Indones. J. Math. Nat. Sci. 44(2) 2021
Indonesian Journal of Mathematics and Natural Sciences
http://journal.unnes.ac.id/nju/index.php/JM

Penyelesaian *Resource Constrained Project Scheduling Problem* (RCPSP) Menggunakan Algoritma Cat Swarm Optimization

Eka Retna Fitriyani*, Isnaini Rosyida, Mashuri

Jurusan Matematika, FMIPA, Universitas Negeri Semarang, Indonesia
Gedung D7 Lt. 1, Kampus Sekaran Gunungpati, Semarang 50229
E-mail: fitriyaniekaretna@gmail.com

Diterima 2 Februari 2021 Disetujui 1 September 2021 Dipublikasikan 31 Oktober 2021

Abstrak

Resource-Constrained Project Scheduling Problem (RCPSP) adalah masalah penjadwalan proyek yang harus memenuhi *Precedence Constraint* dan *Resource constraint.* Tujuan penelitian ini adalah mengetahui penerapan penggunaan Algoritma *Cat Swarm Optimization* (CSO) untuk penjadwalan proyek dan simulasi penjadwalan proyek yang optimal pada beberapa kasus RCPSP. Penelitian ini menggunakan CSO dalam menyelesaikan permasalahan dengan bantuan *software* Matlab. Hasil penelitian menunjukkan bahwa penerapan Algoritma CSO yang diawali dengan membuat solusi RCPSP yang valid dan dihitung *maskepan*nya. Bangkitkan sebanyak N kucing, termasuk kucing dengan solusi valid. Evaluasi kucing sesuai dengan posisi tiap kegiatan. Berdasarkan N kucing yang diperoleh hitung nilai *maskepan*nya. Selanjutnya, pindahkan kucing sesuai dengan MR dalam *seeking mode* dan *tracing mode*. Evaluasi kembali posisi setiap kucing untuk menyimpan kucing dengan nilai *fitness* terkecil. Akhiri algoritma dengan mengambil solusi yang memiliki *fitness* terkecil. Kasus RCPSP dengan solusi valid yang memiliki nilai *maskepan* yang masih jauh dari nilai *maskepan* terkecil yang bisa diperoleh menghasilkan solusi yang berbeda-beda setiap kali program dijalankan sehingga diperlukan beberapa kali simulasi untuk memastikan solusi yang didapatkan benar-benar memiliki *maskepan* terkecil. Bila solusi valid yang dibuat memiliki *maskepan* yang telah mendekati *maskepan* terkecil yang bisa diperoleh, maka program akan menghasilkan solusi yang sama setiap program dijalankan.

Kata kunci: RCPSP, CSO, penjadwalan proyek.

Abstract

Resource-Constrained Project Scheduling Problem (RCPSP) was the problem of project scheduling that must satisfy the Precedence Constraint and Resource Constraint. The objective of this research was to know the application of using Cat Swarm Optimization algorithm for optimal project scheduling and simulation of optimal project scheduling in some RCPSP cases. The method by using Cat Swarm Optimization (CSO) algorithm was to solve the problem with Matlab software's help. The results show that the application of the CSO Algorithm begins with creating a valid RCPSP solution and calculating the mask. Form as much as N cat, included the valid solution cat. Evaluate the cat appropriate with the position in every activity. Count the maskepan value according to N cat which gotten. Then, move the cat appropriate with MR in the seeking mode and tracing mode. Re-evaluate the position of every cat for save the cat with the smallest fitness value. Finish the algorithm with take the solution which had the smallest fitness. 2) RCPSP case with valid solution had maskepan value, that far from the smallest maskepan value, gave different solutions in every run program. In this case needed several times of simulation to make sure that the solution gotten was the correct smallest its maskepan. If the valid solution had maskepan which was closer to the smallest maskepan, then the program will produce the same solution in every run program.

Keywords: RCPSP, CSO, *project scheduling*

How to cite:
Fitriyani, E.R., Rosyida, I., & Mashuri. (2020). Penyelesaian Resource Constrained Project Scheduling Problem (RCPSP) Menggunakan Algoritma Cat Swarm Optimization. *Indonesian Journal of Mathematics and Natural Science, 44(2)*, 48-60

PENDAHULUAN

Proyek merupakan kegiatan yang berlangsung dalam jangka waktu yang terbatas dengan mengalokasikan sumber daya tertentu dan dimaksudkan untuk menghasilkan produk atau *deliverable* yang kriteria mutunya telah digariskan dengan jelas (Soeharto,1999). Sebelum proyek dilaksanakan ada beberapa tahap pengelompokan yang meliputi perencanaan, penjadwalan, dan pengkoordinasian. Dari ketiga tahap tersebut tahap penjadwalan dan perencanaan merupakan tahapan yang menentukan berhasil tidaknya suatu proyek karena penjadwalan adalah tahap ketergantungan antar aktivitas yang membangun proyek secara keseluruhan (Arifudin, 2012). Pemecahan masalah penjadwalan yang baik merupakan salah satu faktor keberhasilan pelaksanaan sebuah proyek tepat pada waktunya yang merupakan tujuan yang penting baik bagi pemilik proyek maupun kontraktor.

Masalah penjadwalan aktivitas-aktivitas pada proyek dengan kendala sumber daya terbatas sering disebut dengan *Resource Constrained Project Scheduling Problem* (RCPSP). Setiap aktivitas memiliki durasi aktivitas dan jumlah kebutuhan sumber daya masing-masing. Penjadwalan proyek dengan ketersediaan sumber daya terbatas adalah penjadwalan proyek yang harus memenuhi urutan pengerjaan kegiatan (*precedence constraint*) dan mempertimbangkan sumber daya yang digunakan pada setiap kegiatan agar tidak melebihi kapasitas sumber daya yang tersedia (*resource constraints*).

Metode analitis algoritma B&B (*Branch and Bound*) pada optimalisasi penjadwalan proyek tidak efisien sehingga berdampak pada keterlambatan penyelesaian proyek pembangunan Mega Tower (Widyawati *et al*., 2014). Penggunaan metode metaheuristik untuk menyelesaikan RCPSP seperti algoritma *Harmony Search* (Imansyah, 2013), algoritma *Cross Entropy* (Krisnawati, 2014), algoritma Genetika (Chan *et al*., 1996), dan algoritma *Particle Swarm Optimization* (Zhang *et al*., 2006). Algoritma *Harmony Search* harus dilakukan dengan tujuh langkah yaitu inisialisasi masalah, memasukan data RCPSP, inisialisasi parameter algoritma *Harmony Search*, inisialisasi *harmony memory*, membangkitkan vektor solusi baru, meng-update *harmony memory*, dan mengecek criteria pemberhentian. Dibandingkan dengan Algoritma CPM, solusi algoritma *Harmony Search* tidak selalu merupakan solusi terbaik, karena pencarian solusi dilakukan secara random, sehingga solusi yang diperoleh sangat beragam dan memerlukan waktu lebih lama yang bergantung pada kriteria pemberhentian yang dipilih. Algoritma *Cross Entropy* melibatkan prosedur iterasi, dimana tiap iterasi dapat dipecahkan menjadi dua fase yaitu pembangkitan sampel random dan pembaharuan parameter. Pengunaan jumlah aktivitas proyek 30-60 memperlihatkan bahwa algoritma *Cross Entropy* mempunyai performasi yang sama dengan penyelesaian menggunakan algoritma *Differential Evolution*. Algoritma Genetika melakukan pencarian solusi yang optimal dari kromosom-kromosom yang merepresentasikan jadwal dimana kromosom-kromosom tersebut diproduksi melalui *cross-over* dan *mutation*. Mekanisme *updating* kromosom membuat algoritma Genetika mampu keluar dari solusi yang bersifat lokal optimal. Oleh karena itu, algoritma Genetika lebih unggul dari pada metode analitis dan metode heuristik. Namun demikian, terdapat kekurangan dalam performansi Algoritma Genetika yaitu proses konvergensi yang lambat. Performansi *Particle Swarm Optimization* menunjukan kemampuan untuk mencari optimum global dan lebih efisien dari pada Algoritma Genetika karena fiturnya (Zhang *et al*., 2006).

