabel keputusan:

$$x_{ij} = \begin{cases} 1, & \text{jika job } j \text{ ditempatkan pada urutan ke} - i \\ 0, & \text{jika yang lain} \end{cases}$$

$$Makespan = C_{\max} = C(\pi_n, m) \quad \text{..... (1)}$$

$$Total\ flow\ time = F = \sum_{i=1} C(\pi_i, m) \quad \text{..... (2)}$$

Sedangkan fungsi tujuan diformulasikan sebagai berikut:

$$w_1 \cdot C_{\max} + w_2 \cdot F \quad \text{..... (3)}$$

di mana:
w_1 = bobot *makespan*
w_2 = bobot *total flow time*

ALGORITMA YANG DIUSULKAN

Algoritma DE merupakan salah satu metode metaheuristik terbaru yang diperkenalkan oleh Storn dan Price (1997). Ide awal DE berasal dari algoritma *Genetic Annealing* di mana algoritma berdasarkan pembangkitan populasi dengan melakukan perturbasi (*perturbation*) pada suatu faktor mutan untuk membentuk populasi mutan.Operator *crossover* mengkombinasikan populasi mutan dengan populasi target untuk membangkitkan populasi percobaan. Selanjutnya operator seleksi membandingkan nilai fungsi *fitness* antara populasi percobaan dengan populasi target. Akhirnya individu yang terbaik akan menjadi anggota populasi generasi berikutnya. Proses ini diulang sampai tercapainya suatu konvergensi.

Penyelesaian permasalahan penjadwalan *flow shop* dengan menggunakan DE dapat dilakukan apabila parameter kontinyu pada DE diubah menjadi permutasi *job*. Tasgetiren *et al.,* (2004) mengubah nilai parameter kontinyu menjadi permutasi *job* dengan melakukan prosedur *smallest position value* (SPV), dan Qian *et al*. (2008) menggunakan prosedur *largest-order-value* (LOV).

Langkah-langkah algoritma DE_plus sebagai berikut.

Inisialiasi Populasi

Sebelum melakukan inisialisasi terhadap titik populasi maka perlu dilakukan penentuan batas atas (*ub*) dan batas bawah (*lb*). Untuk pembangkitan nilai awal generasi $g = 0$, variable ke-j dan *vector* ke-i bisa diwakili dengan notasi berikut:

$$x_{j,i,0} = lb_j + rand_j\,(0,1)(ub_j - lb_j) \quad \text{..... (4)}$$

Bilangan *random* dibangkitkan dengan fungsi *rand*(), di mana bilangan yang dihasilkan terletak antara (0, 1). Solusi awal diperoleh dengan cara mengurutkan populasi menggunakan prosedur SPV.

Crossover

Langkah *crossover* pada DE murni dilakukan dengan menyilangkan setiap vektor $x_{i,g}$, dengan vektor mutan $v_{i,g}$, untuk membentuk vektor hasil persilangan $u_{i,g}$.

$$u_{i,g} = u_{j,i,g} = \begin{cases} v_{j,i,g}, & \text{jika } j(rand_j(0,1) \le Cr, \text{ atau } j = j_{rand} \\ x_{j,i,g}, & \text{jika yang lain} \end{cases} \quad \text{... (5)}$$

Crossover pada DE_plus dilakukan sebelum mutasi untuk memilih populasi yang akan dimutasi. Konsep mutasi yaitu bila bilangan random lebih kecil atau sama dengan *Cr* maka populasi tersebut akan dimutasi, bila bilangan random lebih besar dari *Cr* maka populasi itu tidak dimutasi. Probabilitas *crossover*, $Cr \in (0,1)$ adalah nilai yang didefinisikan untuk mengendalikan fraksi nilai parameter yang disalin dari mutan. Probabilitas *crossover* untuk tiap generasi akan ditentukan dengan persamaan Mingyong dan Erbao (2010):

$$Cr = Cr_{\min} + G \cdot \frac{Cr_{\max} - Cr_{\min}}{MAXGEN} \quad \text{..... (6)}$$

Cr_{min} dan Cr_{max} adalah nilai terkecil dan terbesar dari probabilitas *crossover*, *G* adalah iterasi pada saat waktu *trunning time*, sedangkan *MAXGEN* adalah jumlah maksimum iterasi yang diujicobakan. Tujuan dari penentuan nilai *Cr* adalah meningkatkan keragaman vektor yang akan mengalami *crossover* dan menghindar dari *local optima*.

