ORIGINAL ARTICLE

OPTIMASI PENJADWALAN *TWO-STAGE ASSEMBLY FLOWSHOP* MENGGUNAKAN ALGORITMA GENETIKA YANG DIMODIFIKASI

Wayan Firdaus Mahmudy

Program Studi Ilmu Komputer, Program Teknologi Informasi dan Ilmu Komputer,
Universitas Brawijaya, Malang
wayanfm@ub.ac.id

Abstrak

Tulisan ini memaparkan penggunaan algoritma genetika untuk optimasi penjadwalan *two-stage assembly flowshop*. Seperti masalah kombinatorial lainnya maka susunan optimum untuk *two-stage assembly flowshop* hanya bisa diperoleh dengan mencoba semua kemungkinan solusi. Metode enumerasi lengkap seperti algoritma *branch-and-bound* bisa digunakan untuk proses pencarian ini tetapi tetap memerlukan waktu yang relatif lama untuk masalah berukuran besar. Makalah ini mengajukan minimasi *makespan* dalam *two-stage assembly flowshop* menggunakan algoritma genetika yang dimodifikasi (ModGA). Modifikasi dilakukan dengan menambahkan mekanisme untuk mencegah konvergensi dini. Serangkaian 1200 percobaan menunjukkan bahwa ModGA lebih unggul dibandingkan dengan algoritma genetika standar. ModGA menghasilkan solusi optimum lebih banyak dan deviasi nilai solusi yang rendah terhadap solusi optimum.

Kata kunci : *algoritma genetika, two-stage assembly flowshop, makespan, konvergensi dini*

1. Pendahuluan

Penjadwalan merupakan komponen penting yang sangat mempengaruhi produktifitas dalam industri manufaktur. Penjadwalan yang baik akan menghemat waktu dan biaya produksi. Penjadwalan didefininisikan sebagai masalah alokasi sejumlah pekerjaan (*jobs*) ke sejumlah sumber daya dalam rentang waktu tertentu. Sumber daya ini bisa berupa mesin, pekerja, alat, modal dan lain-lain. Sejumlah kendala harus dipenuhi untuk menghasilkan jadwal yang layak [1, 2].

Pada masalah flowshop terdapat *n* job yang harus melewati *m* mesin dalam urutan yang sama tetapi waktu proses yang mungkin berbeda [1]. Pada *two-stage assembly flowshop* terdapat pemrosesan tahap kedua yang hanya bisa dilakukan setelah *m* operasi pada sebuah job telah diselesaikan secara paralel pada *m* mesin pada tahap pertama [3]. Ilustrasi sederhana dari masalah ini adalah pada proses pembuatan sebuah personal computer (PC) yang bisa dianggap sebagai sebuah job. Setelah sejumlah komponen seperti CPU, harddisk, memory dan lain-lain selesai dibuat pada tahap pertama pada tempat/mesin yang berbeda maka komponen-komponen ini masuk ke *assembly-station* pada tahap kedua untuk dirakit sesuai spesifikasi yang dibutuhkan konsumen. Jika pada saat yang sama terdapat beberapa pesanan PC dengan spesifikasi yang berbeda maka masalah yang timbul adalah bagaimana menentukan urutan pembuatan semua pesanan PC supaya didapatkan waktu penyelesaian semua PC dalam waktu yang paling singkat.

Urutan job yang harus diselesaikan menentukan waktu selesainya seluruh job (*makespan*). Proses optimasi dilakukan untuk menentukan urutan operasi yang menghasilkan nilai *makespan* minimum. Seperti masalah kombinatorial lainnya maka susunan optimum hanya bisa diperoleh dengan mencoba semua kemungkinan. Algoritma *branch-and-bound* bisa digunakan untuk proses pencarian ini tetapi tetap memerlukan waktu yang relatif lama untuk masalah berukuran besar. Algoritma meta-heuristik seperti algoritma genetika menjanjikan hasil yang memuaskan dalam waktu yang relatif cepat [2].