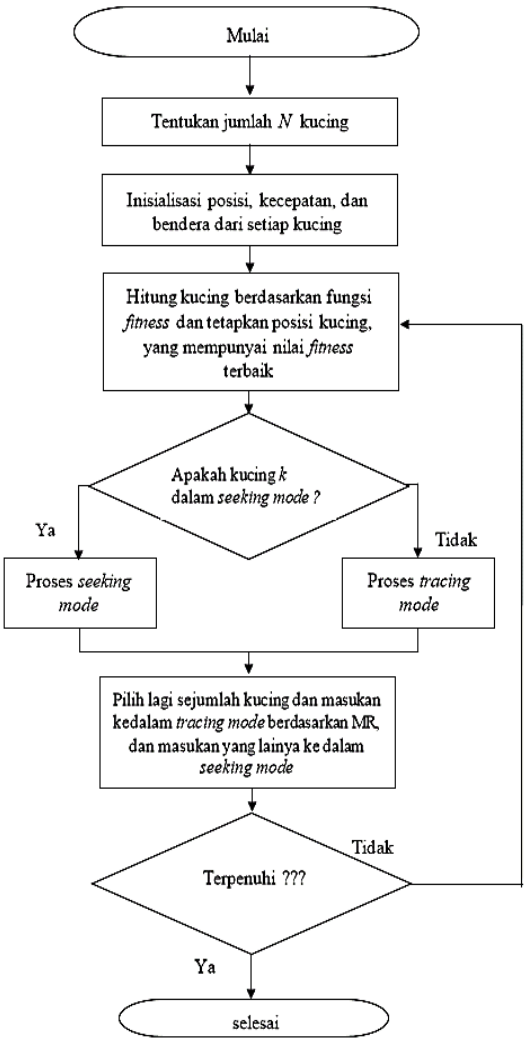
Dari enam tes yang dilakukan oleh Chu dan Tsai (2006), menunjukan kinerja Algoritma *Cat Swarm Optimization* lebih efektif dibandingkan dengan Algoritma *Particle Swarm Optimization* dan Algoritma *Particle Swarm Optimization with weighting.* Algoritma *Cat Swarm Optimization* (CSO) dapat menghasilkan klasifikasi yang lebih baik dalam hal jumlah iterasi yang dibutuhkan untuk mencapai titik optimal dan memiliki tingkat akurasi yang lebih baik (Dhanasaputra & Santosa, 2010).

CSO merupakan salah satu algoritma metaheruistik untuk masalah optimasi kobinatorial. Algoritma CSO pertama kali diperkenalkan oleh Shu-Chuan Chu dan Pei-Wei Tsai (Taiwan) pada tahun 2006 (Chu & Tsai, 2006). Algoritma CSO terbagi menjadi 2 langkah dalam menyelesaikan masalah optimasi, yaitu *Seeking Mode* (SM) yang menggambarkan kucing saat istirahat, melihat sekeliling, menyusun strategi selanjutnya dan *Tracing Mode* (TM) yang menggambarkan kucing saat mengikuti mangsa buruan. Dua sub model *Seeking Mode* dan *Tracing Mode* dikombinasikan dalam satu algoritma melalui *Mixture Ratio* (MR). Tujuan penelitian ini adalah mengetahui penerapan penggunaan algoritma CSO untuk penjadwalan proyek dan mengetahui susunan penjadwalan yang optimal pada beberapa kasus RCPSP.

METODE

Penelitian ini menggunakan metode penelitian studi pustaka untuk mempelajari teori-teori yang berhubungan dengan algoritma CSO untuk menyelesaikan masalah yang dikaji. *Software* yang digunakan adalah Matlab. Data yang digunakan dalam penelitian ini adalah data sekunder yang diperoleh dari Merkel *et al*. (2002) dan Zhang *et al.* (2006). Data 1 terdiri dari 6 kegiatan dengan kapasitas sumber daya 4 unit, data 2 terdiri dari 7 kegiatan dengan kapasitas sumber daya 5 unit, dan data 3 terdiri dari 25 kegiatan 3 jenis sumber daya dengan kapasitas sumber daya 6 unit.

Langkah-langkah Algoritma CSO diberikan pada Gambar 1.



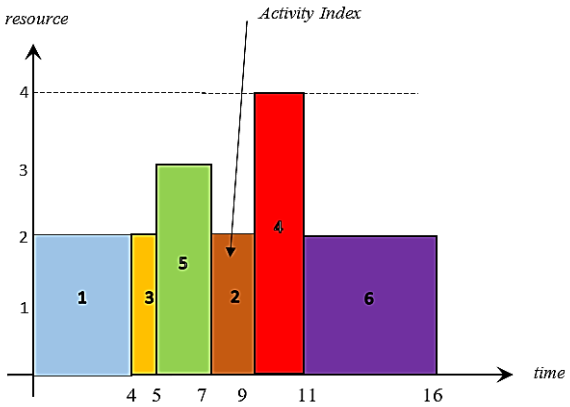
Gambar 1. Flowchart Algoritma

HASIL DAN PEMBAHASAN

Pembuatan solusi RCPSP yang valid merupakan langkah awal sebelum menerapkan Algoritma CSO pada masalah RCPSP. Solusi RCPSP dikatakan valid apabila jadwal yang dihasilkan memenuhi *precedence constraint* dan *resource constraint* yang telah ditetapkan. Setiap solusi RCPSP yang valid memiliki waktu penyelesaian proyek (*maskepan*). Solusi RCPSP awal yang valid diterapkan pada Algoritma CSO sebagai salah satu kucing, tujuanya untuk mendapatkan solusi yang lebih optimal dengan *maskepan* minimum. Solusi valid pada kasus RCPSP ini diperoleh solusi valid yang paling sederhana dengan mengurutkan kegiatan yang ada pada gambar visualisasi kasus tersebut. Terdapat beberapa kemungkinan solusi valid pada data ke-1, yaitu $1-3-5-2-4-6$, $1-2-3-4-5-6$, $1-3-2-4-5-6$, dan $2-4-6-1-3-5$. Perhitungan manual menggunakan solusi valid $1-3-5-2-4-6$, solusi valid yang lain dicoba pada program simulasi. *Maskepan* dari solusi valid data ke-1 adalah 16 satuan waktu seperti pada Gambar 2.