Mutasi

Setelah tahapan *crossover*, DE_plus melakukan memutasi dan kombinasi terhadap populasi yang telah dipilih pada langkah *crossover*. Mutasi dilakukan dengan cara menambahkan perbedaan dua vektor terhadap vektor ketiga secara acak. Formulasinya sebagai berikut:

$$v_{i,g} = x_{r0,g} + F(x_{r1,g} - x_{r2,g}) \quad \text{..... (7)}$$

Faktor skala $F \in (0, 1)$ adalah bilangan *real* positif yang mengendalikan tingkat pertumbuhan populasi. Pada langkah ini nilai parameter *F* tiap generasi akan berubah-ubah dengan menghitung nilai parameter pada tiap generasi dengan formula yang dikembangkan oleh Tvrdik (2006) sebagai berikut:

$$F = \begin{cases} \max\left(F_{\min}, 1 - \left|\frac{f_{\max}}{f_{\min}}\right|\right) & \text{jika } \left|\frac{f_{\max}}{f_{\min}}\right| < 1 \\ \max\left(F_{\min}, 1 - \left|\frac{f_{\min}}{f_{\max}}\right|\right) & \text{jika yang lain} \end{cases} \quad \text{..... (8)}$$

Di mana $f_{\min}$ adalah nilai fungsi minimum dari populasi dan f_{max} adalah nilai fungsi maksimum dari populasi. $F_{\min}$ merupakan input parameter yang memastikan $F \in [F_{\min},1]$. Formulasi ini mencerminkan pencarian yang lebih beragam pada tahap awal dan lebih intensif pada tahap berikutnya.

Selection

Jika *trial vector* $u_{i,g}$, mempunyai fungsi tujuan lebih kecil dari fungsi tujuan vektor targetnya yaitu $x_{i,g}$, maka $u_{i,g}$ akan menggantikan posisi $x_{i,g}$ dalam populasi pada generasi berikutnya. Jika sebaliknya, target akan tetap pada posisinya dalam populasi.

$$x_{i,g+1} = \begin{cases} u_{i,g} & \text{jika } f(u_{i,g}) \le f(x_{i,g}) \\ x_{i,g} & \text{jika sebaliknya} \end{cases} \quad \text{..... (9)}$$

Local Search

Hasil dari seleksi akan dikenai prosedur *insert-based local search* yang cenderung mengarahkan pencarian ke daerah solusi yang menjanjikan dalam waktu relatif singkat. Prosedur dari *insert-based local search* adalah sebagai berikut (Qian *et al.*, 2008):

Langkah 1 : Ubah nilai individual $x_i(t)$ menjadi permutasi *job* π_{i_0}
Langkah 2 : Pilih secara acak *u* dan *v*,
di mana $u \neq v$; $\pi_i = insert(\pi_{i\_0}, u, v)$.
Langkah 3 : *Set loop* = 1;
Do
Pilih secara acak *u* dan *v*, di mana $u \neq v$;
$\pi_{i\_1} = insert(\pi_i, u, v)$;
jika $f(\pi_{i\_1}) < f(\pi_i)$, maka $\pi_i = \pi_{i\_1}$;
$loop^{++}$;
while loop< (*n*x (*n*-1)).
Langkah 4 : Jika $f(\pi_i) < f(\pi_{i\_0})$, maka $\pi_{i\_0} = \pi_i$

Proses mutasi, *crossover*, seleksi dan *local search* akan diulang sampai kriteria pemberhentian dicapai.

Kriteria Pemberhentian

Kriteria pemberhentian yang digunakan adalah iterasi maksimal.

HASIL DAN PEMBAHASAN

Pengujian algoritma dilakukan dengan membuat kode program algoritma DE_plus pada s*oftware* Matlab. Kode dijalankan dengan menggunakan spesifikasi komputer Intel Core Duo 1,66 GHz, RAM 1 GB, serta menggunakan *software* Matlab seri 7.8. Data uji menggunakan data yang terdapat pada *OR-Library*.