Sejumlah literatur melaporkan solusi *two-stage assembly flowshop* dengan berbagai metode. Allahverdi and Al-Anzi [3] menyelesaikan masalah *two-stage assembly flowshop* menggunakan tiga algoritma yaitu *simulated annealing* (SA), *ant colony optimization* (ACO) dan *self-adaptive differential evolution* (SDE). SA dilaporkan memberikan hasil terbaik dibandingkan dengan ACO dan SDE. SA juga memerlukan waktu komputasi yang lebih rendah. Hsu, et al. [4]

Mahmudy, WF 2014, 'Optimasi penjadwalan two-stage assembly flowshop menggunakan algoritma genetika yang dimodifikasi', *Konferensi Nasional Sistem Informasi (KNSI)*, STMIK Dipanegara, Makassar, 27 Februari - 1 Maret, pp. 478-483.

memodifikasi algoritma heuristik Johnson's rule untuk menyelesaikan masalah ini. Kombinasi dengan *first-fit* memberikan hasil yang terbaik dibandingkan dengan *dispatching rule* yang lain. Neppali, et al. [5] menguji efektifitas berbagai konfigurasi parameter algoritma genetika (*genetic algorithms*/GAs) dalam meyelesaikan masalah ini. *Hypermutation* yang secara adaptif mengubah nilai *mutation-rate* dan nilai *entropy* untuk menguji konvergensi dari populasi digunakan untuk menjaga keragaman antar chromosome dalam populasi.

Makalah ini mengajukan minimasi *makespan* dalam *two-stage assembly flowshop* menggunakan GAs. GAs telah terbukti berhasil dalam menyelesaikan berbagai masalah penjadwalan kompleks dengan area pencarian yang sangat luas [6]. Metode pengujian konvergensi populasi dikembangkan dan diterapkan secara periodik sepanjang generasi. Jika nilai keragaman populasi dibawah nilai threshold maka mutasi akan dilakukan terhadap sebagaian chromosome. Kesederhanaan dan penentuan interval penerapan yang sesuai diharapkan dapat menjaga keragaman populasi tanpa terlalu membebani waktu komputasi. Pada pembahasan selanjutnya, GAs yang dimodifikasi ini disebut *modified genetic algorithm* (ModGA). Kinerja GAs standar dan ModGA yang dimodifikasi ini dibandingkan dengan keluaran solusi optimum menggunakan algoritma *branch-and-bound*.

2. Penjadwalan Two-Stage Assembly Flowshop

Dalam masalah *two-stage assembly flowshop* terdapat *n* job yang secara bersamaan tersedia pada waktu ke-nol. Setiap job memiliki urutan operasi yang sama. Setiap job harus menjalani *m*+1 operasi. *m* operasi pertama diproses oleh *m* mesin paralel pada tahap pertama, operasi terakhir diproses oleh mesin assembly pada tahap kedua [3]. Beberapa notasi yang digunakan adalah:

t_{ij} waktu operasi job pada posisi *i* pada mesin$_j$, i=1,...,n, j=1,...,m

p_i waktu operasi job pada posisi *i* pada mesin assembly

C_i waktu selesai job pada posisi *i* (*complete*)

Job$_k$ dikatakan *complete* jika semua operasinya t_{kj} (j=1,...,m) dan p_k sudah selesai. Operasi p_k hanya bisa dimulai jika semua operasi t_{kj} (j=1,...,m) sudah selesai. Waktu selesai job$_j$ bisa dihitung dengan rumus berikut:

$$C_j = \max\left\{ \max_{k=1,\ldots,m}\left\{ \sum_{i=1}^{j} t_{i,k} \right\} \;,\; C_{j-1} \right\} + p_j \qquad (1)$$

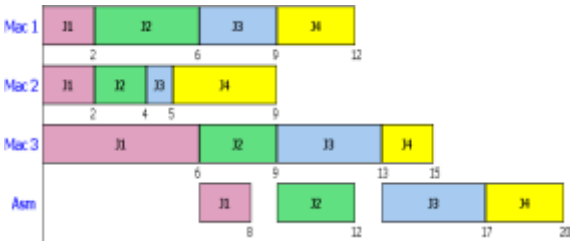
C_0 = 0 dan makespan=C_n.

Misalkan terdapat 4 job yang harus diproses pada 3 mesin dengan waktu operasi disajikan pada Tabel 1.

