A Cross Entropy-Genetic Algorithm Approach for m-Machines No-Wait Job-Shop Scheduling Problem

Budi Santosa
Industrial Engineering
Kampus ITS Sukolilo
Surabaya
phone +62315992364
budi_s@ie.its.ac.id

Muhammad Arif Budiman
Industrial Engineering
Kampus ITS Sukolilo
Surabaya
phone +62315992364
arif_doudo@ie.its.ac.id

Stefanus Eko Wiratno
Indusrial Engineering
Kampus ITS Sukolilo
Surabaya
phone +62315992364
eko_w@ie.its.ac.id

ABSTRACT
No-wait job-shop scheduling (NWJSS) problem is one of the classical scheduling problem which is exist on many kinds of industries with no-wait constraint, such as metal working, plastic, chemical, and food industries. Several methods have been used for solving this problem, whether with exact (i.e. integer programming) or metaheuristic methods. Cross entropy, as a new metaheuristic method, can be an alternative method to approach to NWJSS problem. This method has been used in combinatorial optimization, also multi-extermal optimization and rare-event simulation. On these problems, cross entropy implementation result an optimal value with less computation time, i.e. on Support Vector Machines problem. This paper implemented cross entropy method combined with genetic algorithm (cross entropy – genetic algorithm/CEGA) on m-machines NWJSS, then compare the results with the other methods, Gasa and hybrid tabu search, on the same problem. The results showed that CEGA providing better or at least the same makespan with the other methods..

Keywords
no-wait job-shop scheduling, cross entropy, genetic algorithm, optimization, metaheuristic

1. INTRODUCTION
No-wait job-shop scheduling problem is a problem categorized to non-polynomial hard (NP-Hard) problem, especially for m-machines (Pan, 2009). Generally in job shop problem, each job has its own unique operation route. Then, when a wrong method used in scheduling purpose, it can make the makespan longer. In additional, the existing of no-wait constraint – e.g. on metal, plastic, and food industries – made it more complex, while a wrong choosing of scheduling method can make the makespan more-significantly longer, because the continuity of operation on each job must be kept to avoid operation reworking or job redoing.

Many researches using various methods to approach this problem were conducted, for example with Genetic Algorithm-Simulated Annealing (Schuster, 2003) or Hybrid Tabu Search (Bożejko, 2009). Several methods can't approach the optimum solution, the others can approach but with a long computational time.

Cross entropy method as a relatively new metaheuristic method has been used widely in many problems, such as combinatorial optimization, continuous optimization, noisy optimization, and rare event simulation (Rubinstein, 2004). At these problems, cross entropy can approach the optimal solution with a less computational time, for example in Support Vector Machine problem (Widyarini, 2009). This research will develop a new algorithm based to cross entropy hybridized with genetic algorithm (cross entropy-genetic algorithm or CEGA) for approaching the solution of NWJSS problem. Cross entropy-genetic algorithm purposed to become an alternative for getting a best schedule with better makespan on NWJSS problem.

2. PROBLEM OVERVIEW
No-wait job-shop scheduling (*NWJSS*) problem is a job-shop scheduling problem with a constraint not to allow any waiting time among two sequential operations for each job. This problem is met on real application for many industries with "no-wait" constraint, such as steel processing, plastic industries, and chemical-related industries (such as pharmacy and food industries), also for semiconductor testing purposes (Pan, 2009, and Liaw,

2008). On such industries, if there's waiting time exist on process, it may cause a defect on the product and need to be reworked with only such operation, or may cause a failure on product, means that we must redo all operations for related job from the beginning.

Based on the definition by Graham et. al. on Schuster (2003), NWJSS problem can be defined as below:

Given a set of machines M = {1, 2, …, m} purposed to process a set of jobs J = {1, 2, …, n}. For each i-th job ∈J, given a sequence of j operations O = {O(i,1), O(i,2), …, O(i,j)} as the detail of process in i-th job. Each operation has (m(i,j), w(i,j)) ∈ M×ℕ, specifying that operation O(i,j) will be processed on m(i,j) with processing time w(i,j).