Parameter algoritma DE_plus yang digunakan: jumlah populasi = 100 (20 *job*) dan 150 (50 *job*); $F_{min} = 0{,}5$; $Cr = 0{,}3\text{-}0{,}9$ dan jumlah iterasi sebanyak 1000 (20 *job*) dan 1500 (50 *job*). Tiap pengujian algoritma akan direplikasi 10 kali dan hasil terbaik akan dipilih. Bobot dari *makespan* dan *total flow time* ditetapkan sebesar 0,5. Persentase relatif dari multi obyektif *makespan* dan *total flow time* adalah sebagai berikut (Yagmahan dan Yenisey, 2010):

$$RE = w_1.\left(\frac{C_{max} - \min C_{max}}{\min C_{max}}\right) + w_2.\left(\frac{F - \min F}{\min F}\right) \quad \text{..... (10)}$$

Persentase relatif dari obyektif *makespan, total flow time* dan multi obyektif keduanya pada kasus 20 *job* (lihat Tabel 1, 2, dan 3) yang menunjukkan algoritma DE_plus memiliki kinerja yang lebih baik dibandingkan dengan algoritma DE murni, GA, dan MOACSA.

Tabel 1. Performa dari Algoritma untuk Obyektif *Makespan*

Kasus	n × m	GA	MOACSA	DE Murni	DE_plus
ta001	20 × 5	0,00	1,49	0,00	0,00
ta002	20 × 5	0,00	0,73	0,00	0,15
ta003	20 × 5	0,00	5,53	1,57	0,74
ta004	20 × 5	0,00	2,91	0,85	0,70
ta005	20 × 5	1,61	0,31	0,65	0,00
ta006	20 × 5	2,99	0,57	1,26	0,00
ta007	20 × 5	0,00	1,88	0,97	0,97
ta008	20 × 5	4,35	0,00	0,00	0,00
ta009	20 × 5	2,30	0,00	0,00	0,00
ta010	20 × 5	1,95	0,00	0,00	0,00
ta011	20 × 10	0,00	0,66	1,39	0,95
ta012	20 × 10	4,17	0,00	0,96	1,27
ta013	20 × 10	3,15	0,00	1,20	1,27
ta014	20 × 10	0,20	0,00	1,60	0,73
ta015	20 × 10	1,54	0,00	0,63	0,70
ta016	20 × 10	0,00	1,49	1,65	0,29
ta017	20 × 10	0,00	0,70	1,08	0,61
ta018	20 × 10	2,82	0,00	2,47	0,52
ta019	20 × 10	0,00	1,02	1,38	0,94
ta020	20 × 10	1,75	0,00	1,51	0,69
ta021	20 × 20	0,45	0,00	2,48	0,78
ta022	20 × 20	0,00	2,56	0,67	1,10
ta023	20 × 20	7,13	4,31	1,20	1,07
ta024	20 × 20	0,00	0,30	1,53	0,76
ta025	20 × 20	0,21	2,19	0,70	0,74
ta026	20 × 20	0,17	0,00	1,39	0,18
ta027	20 × 20	10,40	9,38	1,85	1,58
ta028	20 × 20	0,00	3,16	0,95	0,77
Rata-rata total		**1,659**	**1,489**	**1,088**	**0,642**