Tabel 1. Contoh Masalah 4 Job dan 3 Mesin

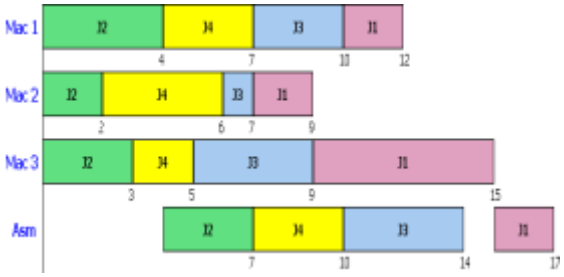
job	mesin			assembly
	1	2	3	
1	2	2	6	2
2	4	2	3	3
3	3	1	4	4
4	3	4	2	3

Misalkan urutan pemrosesan job ditentukan 1-2-3-4 maka pada tahap pertama job 1 diselesaikan terlebih dahulu kemudian diikuti oleh job 2 dan seterusnya. Pada job 1, operasi tahap kedua (*assembly*) bisa dilakukan pada waktu ke-6 setelah setelah semua operasi tahap pertama diselesaikan. *Gantt-chart* dari urutan operasi tersebut ditunjukan pada Gambar 1 dengan nilai makespan=20.



Gambar 1. Gantt-chart untuk urutan job 1-2-3-4

Urutan pemrosesan job yang berbeda akan menghasilkan nilai makespan yang berbeda. Gambar 2 menunjukkan urutan 2-4-3-1 menghasilkan nilai makespan=17. Proses optimasi dilakukan untuk mencari urutan job yang menghasilkan nilai makespan paling kecil.



Gambar 2. Gantt-chart untuk urutan job 2-4-3-1

3. Algoritma Genetika

Algoritma genetika (*genetic algorithms / GAs*) termasuk dalam kelas algoritma pencarian yang menirukan proses evolusi alami. GAs banyak digunakan untuk menyelesaikan berbagai masalah dalam setiap lini industri manufaktur, misalnya dalam perencanaan produksi [7], keseimbangan alokasi pekerjaan ke tiap mesin (*machine loading*

Mahmudy, WF 2014, 'Optimasi penjadwalan two-stage assembly flowshop menggunakan algoritma genetika yang dimodifikasi', *Konferensi Nasional Sistem Informasi (KNSI)*, STMIK Dipanegara, Makassar, 27 Februari - 1 Maret, pp. 478-483.

problem) [8-11], penjadwalan produksi [6, 12], manajemen inventori [13] dan distribusi produk [14].

GAs menggunakan kromosom (*chromosome*) untuk mengkodekan sebuah kemungkinan solusi. Sejumlah kromosom dalam populasi akan mengalami proses reproduksi (tukar silang/*crossover* dan mutasi) untuk membentuk generasi berikutnya. Induk untuk proses reproduksi dipilih secara random dari populasi. Setiap kromosom mempunyai nilai kebugaran (*fitness*) yang menentukan peluangnya untuk tetap bertahan hidup dalam generasi berikutnya. Dengan mekanisme seleksi ini diharapkan nilai *fitness* setiap chromosome akan meningkat pada setiap generasi. Pada akhir generasi, chromosome dengan nilai *fitness* terbaik akan diuraikan menjadi sebuah solusi [1]. Solusi ini mungkin bukan merupakan solusi optimum tetapi fakta empirik membuktikan dengan menentukan paramater-parameter seperti ukuran populasi, *crossover-rate* dan *mutation-rate* yang sesuai, GAs akan memberikan hasil yang memuaskan (mendekati optimum) dalam waktu yang relatif cepat [2].

Siklus lengkap dari GAs ditunjukkan pada Gambar 3. Siklus ini berbeda dangan GAs yang diusulkan Holland [15] yang mendefinisikan seleksi sebagai proses untuk memilih induk untuk menjalani reproduksi.

```
Tentukan parameter GAs
    Ukuran populasi pop_size, crossover rate
    cr, mutation rate mr
Inisialisasi
    Bangkitkan random chromosomes sebanyak
    pop_size
WHILE not kondisi_berhenti
    Produksi chromosome (anak) baru
        Produksi pop_sizexcr anak
        menggunakan crossover
        Produksi pop_sizexmr anak
        menggunakan mutasi
    Seleksi
        Bentuk generasi berikutnya dengan
        memilih pop_size chromosome dari
        parents (populasi) and himpunan anak
    Terapkan prosedur pencegahan konvergensi
        dini
END WHILE
```

Gambar 3. Siklus GAs

3.1. Representasi Kromosom

Sebuah jadwal direpresentasikan dalam bentuk kromosom (*chromosome*). String kromosom ini tersusun atas sejumlah *gen* yang menggambarkan variabel-variabel keputusan yang digunakan dalam solusi. Representasi string kromosom beserta fungsi *fitness* untuk menilai seberapa bagus sebuah kromosom (untuk menjadi solusi yang layak) harus didefinisikan. Dalam banyak kasus, bagaimana merepresentasikan sebuah solusi menjadi kromosom sangat menentukan kualitas dari solusi yang dihasilkan [10].