No wait constraint is given by setting the condition of $O(i,j)$'s starting time must be same as $O(i,j-1)$'s finishing time. The, the assumptions used are: one job can't be processed at more than a machines on a time unit, or one machine can't process more than a job on a time unit; also there's no interruption or pre-emption allowed.

Generally, this problem is divided into two sub-problems:

- Sequencing, is how to find the best sequence of job-scheduling-priority with the best value of makespan obtained from all of the combination
- Timetabling, is how to get the best starting time for all jobs scheduled for getting better makespan than one obtained from sequencing sub-problem before (Zhu, 2009)

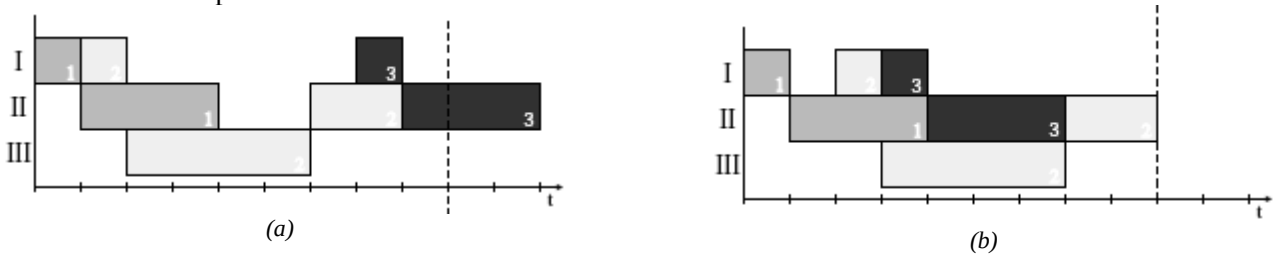
As illustration, given an example below:

Given a set of jobs $J=\{1,2,3\}$ which will be processed on a set of machines {I,II,III} as the route of machines and processing time for each machine stated on table 1below:

Table 1: NWJSS Case Example

Job	Operation	Machine	Processing Time
1	O_{11}	I	1
	O_{12}	II	3
2	O_{21}	I	1
	O_{22}	III	4
	O_{23}	II	2
3	O_{31}	I	1
	O_{32}	II	3

Sequencing sub-problem here is how to find the priorities of each job to be scheduled. There are probably 3! or 6 priority sequence: 1-2-3, 1-3-2, 2-1-3, 2-3-1, 3-1-2, and 3-2-1. Then, the timetabling sub-problem is how to get the best makespan from all priority sequences generated. For example, on priority sequence 1-2-3, with a kind of timetabling method, the timetable will result as be shown at Figure 1.a. When another method used, it may be results as Figure 1.b. From that explanation, we can conclude that the different usage of timetabling method will result different makespan values.



(a) *(b)*

Figure 1: Comparison of Timetable Results Using Different Methods for Sequence 1-2-3

The usage of optimum sequencing method with in-optimum timetabling method may not result an optimum makespan, also otherwise. So, for getting the best makespan result, we must choose the best method for combination of both sub-problems.

3. PROBLEM FORMULATION

Referring to Brizuela's model on Pan (2009), NWJSS problem with minimizing makespan objective can be modelled with integer programming formulation as below:

- Symbols definition

J_i	i-th job
M_k	k-th machine
O_i^k	Operation of J_i to e processed on M_k
$O_{(i,j)}$	j-th operation of J_i
N_i	Number of operations in J_i

- Problem parameters

M	A very large positive number
n	Number of jobs
m	Number of machines
w_i^k	Processing time J_i on M_k
$r_{(i,j,k)}$	1 if $O_{(i,j)}$ requires M_k, 0 otherwise

