PENGEMBANGAN ALGORITMA *VIRAL SYSTEMS* UNTUK MASALAH PENJADWALAN *HYBRID FLOW SHOP* UNTUK MEMINIMASI *MAKESPAN*

Hotna Marina Sitorus, Cynthia Juwono, Kevin P. Dwikaragus
Jurusan Teknik Industri, Universitas Katolik Parahyangan
Jl. Ciumbuleuit 94, Bandung 40141
Email: nina@unpar.ac.id atau ninasitorus@gmail.com

ABSTRAK

Penelitian ini bertujuan untuk mengembangkan algoritma Viral Systems untuk menyelesaikan masalah penjadwalan Hybrid Flow Shop (HFS) untuk meminimasi makespan. Algoritma Viral Systems merupakan metode metaheuristik yang terinspirasi dari cara kerja virus dalam menginfeksi organisme lain. HFS merupakan sistem produksi flow shop yang memiliki karakteristik utama yaitu jumlah tahapan proses produksi (stage) minimal berjumlah 2 dan terdapat minimal 1 stage yang memiliki mesin paralel. Penjadwalan HFS merupakan masalah optimasi kombinatorial, dimana sejumlah n job akan diproses pada m stage untuk memaksimalkan fungsi tujuan. Penelitian ini membahas permasalahan penjadwalan HFS dengan sistem multiprocessor, dimana setiap job dalam sebuah stage dapat diproses pada lebih dari 1 mesin secara simultan. Algoritma penjadwalan yang dikembangkan diujikan pada 6 buah benchmark problems. Performansi algoritma kemudian dibandingkan dengan Genetic Algorithms dan Artificial Bee Colony. Perbandingan performansi dilakukan dengan menggunakan nilai lower bound, yaitu nilai minimum makespan yang dapat dicapai secara teoritis. Berdasarkan perhitungan yang dilakukan, baik algoritma yang dihasilkan maupun Genetic Algorithm dan Artificial Bee Colony tidak mampu mencapai lower bound pada setiap kasus. Dibandingkan dengan algoritma Genetic Algorithm, algoritma yang dikembangkan unggul di 2 kasus, sementara dibandingkan dengan Artificial Bee Colony, algoritma yang dikembangkan unggul di 5 kasus. Hal ini menunjukkan algoritma yang dihasilkan mampu secara efektif menyelesaikan masalah penjadwalan HFS multiprocessor.

***Kata kunci**: Algoritma Viral systems, penjadwalan produksi, Hybrid Flow Shop*

PENDAHULUAN

Penjadwalan dalam sistem produksi dapat diartikan sebagai kegiatan mengalokasikan sejumlah pekerjaan ke sejumlah sumber daya yang ada (Bedworth & Bailey, 1987). Penjadwalan merupakan hal yang penting bagi kegiatan produksi di sebuah perusahaan, karena berpengaruh terhadap produktivitas produksi perusahaan tersebut. Jadwal yang dihasilkan harus menghasilkan efisiensi di sistem produksi perusahaan, sehingga berakibat pada tingkat utilisasi perusahaan yang tinggi. Tingkat utilisasi yang tinggi akan menjaga kapasitas produksi perusahaan di level yang maksimal sehingga otomatis akan meningkatkan keuntungan bagi perusahaan.

Perusahaan harus bisa menggunakan metode penjadwalan yang tepat, yaitu disesuaikan dengan situasi atau strategi yang diterapkan perusahaan dan juga jenis produk yang dihasilkan oleh perusahaan. Terdapat beberapa *process positioning strategy* yang bisa diterapkan di perusahaan, salah satunya adalah *flow shop*. Strategi ini biasanya dilakukan jika perusahaan menghasilkan variasi produk yang relatif sedikit (Fogarty et al., 1991). Pada strategi ini mesin - mesin disusun sedemikian rupa sehingga semua produk akan diproses dengan 1 arah proses.

Hybrid flow shop (HFS) merupakan sebuah sistem yang terdiri atas berbagai tahap (*stage*) proses produksi dan material akan diproses dalam aliran yang searah, dimana setidaknya satu *stage* dalam proses produksi memiliki mesin paralel (Uetake et al., 1995). Dengan karakteristik ini, dapat dikatakan HFS merupakan kombinasi yang menggabungkan karakteristik *flow shop* dan mesin paralel. Pada sistem *flow shop*, produk mendatangi mesin dengan urutan tahap yang sama, dimana pada setiap tahap terdiri atas 1 buah mesin. Pada sistem HFS, setiap tahap dapat terdiri atas 1 atau lebih mesin identik (Serifoglu & Ulusoy, 2004). Sistem HFS ditemukan pada berbagai jenis industri, di antaranya industri elektronik, kertas, tekstil (Ruiz & Rodriguez, 2010), otomotif dan kimia (Amin-Naseri & Beheshti-Nia, 2008).