Lanjutan Tabel 1

Tabel 2. Performa dari Algoritma untuk Obyektif Total *Flow time*

Kasus	n × m	GA	MOACSA	DE Murni	DE_plus
ta001	20 × 5	0,00	0,96	-1,30	-0,96
ta002	20 × 5	1,40	0,00	-0,43	-1,50
ta003	20 × 5	1,17	0,00	-2,02	-1,74
ta004	20 × 5	0,53	0,00	-1,85	-1,49
ta005	20 × 5	0,00	0,38	-2,06	-2,06
ta006	20 × 5	2,32	0,24	-0,24	2,91
ta007	20 × 5	1,11	0,00	-1,06	-1,06
ta008	20 × 5	0,73	0,00	2,11	-1,60
ta009	20 × 5	1,87	0,00	-0,85	-2,66
ta010	20 × 5	1,55	0,33	1,11	0,31
ta011	20 × 10	0,80	0,00	0,11	-1,94
ta012	20 × 10	0,00	0,43	3,35	-1,34
ta013	20 × 10	0,00	0,30	-1,21	0,16
ta014	20 × 10	2,87	0,00	0,34	-0,91
ta015	20 × 10	0,00	0,41	2,84	2,88
ta016	20 × 10	0,00	0,44	0,86	-0,84
ta017	20 × 10	1,78	0,45	-1,84	-0,28
ta018	20 × 10	1,45	0,00	0,39	-1,93
ta019	20 × 10	0,48	0,00	2,08	0,47
ta020	20 × 10	1,36	0,60	2,38	0,07
ta021	20 × 20	0,96	0,00	-0,76	-0,13
ta022	20 × 20	0,31	0,00	-1,64	-0,86
ta023	20 × 20	5,51	4,01	3,20	-0,40
ta024	20 × 20	0,00	0,05	-1,29	-1,29
ta025	20 × 20	6,43	5,89	-1,43	-1,55
ta026	20 × 20	1,72	0,00	-2,06	-2,99
ta027	20 × 20	1,50	0,00	-1,02	-0,90
ta028	20 × 20	0,54	0,00	-0,97	0,55
Rata-rata total		**1,354**	**0,566**	**-0,158**	**-0,766**

Tabel 3. Performa dari Algoritma untuk Multi Obyektif

Kasus	n × m	GA	MOACSA	DE Murni	DE_plus
ta001	20 × 5	0,00	1,53	-0,48	-0,48
ta002	20 × 5	1,34	0,00	1,18	0,14
ta003	20 × 5	0,00	0,05	1,02	0,79
ta004	20 × 5	0,40	0,00	1,10	0,93
ta005	20 × 5	0,00	0,67	-0,71	-0,71
ta006	20 × 5	2,12	0,12	1,10	2,08
ta007	20 × 5	0,90	0,00	1,63	0,65
ta008	20 × 5	1,15	0,00	1,51	-0,59
ta009	20 × 5	1,50	0,00	0,05	-0,40
ta010	20 × 5	1,30	0,00	1,91	0,47
ta011	20 × 10	0,86	0,00	1,00	-0,27
ta012	20 × 10	0,00	0,19	2,21	1,11
ta013	20 × 10	0,16	0,00	0,00	0,71
ta014	20 × 10	2,40	0,00	0,97	-0,09
ta015	20 × 10	0,00	0,33	1,89	1,79
ta016	20 × 10	0,00	0,40	1,93	0,54
ta017	20 × 10	1,62	0,38	-0,24	0,20
ta018	20 × 10	1,37	0,00	1,59	-0,70
ta019	20 × 10	0,16	0,00	1,73	0,74
ta020	20 × 10	1,16	0,37	2,26	0,85
ta021	20 × 20	1,02	0,00	2,08	3,17
ta022	20 × 20	0,34	0,00	-0,48	0,28
ta023	20 × 20	5,67	4,27	2,42	0,51
ta024	20 × 20	0,00	0,35	0,76	0,81
ta025	20 × 20	6,47	5,69	-0,13	-0,12
ta026	20 × 20	1,67	0,00	0,09	-0,94
ta027	20 × 20	1,51	0,00	1,67	0,81
ta028	20 × 20	0,55	0,00	0,29	0,94
Rata-rata total		**1,266**	**0,564**	**1,001**	**0,486**