- Decision variables

C_{max}	Maximum completion time of all jobs (makespan)
s_i^k	The earliest starting time of J_i on M_k
$Z_{(i,i',k)}$	1 if J_i precedes $J_{i'}$ on M_k, 0 otherwise

- Problem formulation

Minimize C_{max}

Subject to

$$\sum_{k=1}^{m} r_{(i,j,k)}\left(s_i^k + w_i^k\right) = \sum_{k=1}^{m} r_{(i,j+1,k)} s_i^k \quad \textbf{(1)}$$

$$s_{i'}^k - s_i^k \geq w_i^k - M\left(1 - Z_{(i,i',k)}\right) \quad \textbf{(2)}$$

$$s_i^k - s_{i'}^k \geq w_{i'}^k - M Z_{(i,i',k)} \quad \textbf{(3)}$$

$$\sum_{k=1}^{m} r_{(i,N_i,k)}\left(s_i^k + w_i^k\right) \leq C_{max} \quad \textbf{(4)}$$

$$C_{max} \geq 0;\ s_i^k \geq 0; \quad \textbf{(5)}$$

with

$i \in \{1,2,...,n\}$; $j \in \{1,2,...,N_i\text{-}1\}$; $k \in \{1,2,...,m\}$; $Z_{(i,i',k)} \in \{0,1\}$; and $1 \leq i \leq i' \leq n$

Constraint (1) restrict that M_k begins the processing of $O_{(i,j+1)}$ right after $O_{(i,j)}$ finished (to ensure that no-wait constraints are obeyed). Constraints (2) and (3) enforce that only one job may be processed on a machine at any time. It's together used for guarantee that one of the constraints must hold when the other is eliminated, since $Z_{(i,i',k)}$ is a binary variable. Constraint (4) used to minimize C_{max} in objective function purposes. Then, constraint (5) guarantees that C_{max} and s_i^k are non-negative.

4. PROPOSED ALGORITHM

In this research, the proposed method to approach the NWJSS problem is a hybrid of cross entropy with genetic algorithm (hereinafter called the Cross Entropy-Genetic Algorithm / CEGA). The concept of cross entropy is used as the basic concept of the algorithm, while the genetic algorithm concept is applied only for samples generation.

For reference, cross entropy method was inspired by a concept of modern information theory, namely the concept of Kullback-Leibler distance or also well-known with the same name: the concept of cross entropy distance (Rubinstein, 2004). This concept was developed by Solomon Kullback and Richard Leibler, to measure the difference in the distance difference between an ideal reference distribution and the actual distribution. This method generally has two basic concepts, generating samples with specific mechanism and updating parameters based on elite samples. Meanwhile, genetic algorithm method was inspired by natural biological evolution theory developed by Mendel, which includes genes transmission, natural selection, crossover / recombination and mutation (Wikipedia, 2010).

For this NWJSS problem, the flowchart of algorithm proposed to approaching the best solution can be seen below:

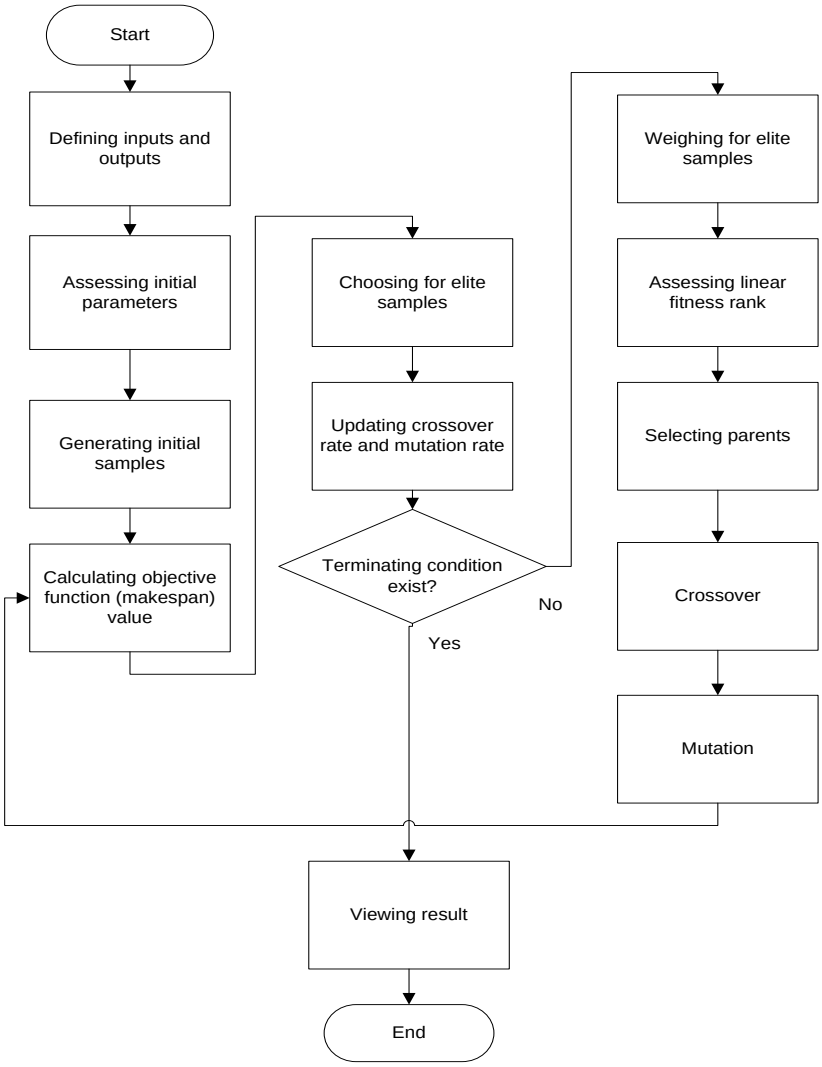


Figure 2: Flowchart of CEGA

The explanations of flowchart above are:

1) Defining inputs and outputs

Inputs are determined as something will be processed in the algorithm and outputs as something will be displayed as the result of process. As in this study, remembering that the problem is NWJSS and the method used is hybrid cross entropy-genetic algorithm, the inputs and outputs are determined as follows:

- Inputs:
  - Machine routing matrix (***R(j,k)***; *j* state the job number and *k* state the operation number
  - Processing time matrix (***W(j,k)***)
  - Number of sample generated (***N***)
  - Rarity parameter (***ρ***)
  - Smoothing coefficient (***α***)
  - Initial crossover rate ($\boldsymbol{P_{ps}}$)

- Terminating criterion (***β***)

- Outputs:
  - Best schedule's timetable (starting and finishing time of each job)
  - Best schedule's makespan ($\mathbf{C_{max}}$)
  - Computational time (***T***)
  - Number of iteration (***iterasi***)

2) Assessing initial parameters

The initial values of predefined inputs (***N***, ***ρ***, ***α***, initial $\boldsymbol{P_{ps}}$, and ***β***) are determined as parameters of results/outputs performance. Actually, different parameter values used will certainly get different results.

The rule of assessing initial values of parameters defined above is as follows:

- For number of sample generated ***N***, there's no restriction or guidance for how much it will be, but actually for more larger the number of job, more larger the number of sample generated. In this research we will define ***N*** as cubic of number of jobs (n^3)
- For rarity parameter ***ρ***, suggested range is between 1% and 10% (Kroese, 2009). In this research, we will use 2% as the value of ***ρ***
- For smoothing coefficient ***α***, it must be ranged at 0 – 1, but 0.4 – 0.9 is empirically the optimum range of value (de Boer, 2003). In this research, we will use 0.8 as the value of ***α***
- For crossover rate ($\boldsymbol{P_{ps}}$), we will use 1 for initial value
- For terminating criterion ***β***, the value must be limited to 0 ($\lim_{\beta \to 0} \beta$). In this research, we will use 0.001 as the value of ***β***