Dalam prakteknya permasalahan penjadwalan HFS banyak ditemukan, terutama pada industri yang memiliki mesin majemuk dalam setiap tahap produksinya. Penjadwalan HFS merupakan masalah penjadwalan dimana n *job* akan diproses pada m tahapan proses (*stage*) untuk memaksimalkan fungsi tujuan (Ruiz & Rodriguez, 2010). Berbagai penelitian telah dilakukan untuk membahas permasalahan penjadwalan HFS. Uetake et al. (1995) memformulasikan *makespan* dan maksimum WIP sebagai kriteria dalam penjadwalan HFS. Gupta (1998) membahas kompleksitas

masalah penjadwalan 2 *stage* HFS dan merancang metode heuristik untuk meminimasi *makespan*. Engin dan Doyen (2004) menerapkan *Artificial Immune System* dalam penjadwalan HFS.

Dalam perkembangannya, permasalahan penjadwalan HFS menjadi semakin bervariasi (Linn & Zhang, 1999; Ruiz & Rodriguez, 2010). Permasalahan HFS dengan sistem *multiprocessor task* muncul pada saat terdapat kebutuhan untuk memproses *job* pada lebih dari 1 mesin pada suatu *stage* secara simultan. Penjadwalan HFS dengan sistem *multiprocessor task* ini semakin banyak dibahas dalam komunitas penelitian (Choong et al., 2011). Beberapa penelitian telah dilakukan untuk menyelesaikan permasalahan ini menggunakan metode metaheuristik, di antaranya *Oguz et al.* (2002) yang menggunakan *Tabu Search*, Serifoglu & Ulusoy (2004) menggunakan *Genetic Algorithm*, dan Kusumo (2012) menggunakan *Artificial Bee Colony*.

Penelitian ini membahas permasalahan penjadwalan HFS dengan sistem *multiprocessor task* dengan menggunakan konsep *Viral Systems* (VS). VS merupakan metode metaheuristik yang terinspirasi cara kerja virus dalam menginfeksi organisme lain yang dikemukakan oleh Cortes et al. (2007). Pada penelitian tersebut, VS digunakan untuk menyelesaikan permasalahan *Steiner Problem*, dan algoritma yang dihasilkan mampu memberikan solusi yang lebih baik dibandingkan *Genetic Algorithms* dan *Tabu Search*.

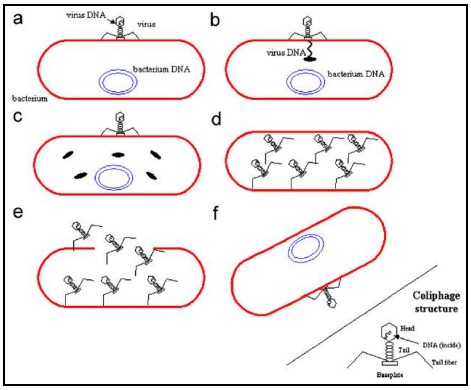
KERANGKA TEORITIS

Viral Systems

Viral Systems merupakan suatu algoritma metaheuristik yang terinspirasi oleh performansi virus. Mekanisme replikasi virus, seperti proses menginfeksi inangnya digunakan untuk mengembangkan suatu algoritma metaheuristik yang dapat menghasilkan suatu nilai yang berharga. Virus merupakan parasit intraseluler yang terbentuk dengan *nucleic acids*, seperti DNA atau RNA, dan protein. Protein membentuk kapsul, disebut dengan *capsid*. Di dalam *capsid* terdapat *nucleic acid*. *Capsid* dan *nucleic acid* membentuk *nucleus-capsid*, yang mengidentifikasikan sebuah virus.

Setiap jenis virus memiliki kemampuan yang berbeda-beda untuk menginfeksi sel, termasuk mekanisme replikasi dan inokulasi, dan strategi untuk melemahkan sistem kekebalan inangnya. Virus berkembang biak dengan mereplikasi dirinya agar menjadi banyak. Meskipun terdapat berbagai jenis virus, VS mengadopsi mekanisme replikasi yang terjadi pada virus yang menyerang bakteri. Mekanisme replikasi virus ini dapat terbagi menjadi 2 macam, yaitu replikasi *lytic* dan replikasi *lysogenic* seperti tampak pada Gambar 1 dan 2. Proses replikasi *lytic* adalah sebagai berikut (Cortes, et. al. 2007):

1. Virus menempel pada dinding bakteri kemudian memasukkan DNA virus ke dalam bakteri tersebut.
2. Sel yang terinfeksi virus mulai memproduksi protein virus untuk mereplikasi virus tersebut.
3. Setelah melakukan replikasi, dinding bakteri pecah, dan virus-virus baru hasil replikasi keluar untuk menginfeksi sel lainnya.