Kinerja pada kasus yang lebih besar, dilakukan dengan pengujian pada kasus 50 *job*. Kinerja masing-masing algoritma ditunjukkan pada Tabel 4, 5, dan 6. Kinerja algoritma DE_plus pada kasus 50 *job* lebih baik daripada GA dan algoritma DE murni, akan tetapi lebih jelek bila dibandingkan MOACSA. Kinerja MOACSA pada kasus 50 *job* dipengaruhi oleh inisialisasi yang menggunakan algoritma NEH (Nawaz *et al*., 1983), di mana algoritma NEH ini tangguh untuk penjadwalan *flow shop* (Nagano dan Moccellin, 2002). Sedangkan DE_plus menggunakan inisialisasi secara acak. Algortima NEH dilakukan dengan mengurutkan *job* mulai dari yang memiliki total waktu proses yang terlama sampai dengan yang paling singkat. Urutan awal ini akan disempurnakan dengan prosedur penyisipan untuk mencari urutan *job* yang terbaik secara iteratif. Penggunaan algoritma NEH sebagai tahap inisialisasi menyebabkan waktu komputasi yang lama. Rata-rata waktu komputasi MOACSA untuk kasus 50 *job* sebesar 8,44 jam, sedangkan DE_plus sebesar 4,51 jam (16262,1 detik), meskipun waktu komputasi juga dipengaruhi oleh *software* yang digunakan (MOACSA menggunakan VP; DE_plus dengan MATLAB) maupun spesifikasi komputer.

Tabel 4. **P**erforma dari Algoritma untuk Obyektif *Makespan*

Kasus	n × m	GA	MOACSA	DE murni	DE_plus
ta031-40	50 × 5	0,438	0,545	4,246	0,458
ta041-50	50 × 10	2,549	0,000	4,156	0,679
ta051-60	50 × 20	2,191	0,050	5,032	1,601
Rata-rata		**1,726**	**0,198**	**4,478**	**0,913**

Tabel 5. Performa dari Algoritma untuk Obyektif *Total Flow time*

Kasus	n × m	GA	MOACSA	DE murni	DE_plus
ta031-40	50 × 5	4,246	0,000	4,500	0,444
ta041-50	50 × 10	3,617	0,000	2,556	0,205
ta051-60	50 × 20	3,162	0,000	2,471	0,290
Rata-rata		**3,675**	**0,000**	**3,176**	**0,313**

Tabel 6. Performa dari Algoritma untuk Multi Obyektif

Kasus	n × m	GA	MOACSA	DE murni	DE_plus
ta031-40	50 × 5	4,028	0,000	4,903	0,976
ta041-50	50 × 10	3,551	0,000	3,748	1,144
ta051-60	50 × 20	3,128	0,000	4,111	1,504
Rata-rata		**3,569**	**0,000**	**4,254**	**1,208**

Penurunan kinerja algoritma DE_plus pada kasus yang lebih besar terjadi karena jumlah ruang pencarian yang semakin besar dan meningkat secara faktorial. Dengan peningkatan ukuran *job* dari 20 menjadi 50, terjadi peningkatan ruang pencarian dari 20! menjadi 50!. Kinerja algoritma DE_plus akan lebih baik apabila dilakukan penambahan jumlah iterasi, tetapi juga berimbas pada waktu komputasi yang semakin lama. Namun demikian pada kasus penambahan *job*, penambahan jumlah iterasi dari 1000 pada kasus 20 *job* menjadi 1500 pada kasus 50 *job* dan jumlah populasi dari 100 menjadi 150 ternyata masih belum mampu untuk mencegah penurunan kinerja algoritma DE_plus. Penambahan iterasi dan populasi tersebut memberi dampak peningkatan waktu komputasi secara drastis yaitu lebih dari lima kali lipat.

Pengujian juga dilakukan untuk melihat pengaruh jumlah populasi dan jumlah iterasi terhadap kinerja algoritma DE_plus. Skenario yang digunakan

yaitu jumlah populasi tetap dengan jumlah iterasi semakin besar, jumlah iterasi tetap dengan jumlah populasi semakin besar serta kombinasi antara jumlah populasi dan iterasi yang berubah-ubah. Hasil pengujian mengkonfirmasikan bahwa jumlah iterasi dan jumlah populasi berpengaruh terhadap performa algoritma DE_plus di mana semakin besar jumlah iterasi dengan jumlah populasi tetap atau semakin besar jumlah populasi dengan jumlah iterasi tetap akan menghasilkan kinerja yang semakin baik. Semakin besar jumlah populasi dan iterasinya, dapat menghasilkan kinerja yang lebih baik karena memiliki ruang pencarian yang semakin besar pula. Pada kombinasi tersebut, algoritma DE_plus dapat konsisten menghasilkan *makespan* yang selalu optimal dan *total flow time* yang selalu lebih kecil dibandinghasil referensi yang digunakan sehingga dapat menghasilkan kinerja multi obyektif di bawah 0.