Tahap penting dari algoritma CSO yaitu menentukan berapa banyak kucing yang digunakan dalam menyelesaikan masalah. Setiap kucing tersebut mewakili himpunan solusi acak awal, yaitu urutan acak dari kegiatan J_1 sampai dengan kegiatan J_{n+1}. Solusi acak awal adalah solusi yang dibuat secara sembarang tanpa mempertimbangkan batasan yang telah ditetapkan, sehingga kucing yang dibuat dapat berupa solusi yang valid maupun yang tidak valid (Apriana *et al*., 2016).

Banyaknya kucing yang digunakan dalam data ke-1 adalah 5 tanpa iterasi dalam perhitugan manual. Dimensi merupakan penggambaran dari setiap kegiatan. Kucing data ke-1 memiliki susunan posisi 6 dimensi. Inisialisasi posisi dari setiap kucing dilakukan secara random. Sedangkan kecepatan kucing pada inisialisasi awal = 0 untuk setiap dimensi. Inisialisasi ruang kucing juga dilakukan secara random, sehingga kucing bebas akan masuk ke *seeking mode* atau *tracing mode*. Hanya jumlah kucing yang ditentukan dalam memasuki tiap *mode*.



Gambar 2. Visualisasi Bentuk Jadwal Solusi Valid Data ke-1

Setiap kucing akan dievaluasi berdasarkan posisi tiap kegiatan, dimana setiap posisi harus memenuhi urutan pengerjaan kegiatan dan mempertimbangkan sumber daya yang digunakan agar tidak melebihi kapasitas sumber daya yang tersedia. Kucing yang memenuhi setiap batasan dapat dihitung nilai *fitness*nya dan begitu sebaliknya. Evaluasi kucing berguna untuk mempercepat perhitungan nilai *fitness.* Jika ada kucing yang melanggar 1 batasan saja maka kucing tidak dapat dihitung nilai *fitness*nya

Lebih jelas penerapan algoritma CSO dalam *Resource Constrained Project Scheduling Problem,* berikut perhitungan manual dengan data kasus yang kecil, karena apabila menggunakan data yang cukup besar perhitunganya akan rumit dan sangat panjang. Pada penerapan ini data yang digunakan adalah data ke-1 dan data ke-2.

Langkah 1 : Membuat solusi RCPSP awal yang valid. Langkah ini dalam perhitungan manual dapat dihilangkan karena umumnya dapat diketahui secara langsung apabila solusi yang dibuat valid atau tidak. Berbeda bila program yang menjalankan karena inisialisasi posisinya random kemungkinan program dapat membangkitkan semua kucing dengan solusi yang tidak valid. Solusi validnya diperoleh $1-3-5-2-4-6$.

Langkah 2 : Menghitung nilai *fitness* dari solusi valid. Diperoleh nilai *fitness*nya adalah 16 satuan waktu.

Langkah 3 : Membangkitkan sejumlah N kucing dalam proses. Setiap kucing mewakili himpunan solusi acak awal, yaitu urutan acak dari aktivitas J_1 sampai dengan aktivitas J_{n+1}. Pada kasus ini akan dibangkitkan 5 kucing termasuk kucing dengan solusi valid. Diperoleh kucing sebagai berikut:

i. Kucing ke-1: $1-3-5-2-4-6$
ii. Kucing ke-2: $1-3-2-5-4-6$
iii. Kucing ke-3: $2-4-1-3-5-6$
iv. Kucing ke-4: $1-2-4-3-5-6$
v. Kucing ke-5: $1-3-4-2-5-6$

Langkah 4 : Inisialisasi posisi, kecepatan, dan ruang kucing. Kecepatan awal = 0.

Langkah 5 : Evaluasi semua kucing sesuai dengan posisi setiap kegiatan. Kucing ke-1,2,3, dan 4 memenuhi *precedence constraint* sedangkan kucing k-5 tidak memenuhi artinya kucing ke-5 merupakan solusi yang tidak valid.

Langkah 6 : Hitung semua nilai *fitness* dari N kucing, dan didapat satu kucing dengan nilai *fitness* terbaik.

i. Kucing ke-1 : $1-3-5-2-4-6$ nilai fitness kucing adalah 16
   Menghitung nilai fitness dapat dengan mudah diperoleh menggunakan Gambar 1.
ii. Kucing ke-2 : $1-3-2-5-4-6$ nilai *fitness* kucing adalah 15
iii. Kucing ke-3: $2-4-1-3-5-6$ nilai *fitness* kucing adalah 16
iv. Kucing ke-4 : $1-2-4-3-5-6$ nilai *fitness* kucing adalah 14
v. Kucing ke-5: $(1-3-4-2-5-6)$ jika kegiatan 4 mendahului kegiatan 2 maka kegiatan 4 tidak dapat terlaksana karena kegiatan 2 belum dikerjakan sehingga tidak dapat dihitung nilai *fitness*nya.

Langkah 7 : Pindahkan kucing sesuai MR (*mixture ratio).* Jika kucing berada dalam *seeking mode* perlakukan kucing sesuai *seeking mode*, dan begitu juga sebaliknya. Nilai MR dan parameter lain seperti SRD, CDC, C diambil dari kombinasi penelitian Apriana et al. (2016) dan Chu & Tsai (2006) sedangkan nilai parameter SMP diambil 2 agar perhitungannya lebih cepat dan dapat melakukan proses *seeking* dengan baik. Pengambilan parameter tanpa melakukan pengujian parameter terlebih dahulu seperti pada Tabel 1.

Tabel 1. Kombinasi Parameter yang Digunakan untuk Perhitungan Manual

Nama Parameter	Nilai
MR	0.5
SMP	2
SRD	0.2/20%
CDC	0.6/60%
C1	2
R	[0,1] random $0-1$
W	[0.4,0.9] random $0.4-0.9$

JT $= MR \times$ JK
$= 0.5 \times 5 = 2.5$ dibulatkan ke bawah menjadi 2 karena jumlah *tracing* harus lebih kecil dari pada jumlah *seeking*
JS = JK − JT
$= 5 - 2 = 3$

Langkah 8 : Pilih sejumlah kucing dan masukan kedalam *tracing mode* sesuai MR dan sisanya masukan ke dalam *seeking mode*.