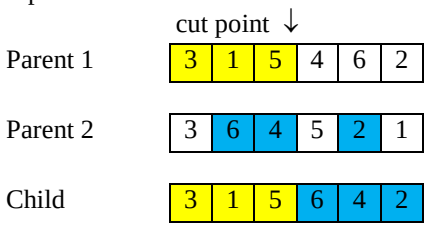
Dalam penelitian ini pengkodean permutasi digunakan sehingga setiap kromosom berisi untaian bilangan integer yang menunjukkan nomer dari job. Posisi (indeks) menunjukkan urutan penyelesaian seluruh job. Misalkan terdapat 6 job maka contoh urutan penyelesaian seluruh job bisa digambarkan dalam Gambar 4. Job 2 diselesakan terlebih dahulu, diikuti oleh job 5, dan seterusnya sampai job 3 dikerjakan terakhir.

Posisi	1	2	3	4	5	6
Gen	2	5	1	6	4	3

Gambar 4. Representasi kromosom

3.2. Crossover

Operator *crossover* digunakan untuk menghasilkan himpunan solusi baru (*offspring*) dari proses rekombinasi dua individu (*parents*). Pada makalah ini digunakan modifikasi *one-cut-point crossover* yang biasa digunakan pada representasi biner. Seperti ditunjukkan pada Gambar 5, segment kiri dari kromosom child didapatkan dari parent 1 dan segmen kanan didapatkan dari urutan gen tersisa dari parent 2.



Gambar 5. Crossover pada Representasi Permutasi

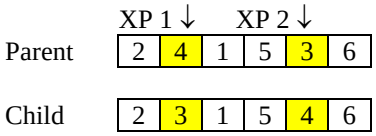
Dalam tahap ini harus ditentukan tingkat *crossover* (*crossover rate*). Nilai ini menyatakan rasio *offspring* yang dihasilkan proses *crossover* terhadap ukuran populasi (*popSize*) sehingga akan dihasilkan *offspring* sebanyak *pc* x *popSize*.

3.3. Mutasi

Operator mutasi digunakan untuk menghasilkan anak dengan melakukan perubahan pada satu individu. Pada penelitian ini digunakan *reciprocal exchange mutation* dan *insertion mutation*. Dua metode mutasi ini dipilih secara acak pada setiap generasi untuk menghasilkan populasi yang lebih beragam [16].

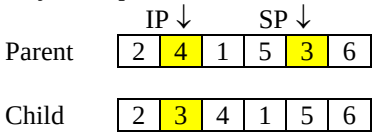
Reciprocal exchange mutation bekerja dengan memilih dua posisi (*exchange point* / XP) secara random kemudian menukarkan nilai pada posisi tersebut seperti ditunjukkan pada Gambar 6.

Mahmudy, WF 2014, 'Optimasi penjadwalan two-stage assembly flowshop menggunakan algoritma genetika yang dimodifikasi', *Konferensi Nasional Sistem Informasi (KNSI)*, STMIK Dipanegara, Makassar, 27 Februari - 1 Maret, pp. 478-483.



Gambar 6. *Reciprocal exchange mutation*

Insertion mutation bekerja dengan memilih satu posisi (*selected point* / SP) secara random kemudian mengambil dan menyisipkan nilainya pada posisi lain (*insertion point* / IP) secara random seperti ditunjukkan pada Gambar 7.