3) Generating samples

Generated samples here are defined as the sequence of job priorities to be scheduled as early as possible. Generation of initial samples (iterasi=1) is conducted with fully randomized sampling technique, whereas in all the next generations (iterasi>1) are conducted with genetic algorithm mechanism (crossover and mutation), which is detailed as below:

a) Weighting elite samples

This weighting is necessary for the next step (selecting parents), where the first parent will be selected from elite samples by considering the weight of each elite sample. The weighting rule is, if the makespan generated by the sequence is better than the best makespan ever visited of the previous iteration, the weight given is same as the number of elite sample, otherwise is given only 1.

b) Assessing linear fitness rank

Linear fitness rank (LFR) for actual iteration calculated from fitness value of all samples generated in the previous iteration. The value of *LFR* is formulated by $LFR(I(N-i+1))=F_{max}-(F_{max}-F_{min})((i-1)/(N-1))$, where the fitness value is same as 1/makespan value. *i* is stated the *i*-th sample (which is valued between 1 and *N*), and *I* state the job index on sample matrix.

c) Selecting parents

Parents selection is conducted by using roulette wheel selection technique, which samples with higher weight will have larger chance to be selected as parent. The first parent will only be selected from elite samples (with the weight calculated by step **3a)**), and the second parent will be selected from all of the last iteration samples with LFR weight from step **3b)**.

d) Crossover

Crossover will be conducted with two-point order-crossover technique, which the choosing of points held randomly from both of parents. The offspring resulted from this technique have the same segment between these two points with their parents. Other side, the other segment will be kept from the other different parent's sequence of jobs.

e) Mutation

Mutation will be conducted with swapping mutation technique, whereas mutation conducted by exchanging selected job with another job in the same offspring.

4) Calculating makespan value

The calculation of makespan value will be conducted with simple shift timetabling method, adapted from shift timetabling method by Zhu et. al. (2009). The steps are:

a) Schedule the first job from t = 0

b) Schedule the next job from t=0, then check while there are machine overloads. If they exist, shift job rightside until there are no machine overload exist

c) Repeat **b)** until all jobs were scheduled

5) Choosing elite samples

Elite samples will be chosen as $\lfloor \rho N \rfloor$ from number of sample generated $\boldsymbol{N}$ according to the value of makespan. Before this step conducted, better if we sort the samples by their makespans, from the lowest value to the largest ones.

6) Updating crossover rate and mutation rate

Parameter updating will be conducted with the value of u, which is taken each of iteration from ratio value of makespan average from elite samples with two times of the best makespan. Crossover rate then updated with $P_i=(1-\alpha)u+P_{i-1}$, and mutation rate defined as half of crossover rate.

7) Checking for terminating condition

Terminating condition in this research is when the difference between crossover rate actual with cross over rate from previous iteration is less than β. If this condition met, then stop the iterations and go to the next step. Otherwise, repeat again the algorithm from step **4)**.

8) Viewing result

Result viewed is the outputs defined before (best schedule's timetable and makespan, computational time, and number of iteration).

5. EXPERIMENT

The computational experiment for the algorithm is conducted by coding the algorithm using Matlab software. The Matlab code result is then executed with 30 replications, using computer specification Intel Core 2 Duo 1.66 GHz, 1 GB RAM, and using Matlab 6.5 series. The data used in this experiment are taken from OR Library, including Ft06, Ft10, La01-La25, Orb01-Orb06 and Orb08-Orb10.

The results of experiment are shown in table 2. Ref is the best known solution.