8.1. Misalkan kucing ke-2 dan 3 masuk ke *tracing mode*
  8.1.1. Proses *tracing* kucing ke-2 $(1-3-2-5-4-6)$
    8.1.1.1. Langkah *tracing* 1: Perbarui nilai kecepatan untuk setiap dimensi $(V_{k,d})$ berdasarkan persamaan $V'_{k,d} = w \times V_{k,d} + r_1 \times C_1(X_{best,d} - X_{k,d})$ dimana $C1 = 2$, $r1 = [0,1]$ random dari $0-1$, $w = [0.4,0.9]$ random dari $0.4-0.9$, dan kucing terbaik adalah kucing ke-4 yaitu $1-2-4-3-5-6$ karena memiliki nilai *fitness* terkecil, diperoleh kecepatan terbaru sebagai berikut :
    Kecepatan dimensi ke-1
    $V'_{k,d} = w \times V_{k,d} + r_1 \times C_1(X_{best,d} - X_{k,d})$
    $V'_{3,1} = 0.4 \times 0 + 0.8 \times 2\,(1-1)$
    $= 0 + 0.8 \times 2\,(0) = 0$
    Kecepatan dimensi ke-2
    $V'_{k,d} = w \times V_{k,d} + r_1 \times C_1(X_{best,d} - X_{k,d})$
    $V'_{3,2} = 0.5 \times 0 + 0.5 \times 2\,(2-3)$
    $= 0 + 0.5 \times 2\,(-1) = -1$
    Kecepatan dimensi ke-3

$V'_{k,d} = w \times V_{k,d} + r_1 \times C_1(X_{best,d} - X_{k,d})$
$V'_{3,3} = 0.6 \times 0 + 0.23 \times 2\ (4 - 2)$
$= 0{,}92$ dibulatkan menjadi 1

Kecepatan dimensi ke-4
$V'_{k,d} = w \times V_{k,d} + r_1 \times C_1(X_{best,d} - X_{k,d})$
$V'_{3,4} = 0.7 \times 0 + 0.25 \times 2\ (3 - 5)$
$= 0 + 0.25 \times 2\ (-2) = -1$

Kecepatan dimensi ke-5
$V'_{k,d} = w \times V_{k,d} + r_1 \times C_1(X_{best,d} - X_{k,d})$
$V'_{3,5} = 0.8 \times 0 + 37 \times 2\ (5 - 4)$
$= 0.74$ dibulatkan menjadi 1

Kecepatan dimensi ke-6
$V'_{k,d} = w \times V_{k,d} + r_1 \times C_1(X_{best,d} - X_{k,d})$
$V'_{3,6} = 0.9 \times 0 + 0.77 \times 2\ (6 - 6)$
$= 0 + 0.77 \times 2\ (0) = 0$

8.1.1.2. Langkah *tracing* 2: Periksa apakah kecepatan berada dalam rentang kecepatan maksimum. Jika kecepatan yang baru melebihi rentang, tetapkan nilai sama dengan batas. Tidak ada kecepatan baru yang melebihi rentang kecepatan.

8.1.1.3. Langkah *tracing* 3: Perbarui posisi kucing ke-k dengan persamaan $X'_{k,d} = X_{k,d} + V'_{k,d}$.

Perbaharuan posisi :
Dimensi ke-1 $X'_{3,1} = 1 + 0 = 1$
Dimensi ke-2 $X'_{3,2} = 3 - 1 = 2$
Dimensi ke-3 $X'_{3,3} = 2 + 1 = 3$
Dimensi ke-4 $X'_{3,4} = 5 - 1 = 4$
Dimensi ke-5 $X'_{3,5} = 4 + 1 = 5$
Dimensi ke-6 $X'_{3,6} = 6 + 0 = 6$

Diperoleh posisi kucing terbaru $1 - 2 - 3 - 4 - 5 - 6$ dengan nilai *fitness 14.*

8.1.2. Proses *tracing* kucing ke-3 $(2 - 4 - 1 - 3 - 5 - 6)$

Dengan proses yang sama pada proses *tracing* kucing ke -2 diperoleh posisi kucing baru yaitu $1 - 3 - 2 - 3 - 5 - 6$, karena kegiatan 3 berlangsung 2 kali maka nilai *fitness* tidak dapat dihitung.

8.2. Misalkan kucing ke 1,4 dan ke-5 masuk ke*seeking mode.*

8.2.1. Proses *seeking* kucing ke-1 $(1 - 3 - 5 - 2 - 4 - 6)$

8.2.1.1. Langkah *seeking* 1: Bangkitkan j tiruan dari posisi kucing ke-k, dengan $j = SMP$. Jika nilai SPC benar, maka $j = (SMP - 1)$, kemudian pertahankan posisi saat ini sebagai salah satu kandidat.

Pada kucing ke-1 membangkitkan kucing tiruan sebanyak 2 ($SMP = 2$) dan SPC benilai benar maka $j = 2 - 1 = 1$.

8.2.1.2. Langkah *seeking* 2 : Untuk setiap tiruan, disesuaikan dengan CDC, tambahkan atau kurangkan SRD persen dari nilai saat ini secara acak dan gantikan nilai yang sebelumnya.

Jumlah modifikasi $= 0.6 \times 6$
$= 3.6$ (4 dimensi)

Yang akan dimodifikasi dimensi ke-1, 3, 4 dan dimensi ke-5. Posisi kucing menjadi $2 - 3 - 4 - 3 - 5 - 6$.

8.2.1.3. Langkah *seeking* 3 : Hitung nilai *fitness* (FS) untuk semua titik kandidat. *Fitness* merupakan nilai *maskepan* dari setiap kucing.

Nilai *fitness* kucing tiruan tidak dapat dihitung karena kegiatan 3 terlaksana dua kali sedangkan kegiatan 1 tidak terlaksana. Sehingga kucing yang dipilih kucing ke-1.

8.2.2. Proses *seeking* kucing ke-4 $(1 - 2 - 4 - 3 - 5 - 6)$

Dengan proses yang sama pada proses *seeking* kucing ke-1 diperoleh posisi kucing tiruan menjadi $2 - 1 - 4 - 3 - 6 - 5$. Nilai *fitness* kucing tiruanya 13 satuan waktu.

Peluang terpilihnya kucing ke-1

$$P_h = \frac{|F\ S_h - F\ S_b|}{F\ S_{max} - F\ S_{min}} = \frac{|14 - 14|}{14 - 13} = \frac{0}{1}$$

Peluang terpilihnya kucing tiruan

$$P_h = \frac{|F\ S_h - F\ S_b|}{F\ S_{max} - F\ S_{min}} = \frac{|13 - 14|}{14 - 13} = 1$$

Pilih kucing dengan nilai peluang = 1, terpilih kucing tiruan dari kucing ke-1 dengan nilai *fitness* 13 satuan waktu.

8.2.3. Proses *seeking* kucing ke-5 $(1-3-4-2-5-6)$

Dengan proses yang sama pada proses *seeking* kucing ke-1 diperoleh posisi kucing tiruan ke-1 menjadi $1-2-4-3-6-5$ dan posisi kucing tiruan ke-2 menjadi $2-3-5-1-4-6$. Nilai *fitness* kucing tiruan ke-1 dari kucing ke 5 adalah 13, sedangkan nilai *fitness* kucing tiruan ke-2 tidak dapat dihitung karena kegiatan 3 dan 5 mendahului kegiatan 1 sehingga kegiatan 3 dan 5 tidak dapat terlaksana. Dipilih kucing tiruan ke-1 dari kucing ke-5 karena kucing ke-5 dan kucing tiruan ke-2 tidak ada nilai *fitness*nya.