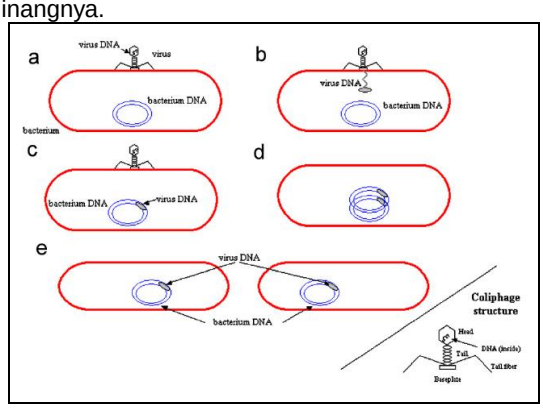
Gambar 1 Ilustrasi Mekanisme Replikasi *Lytic* (Cortes et al., 2007)

Proses replikasi *lysogenic* adalah sebagai berikut (Cortes, et. al. 2007):

1. Virus menginfeksi inangnya dan memasukkan DNA virus ke dalam genom inangnya.
2. DNA virus tetap berada dalam inangnya hingga pada suatu waktu dapat aktif saat terkena sinar ultraviolet ataupun sinar X.
3. Setelah DNA virus aktif, pada saat sel inangnya melakukan replikasi, secara tidak langsung DNA virus yang tersembunyi dalam genom sel turut tereplikasi.

Siklus hidup virus akan terus-menerus berlanjut dengan melakukan replikasi *lytic* atau *lysogenic*. Beberapa virus dapat membentuk respon antigen terhadap organisme inangnya saat inangnya melakukan pertahanan dengan membentuk antibodi terhadap serangan virus tersebut. Infeksi virus dapat berakhir melalui 2 cara, yaitu saat organisme berhasil mengalahkan virus dengan

pertahanan diri atau saat virus berhasil mengalahkan pertahanan antibodi inangnya dan mengakibatkan kematian inangnya.



Gambar 2 Ilustrasi Mekanisme Replikasi *Lysogenic* (Cortes et al., 2007)

Dalam algoritma VS terdapat 3 tahapan utama yang perlu dilakukan untuk memperoleh solusi. Tahapan yang dilakukan adalah sebagai berikut (Cortes et al., 2007):

1. Tahap inisiasi
   Pada tahap ini ditentukan keadaan awal (*state*) dari virus dan organisme. Hal pertama yang dilakukan adalah membuat *clinical picture* awal dan menentukan jenis infeksi virus. Terdapat dua tipe infeksi dalam VS, yaitu *selective infection* dan *massive infection.* Pada *selective infection*, virus hanya menginfeksi satu sel atau memilih sel yang akan diinfeksi, sedangkan pada *massive infection* virus menginfeksi tubuh organisme secara menyeluruh.
2. Tahap *steady state*
   Tahapan *steady state* merupakan tahapan yang menggambarkan seluruh interaksi virus dengan organisme, baik berupa input maupun outputnya. Pada tahap ini dilakukan penentuan tipe replikasi, mekanisme replikasi yang terjadi, penentuan tetangga yang terinfeksi. Tahapan ini menggambarkan sebagian besar tahapan algoritma VS dalam mencari solusi.
3. Tahap akhir
   Terdapat 2 kondisi yang membuat infeksi virus berakhir, yang pertama adalah matinya organisme yang terinfeksi dan yang kedua virus berhasil terisolasi. Kematian organisme terjadi saat perbedaan antara solusi terbaik yang diperoleh dan nilai *lower bound* (LB) lebih kecil daripada nilai *gap* yang telah ditentukan. Virus dikatakan terisolasi jika virus telah mencapai jumlah iterasi maksimum, yang menandakan bahwa organisme bertahan terhadap serangan infeksi virus.

Terdapat 8 parameter yang digunakan dalam VS, yaitu jumlah iterasi (N), ukuran *clinical picture* (POB), batas iterasi *lytic* awal (LNR^0), batas iterasi *lysogenic* awal (LIT^0), peluang terjadinya replikasi *lytic* (plt), peluang infeksi sel tetangga (pi), peluang replikasi nucleus-capsid (pr), dan peluang sel menghasilkan antigen (pan).