SIMPULAN

Hasil penelitian ini menunjukkan bahwa algoritma DE_plus memiliki kinerja yang lebih baik dibandingkan dengan algoritma DE murni, GA dan MOACSA untuk obyektif *makespan*, *total flow time* dan multi obyektif keduanya pada kasus 20 *job*. Pada kasus 50 *job*, kinerja algoritma DE_plus masih lebih baik dibandingkan GA dan algoritma DE murni, namun tidak lebih baik dibandingkan kinerja MOACSA. Meskipun demikian, waktu komputasi algoritma DE_plus masih lebih baik dibanding waktu komputasi yang diperlukan oleh MOACSA. Penelitian lebih lanjut dapat dilakukan untuk mengembangkan algoritma DE_plus pada obyektif tunggal maupun multi obyektif yang lain seperti *mean flow time*, *total tardiness* maupun *maximum tardiness*. Penggunaan algoritma-algoritma tertentu untuk inisialisasi, seperti algoritma NEH pada MOACSA, merupakan agenda penelitian lebih lanjut untuk meningkatkan kinerja algoritma DE_plus.Selain itu juga masih terbuka untuk melakukan penelitian pada permasalahan penjadwalan sistem manufaktur yang lain seperti *job shop*, c*ellular manufacturing*, dan *flexible manufacturing*.

DAFTAR PUSTAKA

Hejazi, S.R. and Saghafian, S., 2005. Flowshop-Scheduling problems with Makespan Criterion: A Review, International Journal of Production Research, 43(14), 2895–2929.

Mingyong, L. and Erbao, C., 2010. An Improved Differential Evolution Algorithm for Vehicle Routing Problem with Simultaneous Pickups and Deliveries and Time Windows, Engineering Applications of Artificial Intelligence, 23, 188–195.

Nagano, M.S. and Moccellin, J.V., 2002. A High Quality Solution Constructive Heuristic for Flow Shop Sequencing, Journal of The Operational Research Society, 53(12), 1374–1379.

Nawaz, M., Enscore, E., and Ham, I., 1983. A Heuristic Algorithm for the m-machine, n-job Flow Shop Sequencing Problem, OMEGA, 11(1), 91–95.

Pan, Q.K., Tasgetiren, M.F., and Liang, Y.C., 2008. A Discrete Differential Evolution Algorithm for The Permutation Flowshop Scheduling Problem, Computers & Industrial Engineering, 55, 795–816.

Qian, B., Wang, L., Huang D., Wang, W.L. and Wang, X., 2008. A Hybrid Differential Evolution Method for Permutation Flow-Shop Scheduling, The International Journal of Advanced Manufacturing Technology, 38, 757–777.

Storn, R. and Price, K., 1997. Differential Evolution - A Simple and Efficient Heuristic for Global Optimization Over Continuous Space, Journal of Global Optimization, 11, 341–359.

Tasgetiren, M.F, et al., 2004. Differential Evolution for Permutation Flowshop Sequencing Problem with Makespan Criterion, Dept. of Management, Fatih University, Istambul-Turkey.

Taillard, E., 1993. Benchmarks for Basic Scheduling Problems, European Journal of Operational Research, 64(2), 278–285.

Tvrdik, J., 2006. Differential Evolution: Competitive Setting of Control Parameters, Proceedings of the International Multiconference on Computer Science and Information Technology, University of Ostrava, Ostrava, 207–213.

Yagmahan, B. and Yenisey, M.M., 2008. Ant Colony Optimization for Multi-Objective Flow Shop Scheduling Problem, Computers & Industrial Engineering, 54(3), 411–420.

Yagmahan, B. and Yenisey, M.M., 2010. A Multi-Objective Ant Colony System Algorithm for Flow Shop Scheduling Problem, Expert Systems with Applications, 37(2), 1361–1368.