Langkah 9 : Evaluasi kembali posisi kegiatan dari semua kucing. Dilihat apakah semua kucing dan kucing tiruanya sesuai dengan *precedence* dan *resource constraint.* Evaluasi ini berguna untuk menyimpan kucing dan kucing tiruan yang dapat dihitung nilai *fitness*nya.

Langkah 10: Perhatikan kondisi akhir algoritma. Jika memuaskan (*fitness* bernilai kecil), maka hentikan program. Apabila belum ulangi langkah 6 sampai langkah 9. Pada perhitungan manual tidak dicoba dengan melakukan iterasi untuk nilai random [0,1] lainya karena akan sangat rumit.

Langkah 11: Akhir pada algoritma yang digunakan berupa *fitness* terkecil, kucing yang terpilih sebagai solusi terbaik adalah kucing yang dapat memenuhi semua batasan yang ada dengan *fitness* terkecil. Terpilih kucing dari proses *seeking* kucing tiruan dari kucing ke-4 $(2-1-4-3-6-5)$ dan kucing tiruan ke-1 dari kucing 5 dengan nilai *finess* yang sama yaitu 13 satuan waktu.

Penerapan Algoritma Cat Swarm Optimization pada data ke-2 menghasilkan *maskepan* terkecilnya adalah 15 satuan waktu yang sama dengan solusi valid yang dibuat. Perhitungan manual data ke-2 dilakukan dengan kombinasi parameter yang sama pada perhitungan manual data ke-1. Solusinya adalah $2-1-3-5-4-6-7$ pada kucing dari proses *tracing* kucing ke-4 dan 5, kucing ke-1, dan kucing tiruan dari kucing ke-1.

Simulasi Penjadwalan Proyek yang Optimal Pada Beberapa Kasus RCPSP Menggunakan *Software* Matlab

Solusi valid dari data ke-2 yaitu $1-2-3-4-5-6-7$, sedangkan data ke-3 yang dipisahkan menjadi tiga jenis data sama yaitu $1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25$. *Maskepan* dari data ke-2 adalah 15 satuan waktu. *Maskepan* dari data 3.1 adalah 63 satuan waktu, *maskepan* dari data 3.2 adalah 63 satuan waktu, dan *maskepan* dari data 3.3 adalah 52 satuan waktu.

Pada data ke-1, 2 dan ke-3 simulasi program digunakan 30 dan 50 kucing dengan masing-masing kucing beriterasi sebanyak 10 dan 20. Tujuan untuk mengetahui apakah ada perubahan jumlah kucing dan iterasi dapat mempengaruhi hasil dari Algoritma Cat Swarm Optimization dalam menyelesaikan kasus RCPSP.

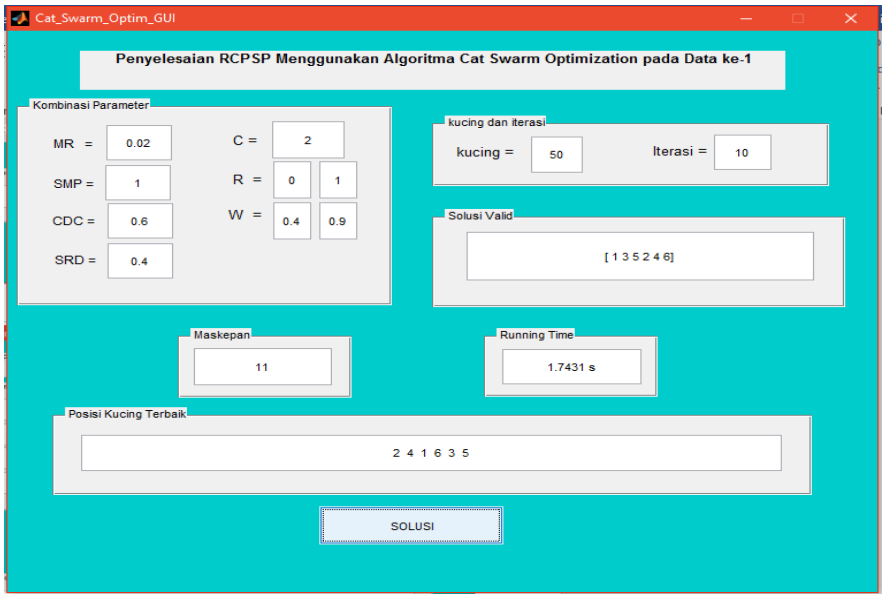
Data ke-2 memiliki susunan posisi 7 dimensi sedangkan data ke-3 memiliki susunan posisi 25 dimensi yang berisikan kegiatan dari kegiatan awal sampai akhir. Kecepatan kucing pada inisialisasi awal = 0 untuk setiap dimensi. Setiap kucing yang dibangkitkan dihitung nilai *fitness*nya. Perhitungan nilai *fitness* ini bertujuan untuk mengetahui kucing terbaik dari semua kucing yang dibangkitkan. Kucing terbaik akan digunakan pada perhitungan *tracing mode.*

Parameter terbaik dengan rata-rata *maskepan* yang terkecil yaitu 12.6 kombinasi parameter yang tepat diperoleh pada Tabel 2.

Tabel 2 Hasil Kombinasi Parameter Terbaik untuk Simulasi

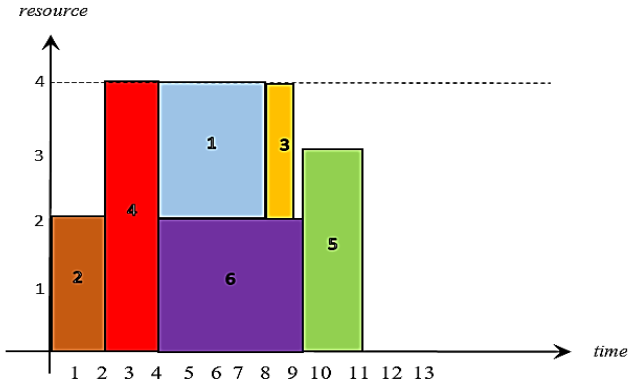
Nama Parameter	Nilai
MR	0.02
SMP	1
SRD	40%/0.4
CDC	60%/0.6
C1	2
R	[0,1] random $0-1$
W	[0.4,0.9] random $0.4-0.9$

Hasil simulasi algoritma CSO pada permasalahan RCPSP dari data ke-1 dengan solusi valid 1 – 3 – 5 – 2 – 4 – 6 menggunakan kombinasi parameter yang tepat dan 2 kali simulasi ternyata dapat diperoleh nilai *maskepan* yang lebih kecil yaitu 11 satuan waktu dengan *running time* 1.7431 s pada simulasi 1 seperti Gambar 3 dengan rangkuman menyeluruh pada Tabel 3 dan 4, sehingga dapat disimpulkan bahwa program yang dibuat valid dan dapat diimplementasikan ke kasus RCPSP yang lebih besar.