Gambar 7. *Insertion mutation*

3.4. Seleksi

Seleksi dilakukan untuk memilih individu dari himpunan populasi dan *offspring* yang dipertahankan hidup pada generasi berikutnya. Semakin besar nilai *fitness* sebuah kromosom maka semakin besar peluangnya untuk terpilih. Hal ini dilakukan agar terbentuk generasi berikutnya yang lebih baik dari generasi sekarang. Metode seleksi yang sering digunakan adalah *roulette wheel*, *binary tournamen*, *elitism*, dan *replacement [10]*. *Elistism selection* menunjukkan kinerja terbaik pada beberapa percobaan pendahuluan pada penelitian ini. Metode seleksi ini bekerja dengan mengumpulkan semua chromosome induk dan anak dan memilih *popSize* chromosome terbaik untuk generasi berikutnya.

3.5. Penanganan Konvergensi Dini

Kinerja GAs sangat dipengaruhi oleh keseimbangan kemampuan eksplorasi dan eksploitasi yang dilakukan sepanjang generasi. Problem utama yang dialami oleh GAs adalah konvergensi dini yang disebabkan oleh kurangnya diversitas populasi setelah melewati sekian generasi [17]. Crossover antar kromosom yang mirip akan menghasilkan anak yang mirip dengan induknya. Jika hal ini terjadi maka interasi GAs tidak akan mampu menghasilkan solusi yang lebih baik hanya dalam beberapa generasi saja.

Satu metode sederhana yang bisa diterapkan adalah dengan melakukan *random injection* yaitu proses seleksi hanya memilih *popSize-n* kromosom (n =1 ..3). n kromosom terakhir dibangkitkan secara random seperti pada saat inisialisasi [8, 9]. Dengan memasukkan n kromosom random ini maka keragaman populasi akan tetap terjaga karena kromosom ini juga terlibat dalam proses reproduksi. Untuk menghemat waktu komputasi, pemasukan kromosom random ini tidak harus pada setiap generasi tapi bisa dilakukan setiap g interval generasi. Penentuan nilai g yang sesuai memerlukan beberapa percobaan pendahuluan.

Pada penelitian ini metode pengujian konvergensi populasi dikembangkan dan diterapkan secara periodik sepanjang generasi. Jika nilai keragaman populasi dibawah nilai threshold maka mutasi akan dilakukan terhadap sebagaian kromosom. Gambar 8 menunjukkan perhitungan kemiripan dari dua kromosom. Setiap gen kromosom dibandingkan dan rasio kemiripan (*simmilarity ratio*) dinyatakan dalam persentase.

```
PROCEDURE Simmilarity
    Input:  chromosome1, chromosome2
    Output: simmilarity_ratio

    length ← LengthOf (chromosome1)
    sim ← 0
    FOR i=1 TO length DO
      IF chromosome1[i]= chromosome2[i] THEN
        sim ← sim + 1
      END IF
    NEXT i
    simmilarity_ratio ← sim/length * 100
    RETURN simmilarity_ratio
END PROCEDURE
```

Gambar 8. Pseudo-code perhitungan rasio kemiripan dua kromosom

Gambar 9 menunjukkan perhitungan nilai keragaman (*diversity*) dari populasi (*pop*). Setiap kromosom dalam populasi dibandingkan dan dihitung rasio kemiripannya. Nilai keragaman didapatkan dengan mengurangi 1 dengan rata-rata rasio kemiripan.

```
PROCEDURE PopulationSimmilarity
    Input: pop, pop_size
    Ouput: diversity

    total ← 0
    n ← 0
    FOR i=1 TO popSize-1 DO
      FOR j=i+1 TO popSize DO
        total ← total + Simmilarity (pop[i],
    pop[j])
        n ← n + 1
      NEXT j
    NEXT i
    diversity ← 1-total/n
END PROCEDURE
```

Gambar 9. Pseudo-code perhitungan nilai keragaman populasi

4. Hasil dan Pembahasan

Untuk mengevaluasi kinerja ModGA, tiga kelompok data uji digunakan. Masing-masing kelompok data uji memiliki banyaknya job 10, 15, dan 20 yang mewakili data berukuran kecil, sedang, dan besar seperti ditunjukan pada Tabel 2. Untuk setiap kelompok data uji digunakan 3 dan 6 mesin.

Mahmudy, WF 2014, 'Optimasi penjadwalan two-stage assembly flowshop menggunakan algoritma genetika yang dimodifikasi', *Konferensi Nasional Sistem Informasi (KNSI)*, STMIK Dipanegara, Makassar, 27 Februari - 1 Maret, pp. 478-483.