Lower Bound

Evaluasi performansi dilakukan dengan mengujikan algoritma yang dihasilkan pada benchmark problems. Ketiadaan informasi mengenai nilai solusi optimal untuk benchmark problems yang tersedia diakomodasi dengan penggunaan nilai Lower Bound (LB), yaitu batas bawah (nilai minimum) yang dapat dicapai secara teoritis dari solusi untuk sebuah kasus. Kriteria yang umum digunakan untuk masalah penjadwalan HFS adalah *makespan*, nilai LB dalam masalah penjadwalan HFS dihitung menggunakan persamaan berikut (Serifoglu & Ulusoy, 2004):

$$LB = \max_{i \in M}\left\{\min_{j \in J}\left\{\sum_{k=1}^{i-1} p[k,j]\right\} + \max\left\{\left(\sum_{j \in A_i} p[i,j] + \left[\frac{\sum_{j \in B_i} p[i,j]\, size[i,j]}{m_i}\right]\right), \quad \frac{1}{m_i}\sum_{j \in J} p[i,j]\, size[i,j]\right\} + \min_{j \in J}\left\{\sum_{k=i+1}^{m} p[k,j]\right\}\right\} \quad \text{..... (1)}$$

dimana *i* adalah indeks *job* (i = 1, 2, …, M), *j* adalah indeks *job* (j = 1, 2, …, J), *p[i,j]* merupakan waktu proses *job* ke-j pada *stage* i, *size[i,j]* merupakan jumlah kebutuhan mesin *job* ke-j pada *stage* i, *mi* merupakan jumlah mesin paralel pada *stage* i, *Ai* merupakan kumpulan *job* dengan jumlah kebutuhan mesin yang lebih besar dari mi/2, dan *Bi* merupakan kumpulan *job* dengan jumlah kebutuhan mesin yang sama dengan mi/2.

Perbandingan antara *makespan* suatu jadwal HFS dengan LB dilakukan dengan menghitung persentase deviasi keduanya. Perhitungan persentase deviasi tersebut adalah sebagai berikut (Serifoglu & Ulusoy, 2004):

$$z = 100 \times (Cmax - LB) / LB \quad \text{..... (2)}$$

dimana LB adalah *lower bound* dan Cmax adalah kriteria yang digunakan.

METODOLOGI PENELITIAN

Pengembangan algoritma VS dalam menyelesaikan permasalahan penjadwalan HFS dengan *multiprocessor task* (selanjutnya disebut algoritma HFS-VS) dilakukan dengan langkah-langkah berikut:

1. Pengkodean solusi
   Pada tahap ini dilakukan pengkodean matematis solusi pada masalah penjadwalan HFS ke dalam model *Viral Systems*. Pengkodean yang digunakan dalam penelitian ini adalah *operation-based representation*.
2. Perancangan algoritma penjadwalan HFS-VS
   Pada tahap ini dirancang algoritma penjadwalan HFS-VS, yang dimulai dari perancangan algoritma pencarian solusi awal, algoritma pencarian solusi tetangga (replikasi *lytic*), algoritma pencarian solusi mutasi (replikasi *lysogenic*), algoritma penyimpanan solusi, hingga algoritma penjadwalan dan perhitungan nilai *fitness*.
3. Verifikasi dan validasi algoritma
   Langkah ini bertujuan untuk memastikan algoritma penjadwalan HFS-VS yang dirancang telah sesuai dengan konsep *Viral Systems* dan mampu menggambarkan dan menyelesaikan permasalahan penjadwalan HFS.
4. Studi komputasional
   Algoritma yang dirancang diujikan pada 6 buah *benchmark problems* yang diambil dari penelitian Serifoglu & Ulusoy (2004). Studi komputasional ini dilakukan dengan bantuan perangkat lunak yang dirancang khusus sesuai algoritma penjadwalan HFS-VS yang dihasilkan. Pada langkah ini dicatat nilai *makespan* terbaik yang dihasilkan pada masing-masing kasus. Sebelum hasil yang diperoleh digunakan pada tahap selanjutnya, dilakukan pengujian verifikasi dan validasi perangkat lunak yang digunakan.
5. Evaluasi performansi algoritma
   Perbandingan performansi algoritma dilakukan dengan mengukur persentase deviasi antara hasil terbaik (*makespan* terbaik) yang diperoleh dengan nilai *lower bound* pada setiap kasus.