Gambar 3. Hasil Simulasi Data 1 pada Program

Set solusi dengan *maskepan* terbaik dari program disajikan pada Gambar 4. Hasil simulasi 1 pada data ke-1 pada Tabel 3 dan hasil simulasi 2 pada data ke-1 pada Tabel 4.



Gambar 4. Visualisasi Bentuk Jadwal Hasil Simulasi Data 1

Tabel 3. Hasil Simulasi 1 Algoritma CSO pada Data ke-1

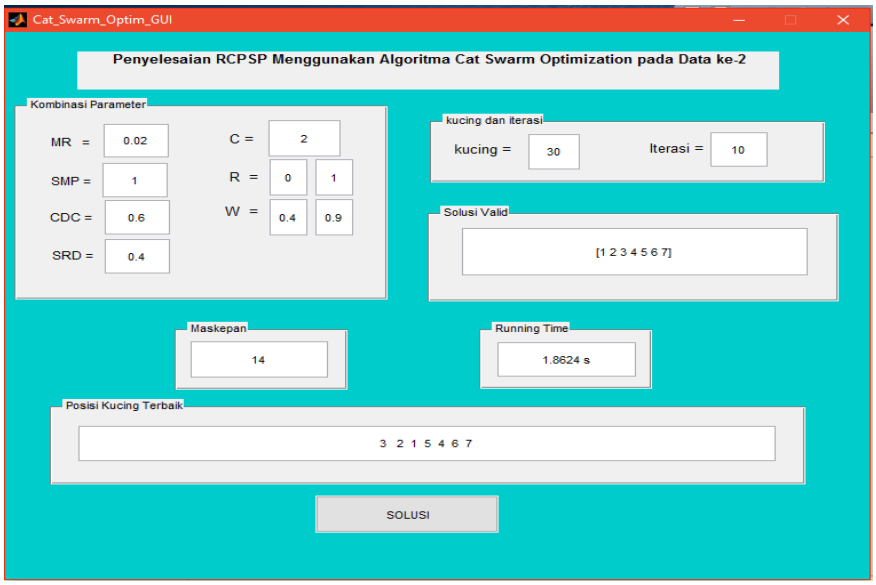
Jumlah Kucing	Iterasi	*Maskepan*	*Running Time*	Posisi kucing
N kucing = 30	10	16	1.0548 s	1-3-5-2-4-6
	20	14	2.0041 s	1-2-3-4-5-6
N kucing =50	10	11	1.7431 s	2-4-1-6-3-5
	20	12	2.8931 s	2-4-1-6-3-5

Tabel 4 . Hasil Simulasi 2 Algoritma CSO pada Data ke-1

Jumlah Kucing	Iterasi	*Maskepan*	*Running Time*	Posisi kucing
N kucing = 30	10	15	1.1071 s	1-3-2-4-6-5
	20	14	1.9937 s	1-2-3-4-5-6
N kucing =50	10	14	1.5527 s	1-2-3-4-5-6
	20	11	3.0872 s	2-4-6-1-3-5

Simulasi program dengan solusi valid yang lain menghasilkan hasil yang sama yaitu $2-4-1-6-3-5$ dengan *maskepan* 11 satuan waktu. Penggunaan solusi valid yang berbeda hanya berpengaruh pada banyak sedikitnya simulasi yang menghasilkan *maskepan* terkecil.

Dari data ke-2 apabila dilakukan perhitungan menggunakan program dengan kombinasi parameter yang tepat dan 2 kali simulasi ternyata dapat diperoleh nilai *maskepan* yang lebih kecil yaitu 14 satuan waktu dengan *running time* 1.8624 s pada simulasi 2 seperti pada Gambar 5 dengan rangkuman menyeluruh pada Tabel 5 dan Tabel 6.



Gambar 5 Hasil. Simulasi Data 2 pada Program

Simulasi dilakukan 2 kali dengan banyaknya kucing 30, dan 50 dengan masing-masing beriterasi sebanyak 10 dan 20.

Tabel 5. Hasil Simulasi 1 Algoritma CSO pada Data ke-2

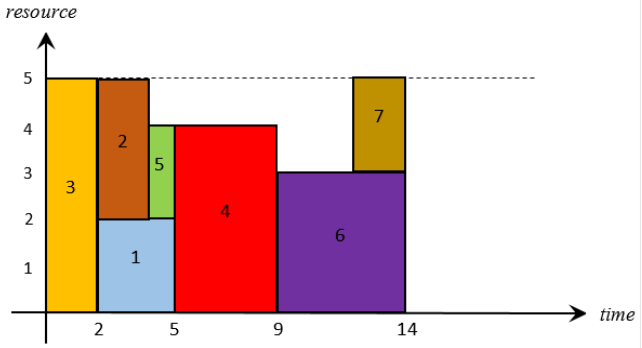
Jumlah Kucing	Iterasi	*Maskepan*	*Running Time*	Posisi kucing
N kucing = 30	10	14	1.8935 s	2-1-5-4-3-6-7
	20	14	3.6837 s	3-2-1-5-4-6-7
N kucing =50	10	14	2.9383 s	2-1-5-3-4-6-7
	20	14	4.8995 s	2-1-5-3-4-6-7

Tabel 6. Hasil Simulasi 2 Algoritma CSO pada Data ke-2

Jumlah Kucing	Iterasi	*Maskepan*	*Running Time*	Posisi kucing
N kucing = 30	10	14	1.8624 s	3-2-1-5-4-6-7
	20	14	3.4141 s	2-1-5-4-3-6-7
N kucing =50	10	14	2.4777 s	2-1-5-4-3-6-7
	20	14	5.4535 s	2-1-5-4-3-6-7

Dari Tabel 5 dan Tabel 6 terlihat bahwa simulasi dengan jumlah kucing dan jumlah iterasi yang berbeda menghasilkan *maskepan* yang sama, sedangkan dari *running time* terlihat bahwa semakin bertambah jumlah kucing dan jumlah iterasi yang digunakan semakin lama *running time* yang

diperlukan. Pemilihan dilakukan dengan memilih *running time* terkecil yaitu pada pembangkitan 30 kucing dengan iterasi 10 disimulasi 2 karena *maskepan* yang dihasilkan sama kecilnya. *Maskepan* dari hasil simulasi adalah 14 satuan waktu dengan solusi penjadwalan $3-2-1-5-4-6-7$. Set solusi dengan *maskepan* terbaik dari program disajikan pada Gambar 6.



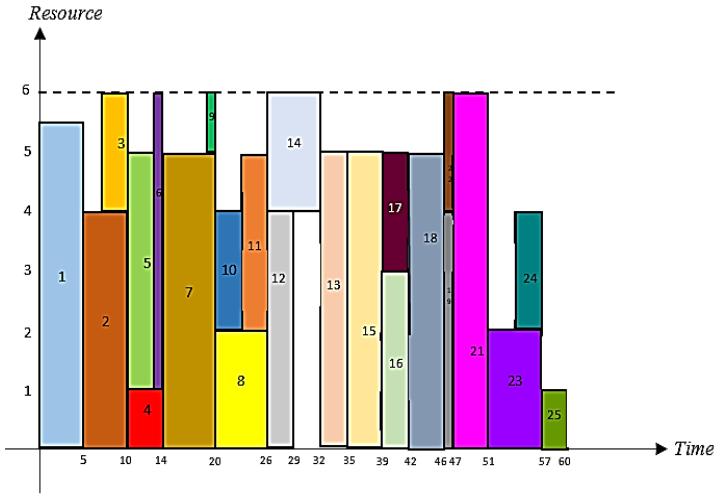
Gambar 6. Visualisasi Bentuk Jadwal Hasil Simulasi Data Ke-2.