Tabel 2. Kelompok Data Uji

Kelompok	Job	Mesin
1	10	3
		6
2	15	3
		6
3	20	3
		6

Program komputer untuk ModGA ditulis menggunakan Turbo Delphi. Percobaan dilakukan untuk membuktikan efektifitas ModGA dibandingkan dengan GAs standar (GA). Karena GAs bersifat stokastis maka hasil yang berbeda akan didapatkan setiap kali program dijalankan pada data uji yang sama. Untuk mendapatakan hasil dan kesimpulan yang *fair* maka untuk setiap pasangan job dan mesin, percobaan diulang sebanyak 100 kali sehingga total percobaan adalah sebanyak 2x6x100=1200 percobaan.

Serangkaian percobaan pendahuluan dilakukan untuk mendapatkan kombinasi nilai parameter yang tepat bagi GA dan ModGA serta didapatkan hasil sebagai berikut:

- Ukuran populasi sebesar 50.
- Banyaknya generasi sebesar 500.
- *Crossover rate* sebesar 0,2.
- *Mutation rate* sebesar 0,05.
- Threshold nilai keragaman populasi sebesar 0,25 dan pengecekan dilakukan setiap 50 generasi.

Untuk mengukur kinerja GA dan ModGA, solusi optimum dari semua data uji didapatkan dengan menggunakan metode *branch-and-bound*. Meskipun metode ini bisa digunakan untuk mendapatkan solusi optimum, pada data berukuran besar diperlukan waktu komputasi yang sangat tinggi dan tidak bisa diterima pada kondisi riil untuk keperluan penjadwalan harian.

Dua parameter digunakan untuk mengevaluasi kinerja GA dan ModGA. Yang pertama adalah banyaknya solusi optimum yang diperoleh (*number of optimum solutions*/NOS) untuk 100 kali percobaan per data uji. Parameter yang kedua adalah rata-rata deviasi solusi GA (*FGA*) dan ModGA terhadap solusi optimum (F_{opt}) seperti ditunjukkan pada persamaan (2). *DEV* yang lebih kecil menunjukkan hasil yang lebih baik.

$$DEV = \frac{F_{opt} - \left(\sum_{i=1}^{100} FGA_i \Big/ 100 \right)}{F_{opt}} \qquad (2)$$

Hasil keseluruhan percobaan disajikan pada Tabel 3. Pada data uji berukuran kecil dan sedang, ModGA mampu memberikan hasil optimum untuk semua data percobaan. Hal ini ditunjukan dengan nilai NOS sebesar 100. Hasil ini tidak bisa dicapai oleh GA yang mendapatkan NOS sebesar 100 hanya pada data uji dengan job sebanyak 10 dan mesin sebanyak 6.

Pada data berukuran besar, ModGA masih mampu untuk mendapatkan NOS sebesar 99, menunjukkan ModGA tidak mampu mencapai solusi optimum hanya pada 1 percobaan dari keseluruhan 100 kali percobaan. Keunggulan ModGA dibandingkan GA juga ditunjukkan pada nilai *DEV* yang jauh lebih kecil.

Tabel 3. Hasil Percobaan

Job	Mesin	GA		ModGA	
		NOS	DEV	NOS	DEV
10	3	99	0,0230	100	0,0000
	6	100	0,0000	100	0,0000
15	3	99	0,0104	100	0,0000
	6	99	0,0011	100	0,0000
20	3	98	0,0031	99	0,0003
	6	97	0,0348	99	0,0048

5. Kesimpulan dan Saran

Optimasi penjadwalan two-stage assembly flowshop telah diselesaikan dengan algoritma genetika standar (GA) dan algoritma genetika yang dimodifikasi (ModGA). Modifikasi GA dilakukan dengan menambahkan mekanisme untuk mencegah konvergensi dini. Serangkaian 1200 percobaan menunjukkan bahwa ModGA lebih unggul dibandingkan dengan GA. ModGA menghasilkan solusi optimum lebih banyak dan deviasi nilai solusi yang rendah terhadap solusi optimum.

Penelitian ke depan akan memperhitungkan *due date* untuk setiap job. ModGA akan diuji pada data berukuran lebih besar pada sistem manufaktur yang sangat sibuk. Pada kasus ini, selain meminimumkan makespan, ModGA juga harus meminimumkan total keterlambatan (*tardiness*) dari semua job. ModGA yang lebih *powerful* dikembangkan dangan melakukan hibridisasi dengan metode heuristik lain seperti *simulated annealing* dan *tabu search*.