HASIL DAN BAHASAN

Perancangan Algoritma HFS-VS

Genom merupakan pengkodean matematis solusi pada VS. Bentuk genom yang dipakai dalam penelitian ini adalah *operation-based representation*, yang diadaptasi dari penjadwalan *job shop* pada algoritma genetika (Gen et al., 1997). Pada pengkodean ini setiap operasi pada *job* yang sama akan dikodekan dengan kode yang sama sesuai dengan nomor *job*. Urutan gen pada genom menunjukkan urutan penjadwalan, sehingga 1 genom akan berisi 1 urutan penjadwalan. Jumlah gen dalam sebuah genom adalah perkalian dari jumlah *job* yang akan dijadwalkan dikali dengan jumlah *stage* yang ada, atau dengan kata lain semua *job* akan muncul pada genom sebanyak jumlah *stage* yang ada. Dalam HFS setiap *job* akan melewati urutan *stage* yang sama, namun suatu *job* dapat saja tidak diproses pada setiap *stage*. *Job* yang tidak diproses pada suatu *stage* ditandai dengan mengisi waktu proses di *stage* tersebut sebesar nol.

Struktur umum algoritma HFS-VS yang dihasilkan adalah sebagai berikut :

1. Masukkan informasi awal dari permasalahan penjadwalan HFS dan juga parameter VS:
   a. Jumlah *job* yang akan dijadwalkan (j)
   b. Jumlah *stage* (K)
   c. Jumlah mesin identik di masing - masing *stage* (mk)
   d. Waktu proses masing - masing *job* di setiap *stage* (Pik)
   e. Jumlah kebutuhan mesin masing - masing *job* di tiap *stage* (Sik)
   f. Batas iterasi *lysogenic* awal (LIT^0)
   g. Batas iterasi *lytic* awal (LNR^0)
   h. Jumlah iterasi yang akan dilakukan (N)
   i. Peluang sel atau genom menghasilkan antigen (pan)
   j. Peluang infeksi tetangga (pi)
   k. Peluang replikasi *lytic* (plt)
   l. Peluang replikasi *nucleus capsids* (pr)
   m. Ukuran *clinical picture* (POB)
2. Menghitung batas nilai pan. Nilai pan yang diinput harus sesuai dengan batas yang dihasilkan oleh algoritma tersebut.
3. Mencari sejumlah solusi awal (genom) dengan menggunakan Algoritma Pencarian Solusi Awal.
4. Simpan solusi (jadwal dan *makespan*) terbaik saat ini.
5. Mulai melakukan iterasi pertama (set n = 1).
6. Lakukan pemeriksaan genom pertama di *clinical picture* (CP) (set G = 1).

7. Lakukan pemeriksaan antigen setiap genom, jika terdapat antigen hapus genom tersebut dari CP dan lanjutkan dengan pemeriksaan genom selanjutnya, jika tidak lanjut ke proses penentuan jenis replikasi.
8. Tentukan jenis replikasi. Jika replikasi *lytic* lakukan sesuai Algoritma *Lytic*, jika replikasi *lysogenic* lakukan sesuai Algoritma *Lysogenic*.
9. Lakukan langkah 7 dan 8 sampai semua genom di CP diperiksa.
10. Perbaharui jumlah genom yang ada di CP sekarang.
11. Periksa apakah CP kosong, lalu juga cek apakah terdapat sel mutasi dan tetangga yang dihasilkan. Jika CP tidak kosong dan tidak terdapat sel mutasi juga sel tetangga yang dihasilkan, lanjut ke iterasi selanjutnya (n = n+1). Jika terdapat sel mutasi dan atau sel tetangga yang dihasilkan, lanjut ke langkah 12. Jika CP kosong dan tidak terdapat sel mutasi dan tetangga yang dihasilkan, lanjut ke langkah 15.
12. Masukkan genom tetangga dan mutasi yang dihasilkan pada CP sesuai dengan Algoritma Simpan Solusi.
13. Perbaharui solusi (jadwal dan *makespan*) terbaik saat ini.
14. Lakukan langkah 6 sampai 13, sampai mencapai iterasi maksimum (N).
15. Tampilkan solusi (jadwal dan nilai *makespan*) terbaik hasil algoritma.

Langkah - langkah Algoritma Pencarian Solusi Awal adalah sebagai berikut :

1. Mulai dengan membentuk genom pertama (Set G = 1).
2. Isi semua gen pada genom ke-G dengan nomor – nomor *job* yang ada secara random, dimana setiap *job* muncul sebanyak jumlah *stage* yang ada di seluruh gen pada suatu genom.
3. Pastikan genom yang telah dihasilkan tidak sama dengan yang telah dihasilkan sebelumnya.
4. Simpan genom yang telah dihasilkan pada CP.
5. Ukur kualitas solusi menggunakan Algoritma Jadwal dan Hitung Nilai *Fitness*.
6. Set Nilai NR dan IT genom tersebut 0.
7. Lakukan langkah 2 sampai 7 sehingga sejumlah POB genom telah dihasilkan.