Kombinasi parameter yang tepat juga diterapkan pada algoritma CSO data ke-3. Simulasi dilakukan hanya sekali dengan banyaknya kucing 30, dan 50 dengan masing-masing kucing beriterasi sebanyak 10 dan 20. Hasil simulasi dari tiap data dari data 3 dapat dilihat pada Tabel 7, Tabel 8, dan Tabel 9.

Tabel 7 Hasil Simulasi Algoritma CSO pada Data ke-3.1

Jumlah Kucing	Iterasi	*Maskepan*	*Running Time*	Posisi kucing
N kucing = 30	10	60	61.3712 s	1-2-3-4-5-6-7-9-8-10-11-12-14-13-15-16-17-18-19-22-21-23-24-25
	20	60	108.1825 s	1-2-3-4-5-6-7-9-8-10-11-12-14-13-15-16-17-18-19-22-21-23-24-25
N kucing =50	10	60	97.6568 s	1-2-3-4-5-6-7-9-8-10-11-12-14-13-15-16-17-18-19-22-21-23-24-25
	20	60	171.7426 s	1-2-3-4-5-6-7-9-8-10-11-12-14-13-15-16-17-18-19-22-21-23-24-25

Dari Tabel 7 terlihat bahwa simulasi dengan jumlah kucing dan jumlah iterasi yang berbeda menghasilkan *maskepan* yang sama, sedangkan dari *running time* terlihat bahwa semakin bertambah jumlah kucing dan jumlah iterasi yang digunakan semakin lama *running time* yang diperlukan. Pemilihan dilakukan dengan memilih *running time* terkecil yaitu pada pembangkitan 30 kucing dengan iterasi 10 karena *maskepan* yang dihasilkan sama kecilnya. *Maskepan* dari hasil simulasi adalah 60 satuan waktu dengan solusi penjadwalan $1-2-3-4-5-6-7-9-8-10-11-12-14-13-15-16-17-18-19-22-21-23-24-25$, seperti diperlihatkan pada visualisasi RCPSP di Gambar 6.

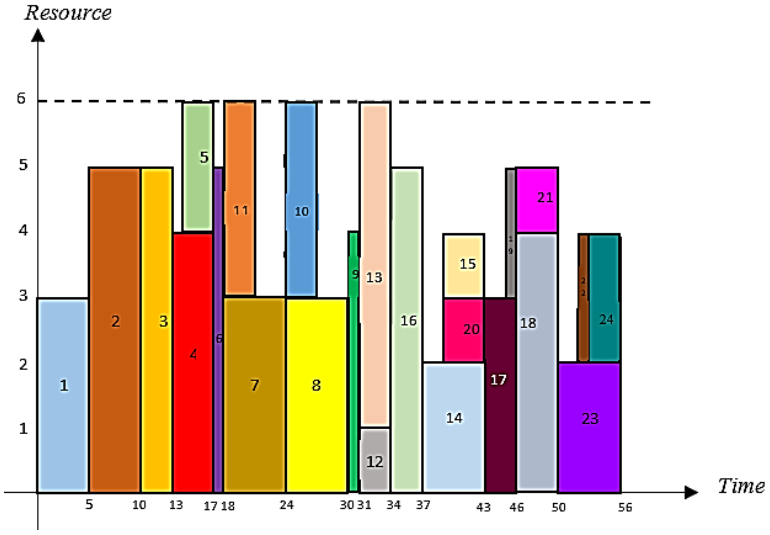


Gambar 6. Visualisasi Bentuk Jadwal Hasil Simulasi Data Ke-3.1.

Tabel 8. Hasil Simulasi Algoritma CSO pada Data ke-3.2

Jumlah Kucing	Iterasi	*Maskepan*	*Running Time*	Posisi kucing
N kucing = 30	10	56	82.4107 s	1-2-3-4-5-6-7-11-8-10-9-12-13-16-14-20-15-17-19-18-21-22-23-24
	20	56	163.1321 s	1-2-3-4-5-6-7-11-8-10-9-12-13-16-14-20-15-17-19-18-21-22-23-24
N kucing =50	10	56	119.2665 s	1-2-3-4-5-6-7-11-8-10-9-12-13-16-14-20-15-17-19-18-21-22-23-24
	20	56	221.3662 s	1-2-3-4-5-6-7-11-8-10-9-12-13-16-14-20-15-17-19-18-21-22-23-24

Dari Tabel 8 terlihat *maskepan* hasil simulasi adalah 56 satuan waktu dengan solusi penjadwalan $1-2-3-4-5-6-7-11-8-10-9-12-13-16-14-20-15-17-19-18-21-22-23-24$, seperti diperlihatkan pada visualisasi RCPSP di Gambar 7.

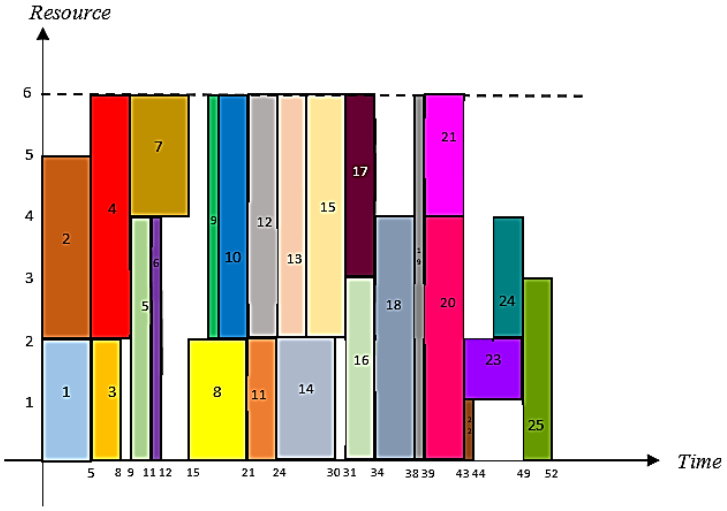


Gambar 7. Visualisasi RCPSP Hasil Simulasi Data Ke-3.2

Tabel 9. Hasil Simulasi Algoritma CSO pada Data ke-3.3

Jumlah Kucing	Iterasi	*Maskepan*	*Running Time*	Posisi kucing
N kucing = 30	10	52	31.2025 s	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25
	20	52	67.4615 s	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25
N kucing =50	10	52	69.6197 s	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25
	20	52	133.4227 s	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25

Dari Tabel 9 terlihat *maskepan* dari hasil simulasi adalah 52 satuan waktu dengan solusi penjadwalan $1-2-3-4-5-6-7-9-8-10-11-12-14-13-15-16-17-18-19-20-22-22-23-24-25$, seperti diperlihatkan pada visualisasi RCPSP di Gambar 8. Solusi penjadwalan yang dihasilkan sama dengan solusi valid yang dibangun, artinya solusi valid yang dibuat sudah merupakan solusi yang optimal.