6. Daftar Pustaka

[1] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*. New York: John Wiley & Sons, Inc., 1997.

[2] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*. New York: John Wiley & Sons, Inc., 2000.

[3] A. Allahverdi and F. S. Al-Anzi, "The two-stage assembly flowshop scheduling problem with bicriteria of makespan and

Mahmudy, WF 2014, 'Optimasi penjadwalan two-stage assembly flowshop menggunakan algoritma genetika yang dimodifikasi', *Konferensi Nasional Sistem Informasi (KNSI)*, STMIK Dipanegara, Makassar, 27 Februari - 1 Maret, pp. 478-483.

mean completion time," *Int J. Adv. Manuf. Technol*, vol. 37, pp. 166–177, 2008.

[4] C. J. Hsu, W. H. Kuo, D. L. Yang, and M. S. Chern, "Minimizing the makespan in a two-stage flowshop scheduling problem with a function constraint on alternative machines," *Journal of Marine Science and Technology*, vol. 14, pp. 213-217, 2006.

[5] V. R. Neppali, C. L. Chen, and J. N. D. Gupta, "Genetic algorithms for the two-stage bicriteria flowshop problem," *European Journal of Operational Research*, vol. 95, pp. 356-373, 1996.

[6] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Real coded genetic algorithms for solving flexible job-shop scheduling problem – Part I: modeling," *Advanced Materials Research*, vol. 701, pp. 359-363, 2013.

[7] R. T. Moghaddam and N. Safaei, "Solving a generalized aggregate production planning problem by genetic algorithms," *Journal of Industrial Engineering International*, vol. 2, pp. 53-64, 2006.

[8] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Optimization of part type selection and loading problem with alternative production plans in flexible manufacturing system using hybrid genetic algorithms – Part 1: modelling and representation," in *5th International Conference on Knowledge and Smart Technology (KST)*, Chonburi, Thailand, 2013, pp. 75-80.

[9] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Optimization of part type selection and loading problem with alternative production plans in flexible manufacturing system using hybrid genetic algorithms – Part 2: genetic operators & results," in *5th International Conference on Knowledge and Smart Technology (KST)*, Chonburi, Thailand, 2013, pp. 81-85.

[10] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Solving part type selection and loading problem in flexible manufacturing system using real coded genetic algorithms – Part I: modeling," *World Academy of Science, Engineering and Technology*, vol. 69, pp. 699-705, 2012.

[11] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Solving part type selection and loading problem in flexible manufacturing system using real coded genetic algorithms – Part II: optimization," *World Academy of Science, Engineering and Technology*, vol. 69, pp. 706-710, 2012.

[12] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Real coded genetic algorithms for solving flexible job-shop scheduling problem – Part II: optimization," *Advanced Materials Research*, vol. 701, pp. 364-369, 2013.

[13] P. Radhakrishnan, V. M. Prasad, and M. R. Gopalan, "Optimizing inventory using genetic algorithm for efficient supply chain management," *Journal of Computer Science*, vol. 5, pp. 233-241, 2009.

[14] M. Gen, F. Altiparmak, and L. Lin, "A genetic algorithm for two-stage transportation problem using priority-based encoding," *OR Spectrum*, vol. 28, pp. 337–354, 2006.

[15] J. H. Holland, *Adaptation in Natural and Artificial System*. Ann Arbor: Michigan Univ. Press, 1975.

[16] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Hybrid genetic algorithms for part type selection and machine loading problems with alternative production plans in flexible manufacturing system," *Accepted in: ECTI-CIT Trans, Special Issue on Knowledge and Smart Technologies*, 2013.

[17] M. Lozano and F. Herrera, "Fuzzy adaptive genetic algorithms: design, taxonomy, and future directions," *Soft Computing*, vol. 7, pp. 545–562, 2003.

Mahmudy, WF 2014, 'Optimasi penjadwalan two-stage assembly flowshop menggunakan algoritma genetika yang dimodifikasi', *Konferensi Nasional Sistem Informasi (KNSI)*, STMIK Dipanegara, Makassar, 27 Februari - 1 Maret, pp. 478-483.