Langkah - langkah Algoritma *Lytic* adalah sebagai berikut :

1. Hitung batas iterasi replikasi *lytic* genom yang sedang diproses (LNR_G; *roundup*).
2. Tentukan jumlah replikasi *lytic* yang akan ditambahkan.
3. Perbaharui jumlah replikasi *lytic* sekarang dengan menambahkan jumlah replikasi z pada jumlah replikasi *lytic* genom yang sedang diproses (NRG = NRG + z).
4. Periksa apakah jumlah replikasi *lytic* genom (NRG) telah mencapai atau melewati batas replikasinya (LNRG). Jika tidak algoritma ini selesai, jika ya maka sel akan pecah (akan menghasilkan genom tetangga) lanjut ke langkah selanjutnya.
5. Mulai dari gen ke-1 (Set g =1).
6. Bentuk genom baru (genom tetangga) dengan menukarkan posisi genom ke-g dan genom ke-(g+1) pada genom awal.
7. Periksa apakah genom tetangga yang dihasilkan sama dengan genom awal. Jika tidak lanjut ke langkah selanjutnya, jika ya lanjutkan ke nomor gen selanjutnya (g = g+1).
8. Periksa secara random apakah genom tersebut terinfeksi. Jika ya lanjut ke langkah selanjutnya, jika tidak lanjut ke langkah 13.
9. Periksa secara random apakah genom tersebut memiliki antigen. Jika tidak lanjut ke langkah selanjutnya, jika ya lanjut ke langkah 13.
10. Periksa apakah genom tetangga tersebut sudah ada di himpunan genom tetangga. Jika tidak lanjut ke langkah selanjutnya, jika ya lanjut ke langkah 13.
11. Simpan genom tetangga tersebut di himpunan genom tetangga.
12. Ukur kualitas solusi menggunakan Algoritma Jadwal dan Hitung Nilai *Fitness*.
13. Periksa apakah sudah semua kandidat genom tetangga (yang bisa dihasilkan) sudah diperiksa / apakah proses pencarian genom tetangga sudah sampai pada titik terakhir (g = (j x K) - 1). Jika tidak lanjut ke gen selanjutnya (g = g +1) dan kembali ke langkah 6, jika ya lanjut ke langkah selanjutnya.
14. Hapus genom awal dari CP.

Langkah - langkah Algoritma *Lysogenic* adalah sebagai berikut :

1. Perbaharui (tambah) jumlah replikasi *lysogenic* genom yang diproses (ke-G) ($IT_G = IT_G + 1$).
2. Hitung batas iterasi replikasi *lysogenic* genom yang sedang diproses (LIT_G; *roundup*).
3. Periksa apakah jumlah replikasi *lysogenic* genom (ITG) telah mencapai atau melebihi batas replikasi *lysogenic*-nya (LITG). Jika ya lanjut ke langkah selanjutnya, jika tidak maka algoritma ini selesai.
4. Pilih 2 gen secara random. Selisih nomor antara kedua gen tersebut harus lebih besar dari 10% dari panjang genom (j x K) yang dibulatkan ke atas. Jika gen yang terpilih tidak sesuai dengan batasan selisih tersebut, pilih lagi gen secara random.
5. Bentuk genom baru (mutasi) dengan membalikkan posisi dari gen ke-1 sampai gen ke-2 terpilih tersebut pada genom awal (ke-G).

6. Periksa apakah genom baru yang dihasilkan sudah ada di himpunan genom mutasi {M} atau di *clinical picture* {CP}. Jika ya hapus genom tersebut dan kembali ke langkah 4, jika tidak lanjut ke langkah selanjutnya.
7. Simpan genom baru tersebut di himpunan genom mutasi.
8. Ukur kualitas solusi menggunakan Algoritma Jadwal dan Hitung Nilai *Fitness*.
9. Hapus genom awal dari {CP} dan juga hapus nilai *fitness* genom tersebut dari {f}.

Langkah – langkah Algoritma Jadwal dan Hitung Nilai *Fitness* adalah sebagai berikut :

1. Mesin paralel di tiap *stage* diperiksa secara berurutan mulai dari mesin dengan indeks terkecil.
2. Apabila di mesin tersebut terdapat *idle* (ruang kosong yang dimungkinkan terjadi penyisipan suatu operasi) maka setiap *idle* diperiksa mulai dari *idle* paling awal (indeks d terkecil). Jika tidak terdapat *idle* maka yang diperiksa adalah *ready time* dari mesin tersebut.
3. Pemeriksaan *idle* dihentikan ketika *idle* yang diperiksa bisa disisipkan operasi yang akan dijadwalkan.
4. Hitung selisih atau perbedaan antara *ready time idle* atau mesin dengan *completion time* sementara *job* yang operasinya akan dijadwalkan.
5. Jika jumlah kebutuhan mesin suatu operasi adalah satu, maka jadwalkan operasi tersebut di mesin dengan prioritas selisih dari nol, lalu selisih negatif terkecil sampai ke terbesar, lalu selisih positif terkecil sampai terbesar. Jika jumlah kebutuhan mesin suatu operasi lebih dari satu maka akan operasi tersebut akan dijadwalkan di mesin - mesin (dengan jumlah sesuai dengan kebutuhan mesinnya) dimana operasi tersebut bisa diproses (awal dan akhir) bersamaan.