Gambar 8. Visualisasi RCPSP Hasil Simulasi Data Ke-3.3

Pada data ke-1 setiap dijalankan simulasi dengan menggunakan *software* Matlab diperoleh hasil yang berbeda-beda sedangkan pada data ke-2 dan ke-3 menghasilkan *maskepan* yang sama hanya *running time*nya yang berbeda. Hal tersebut terjadi karena pada data ke-1 memiliki solusi valid yang *maskepan*nya masih jauh dari *maskepan* terkecil yang bisa didapatkan, sedangkan data ke-2 dan ke-3 *maskepan* solusi valid yang dibuat ternyata sudah mendekati *maskepan* terkecil yang bisa didapatkan. Bentuk visualisasi RCPSP juga berpengaruh akan hasil yang akan didapatkan. Banyak sedikitnya batasan *precedence constraint* dapat mempengaruhi solusi valid yang bisa dibuat. Apabila solusi valid yang bisa dibuat banyak maka solusi yang dihasilkan dapat bermacam-macam, begitupun sebaliknya. Pada data ke-1 memiliki bentuk *serial stucture* sehingga kendala *precedence constraint*nya sedikit sehingga hasil yang diperoleh bisa bermacam-macam. Pada data ke-2 memiliki bentuk visualisasi semi *serial structure* namun karena *maskepan* solusi valid yang dibuat sudah mendekati *maskepan* terkecil yang bisa diperoleh maka hasil tiap simulasinya sama. Pada data ke-3 bentuk visualisasinya *general structure* sehingga hasil yang diperoleh semuanya sama.

SIMPULAN

Penerapan Algoritma Cat Swarm Optimization diawali dengan membuat solusi RCPSP yang valid dan dihitung *maskepan*nya. Bangkitkan sebanyak N kucing, termasuk kucing dengan solusi valid. Evaluasi kucing sesuai dengan posisi tiap kegiatan untuk mempercepat perhitungan nilai *fitness*. Berdasarkan N kucing yang diperoleh hitung nilai *maskepan*nya. Selanjutnya, pindahkan kucing sesuai dengan MR dalam *seeking mode* dan *tracing mode*. Evaluasi kembali posisi setiap kucing untuk

menyimpan kucing dengan nilai *fitness* terkecil. Akhiri algoritma dengan mengambil solusi yang memiliki *fitness* terkecil. Algoritma Cat Swarm Optimization dengan bantuan *software* Matlab telah diterapkan pada 2 kasus RCPSP, yaitu RCPSP dengan solusi valid yang dibuat memiliki nilai *maskepan* yang masih jauh dari nilai *maskepan* terkecil yang bisa diperoleh dan RCPSP dengan solusi valid yang dibuat telah mendekati *maskepan* terkecil yang bisa diperoleh. Pada kasus RCPSP dengan solusi valid yang memiliki nilai *maskepan* yang masih jauh dari nilai *maskepan* terkecil yang bisa diperoleh menghasilkan solusi yang berbeda-beda setiap kali program dijalankan sehingga diperlukan beberapa kali simulasi untuk memastikan solusi yang didapatkan benar-benar memiliki *maskepan* terkecil. Bila solusi valid yang dibuat memiliki *maskepan* yang telah mendekati *maskepan* terkecil yang bisa diperoleh program akan menghasilkan solusi yang sama setiap program dijalankan.Perlu menerapkan Algoritma Cat Swarm Optimization pada permasalahan lain yang akan diselesaikan adalah *Multi-Mode Resource Constrained Project Schedulling Problem* (MMRCPSP) karena permasalahan MMRCPSP lebih mendekati kondisi dunia nyata dari pada *Single-Mode Resource Constrained Project Schedulling Problem* (SMRCPSP). Dalam penelitian ini simulasi program dengan data masih dilakukan secara manual dimana data dimasukan dalam *coding.* Penelitian selanjutnya perlu ada pengembangan program yang lebih praktis dalam penginputan data secara langsung melalui *Exel, notepad, dll.* Dalam menyelesaikan kasus *Resource Constrained Project Schedulling Problem* (RCPSP) dapat menerapkan algoritma CSO yang lain, misalnya *Binary Cat Swarm Optimization, Average-Inertia Weighted Cat Swarm Optimization,* dan lainnya.

DAFTAR PUSTAKA

Arifudin, R. (2012). Optimasi penjadwalan proyek dengan penyeimbangan biaya menggunakan kombinasi CPM dan algoritma genetika. *Jurnal Masyarakat Informatika*, 2(4), 1-14.

Apriana, I.W.R., Tastrawati, N. K. T., & Sari, K. (2016). Implementasi algoritma cat swarm optimization dalam menyelesaikan job shop scheduling problem (JJSP). *E-jurnal Matematika,* 5(3), 90-97.

Chan, W. T., Chua, D. K., & Kannan, G. (1996). Construction resource scheduling with genetic algorithms. *Journal Construction Engineering and Management*, 122(2), 125-132.

Chu, S.C. & Tsai, P.W. (2006). Cat swarm optimization. *Proceedings of the 9th Pasific Rim International Conference on Artificial Intelligence LNAI* 4099, 854-858.

Dhanasaputra, N., Santosa, B., & Industri, J. T. (2010). Pengembangan algoritma cat swarm optimization (CSO) untuk klasifikasi. Jurnal ITS. 3.

Dhanasaputra, N. & Santosa B. (2010). Pengembangan algoritma cat swarm optimization (CSO) untuk klasifikasinya. Jurnal ITS. 3.

Putra, R.I., & Putranto. (2013). Penerapan algoritma harmony search pada resource-constrained project scheduling problem (RCPSP). *Jurnal Online Universitas Negeri Malang*. Diakses tanggal 20 Mei 2017.

Krisnawati, M. (2014). Penyelesaian permasalahan penjadwalan aktivitas proyek dengan batasan sumber daya menggunakan metode cross entropy. *Dinamika Rekayasa*, 10(1), 1-5.

Merkel, D., Minddendorf, M., & Schmeck, H. (2002). Ant colony optimization for resource-constained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4), 333-346.

Soeharto, I. (1999). *Manajemen Proyek (dari Konseptual sampai Operasional), Jilid I, Edisi 2*. Jakarta: Erlangga.

Widyawati, K., Mashuri, & Arifudin, R. (2014). Analisis algoritma branchand bound untuk menyelesaikan masalah penjadwalan proyek pembangunan mega tower. *UNNES Journal of Mathematics,* 3(1), 2460-5859.

Zhang, H., Li, H., & Tam, C.M. (2006). Particle swarm optimization for resource-constrained project scheduling. *International Journal of Project Management,* 24(1), 83- 92.