Sebelum dievaluasi lebih lanjut, dilakukan pengujian verifikasi dan validasi algoritma HFS-VS yang dihasilkan. Verifikasi algoritma dilakukan dengan membandingkan dan menyesuaikan langkah - langkah pada algoritma yang dikembangkan dengan langkah - langkah pada penelitian yang dilakukan oleh Cortes et al. (2007). Validasi dilakukan dengan mengujikan algoritma pada sebuah kasus penjadwalan HFS yang sederhana dan melihat kesesuaian output pada setiap langkah algoritma.

Studi Komputasional

Algoritma yang dirancang diujikan pada 6 buah *benchmark problems* yang diambil dari penelitian Serifoglu & Ulusoy (2004). Nilai parameter yang VS yang digunakan dalam studi komputasional dapat dilihat pada Tabel 1. Perhitungan pada setiap kasus dilakukan dengan 100 iterasi dan 5 kali replikasi. Data kasus yang diujikan dan hasil algoritma penjadwalan HFS-VS dapat dilihat pada Tabel 2. Studi komputasional ini dilakukan dengan bantuan perangkat lunak yang dirancang khusus sesuai algoritma penjadwalan HFS-VS yang dihasilkan. Sebelum hasil yang diperoleh digunakan pada tahap selanjutnya, dipastikan bahwa perangkat lunak yang digunakan terverifikasi dan valid.

Tabel 1 Nilai Parameter VS yang Digunakan

Parameter	Nilai	
LIT^0	2	10
LNR^0	2	10
pi	0.2	0.8
plt	0.2	0.8
pr	0.2	0.8
N	100	
pan	Sesuai nilai batas	
POB	100	

Tabel 2 Data Kasus yang Diuji dan Hasil Pengujian Algoritma

Kasus	Kode Kasus	Jumlah *Job*	Jumlah *Stage*	LB	*Makespan* Terbaik	% Deviasi
1	55HA7	5	5	397	473	19.14
2	510HA8	5	10	678	853	25.81
3	105HA6	10	5	512	630	23.05
4	1010HA5	10	10	956	1078	12.76
5	205HA1	20	5	972	1028	5.76
6	2010HA5	20	10	1436	1817	26.53

Performansi algoritma HFS-VS dibandingkan dengan *Genetic Algorithm* (Serifoglu & Ulusoy, 2004) dan *Artificial Bee Colony* (Kusumo, 2012), masing-masing selanjutnya disebut algoritma HFS-GA dan HFS-ABC. Nilai persentase deviasi *makespan* terhadap LB dari ketiga algoritma tersebut pada setiap kasus dapat dilihat pada Tabel 3. Pada studi komputasional ini, menggunakan jumlah iterasi sebesar 100, sedangkan algoritma HFS-GA dan HFS-ABC masing-masing menggunakan 400 dan 500-750 iterasi.

Dari Tabel 3 dapat dilihat bahwa baik algoritma HFS-VS maupun HFS-GA dan HFS-ABC memiliki nilai persentase deviasi yang selalu positif. Hal ini menunjukkan bahwa ketiga algoritma tidak mampu mencapai nilai LB. Jika dibandingkan dengan algoritma HFS-GA, algoritma HFS-VS yang dihasilkan unggul pada 2 kasus yaitu kasus 1 dan kasus 4. Sementara jika dibandingkan dengan algoritma HFS-ABC, algoritma HFS-VS unggul di 5 kasus yaitu di kasus 2, 3, 4, 5 dan 6. Pada kasus 1 algoritma HFS-VS menghasilkan persentase deviasi yang sama dengan HFS-ABC. Hal ini menunjukkan algoritma HFS-VS yang dihasilkan mampu secara efektif menyelesaikan masalah penjadwalan *Hybrid Flow Shop*.

Tabel 3 Perbandingan Nilai *Makespan* untuk Kasus yang Diuji

Kasus	Kode Kasus	Persentase Deviasi		
		HFS-VS	HFS-GA	HFS-ABC
1	55HA7	19.14	21.64	19.14
2	510HA8	25.81	22.51	43.51
3	105HA6	23.05	9.12	59.96
4	1010HA5	12.76	13.35	26.26
5	205HA1	5.76	1.4	23.66
6	2010HA5	26.53	8.64	64.14

Penelitian ini juga menganalisis pengaruh nilai parameter VS terhadap solusi yang dihasilkan. Dari 6 kasus yang diujikan, parameter yang paling banyak berpengaruh secara signifikan pada semua kasus adalah parameter plt (kecuali pada kasus 2), dimana untuk mendapatkan nilai *makespan* yang baik parameter tersebut harus menggunakan nilai yang relatif tinggi. Parameter yang tidak berpengaruh pada performansi algoritma di semua kasus adalah parameter pi dan LNR0.

KESIMPULAN

Penelitian ini menghasilkan algoritma *Viral Systems* yang digunakan dalam permasalahan penjadwalan HFS dengan *multiprocessor task*. Berdasarkan perhitungan yang dilakukan pada 6 *benchmark problems*, dapat disimpulkan bahwa algoritma yang dihasilkan secara efektif dapat digunakan untuk menyelesaikan masalah penjadwalan HFS dengan *multiprocessor task*. Kualitas solusi yang dihasilkan juga dapat dinilai cukup baik. Dibandingkan dengan *Genetic Algorithm*, algoritma yang dihasilkan mampu unggul di 2 dari 6 kasus. Dibandingkan dengan *Artificial Bee Algorithm*, algoritma yang dihasilkan unggul di 5 kasus dan memiliki performansi yang setara di 1 kasus.

Evaluasi kualitas solusi yang dihasilkan dilakukan dengan menggunakan nilai *lower bound*. Ketersediaan data solusi optimal untuk *benchmark problems* yang digunakan atau penggunaan kasus lain yang telah diselesaikan dengan metode optimasi akan meningkatkan kualitas proses evaluasi algoritma ini. Selain itu untuk penelitian lebih lanjut dapat dipertimbangkan pula penggunaan mesin paralel yang tidak identik dan batasan kapasitas penyimpanan *buffer* antar tahapan proses (*stage*).

DAFTAR PUSTAKA

Amin-Naseri, M.R and M. A. Beheshti-Nia. 2010. “Hybrid flow shop scheduling with parallel batching”. *European Journal of Operational Research,* 205: 1–18.

Bedworth, D.D. and J. E. Bailey. 1987. *Integrated Production Control Systems*, Toronto: John Wiley & Sons.

Choong, F., S. Phon-Amnuaisuk, M.Y. Alias. 2011. “Metaheuristic methods in hybrid flow shop scheduling problem”. *Expert Systems with Applications*. 38: 10787–10793.

Cortes, P., J.M. Garcia, J. Munuzuri, and L. Onieva. 2007. “Viral systems: A new bio-inspired optimization approach”. *Computers & Operations Research*. 35: 2840-2860.

Engin, O. and A. Doyen. 2004. “A new approach to solve hybrid flow shop scheduling problems by artificial immune system”. *Future Generation Computer Systems.* 20: 1083–1095.

Fogarty, D.W., J.H. Blackstone and T.R Hoffman. 1991. *Production & Inventory Management.* Cincinnati: South Western Publishing.

Gen, Mitsuo and R. Cheng. 1997. *Genetic Algorithm & Engineering Design*. Toronto: John Wiley & Sons.

Gupta, J.N.D. 1988. “Two stage, hybrid flowshop scheduling problem”. *Journal of The Operational Research Society*. 39(4): 359-364.

Kusumo, D. 2012. *Penerapan Algoritma Artificial Bee Colony Dalam Menyelesaikan Masalah Penjadwalan Hybrid Flow Shop Untuk Meminimasi Makespan.* Draft Skripsi. Bandung: Jurusan Teknik Industri Universitas Katolik Parahyangan.

Linn, R. and Wei Zhang. 1999. “Hybrid flow shop scheduling: A survey”. *Computers & Industrial Engineering*. 37: 57-61.

Oguz, C., Y. Zinder, V.H. Do, and A. Janiak. 2002. “Hybrid flow-shop scheduling problemswith multiprocessor task systems". *European Journal of Operational Research*. 152: 115-131.

Ruiz, Ruben and J.A.V Rodriguez. 2010. “The hybrid flow shop scheduling problem’. *European Journal of Operational Research*. 205: 1-18.

Serifoglu, F.S and G. Ulusoy. 2004. “Multiprocessor task scheduling in multistage hybrid flow shop: a genetic algorithm approach.” *Journal Of Operational Research Society*. 55 (May 2004): 504-512.

Uetake, T., H. Tsubone and M. Ohba. 1995. “A production scheduling system in a hybrid flow shop”. *International Journal of Production Economics*. 41: 395-398.