ARPD = best-ref/ref

Table 2: Statistical Results of CEGA for Small Instances

Instances	Size		Ref	Makespan CEGA				Time (sec)	
	Job	Mach		Best	Avg	StDev	ARPD	Avg	StDev
ft06	6	6	73	73	73,0	0,0	0,0	7,1	0,1

Instances	Size		Ref	Makespan CEGA				Time (sec)	
	Job	Mach		Best	Avg	StDev	ARPD	Avg	StDev
la01	10	5	971	975	990,1	14,7	0,4	132,6	10,8
la02	10	5	937	961	970,9	9,0	2,5	141,1	6,1
la03	10	5	820	820	852,4	19,3	0,0	133,0	12,2
la04	10	5	887	887	891,7	8,8	0,0	130,1	12,7
la05	10	5	777	781	788,0	11,4	0,5	149,3	12,4
ft10	10	10	1607	1607	1611,9	12,4	0,0	269,4	12,1
orb01	10	10	1615	1615	1630,6	20,2	0,0	307,8	24,2
orb02	10	10	1485	1485	1509,2	14,8	0,0	240,4	10,8
orb03	10	10	1599	1599	1620,2	19,8	0,0	295,8	16,9
orb04	10	10	1653	1653	1692,7	39,3	0,0	278,5	14,0
orb05	10	10	1365	1370	1390,3	18,7	0,4	257,1	14,9
orb06	10	10	1555	1555	1559,1	15,5	0,0	284,2	15,0
orb08	10	10	1319	1319	1319,0	0,0	0,0	291,7	3,4
orb09	10	10	1445	1445	1482,6	39,2	0,0	270,7	23,2
orb10	10	10	1557	1557	1585,6	16,2	0,0	253,4	17,7
la16	10	10	1575	1575	1581,5	19,9	0,0	250,9	13,6
la17	10	10	1371	1384	1405,5	24,9	0,9	241,5	21,6
la18	10	10	1417	1507	1509,7	9,1	6,0	240,0	17,8
la19	10	10	1482	1491	1531,4	34,8	0,6	253,2	7,4
la20	10	10	1526	1526	1542,5	28,8	0,0	255,8	11,4

Table 3: Statistical Results of CEGA for Large Instances

Instances	Size		Ref	Makespan CEGA				Time (sec)	
	Job	Mach		Best	Avg	StDev	ARPD	Avg	StDev
la06	15	5	1248	1304	1342,0	24,3	4,3	1635,8	236,9
la07	15	5	1172	1221	1265,7	23,9	4,0	1670,1	205,8
la08	15	5	1244	1274	1323,8	23,2	2,4	1626,9	187,2
la09	15	5	1358	1382	1443,1	21,2	1,7	1745,9	202,6
la10	15	5	1287	1299	1353,9	30,7	0,9	1624,3	250,6
la11	20	5	1671	1722	1793,5	31,9	3,0	10061,7	1097,2
la12	20	5	1452	1538	1597,9	26,2	5,6	9695,1	1102,8
la13	20	5	1624	1674	1759,1	30,7	3,0	10525,0	1028,4
la14	20	5	1691	1749	1821,4	31,6	3,3	9976,7	1199,6
la15	20	5	1694	1752	1851,9	41,3	3,3	10722,1	1237,0
la21	15	10	2048	2054	2209,8	62,1	0,3	3032,3	426,2
la22	15	10	1887	1910	1972,1	42,1	1,2	2970,9	418,5
la23	15	10	2032	2098	2184,0	45,2	3,1	2995,8	429,7
la24	15	10	2015	2056	2133,6	36,9	2,0	2889,1	332,4
la25	15	10	1917	1994	2059,4	31,7	3,9	3049,5	458,7

From the experiment, we can see that the performance of CEGA is relatively well, especially for small instances. It can be seen from ARPD value on small instances which mostly on level 0.0 and almost overall levelled below 1.0 (except La02 and La18), which means that the best makespan value obtained by CEGA are almost same as optimum value obtained from manually branch and bound calculation. In addition, for Ft06 and Orb08, this algorithm can result ARPD value 0.0 with standard deviation 0.0, which means that the results obtained from all 30 repetitions are uniform with the optimal value.

For larger size of instances, CEGA performance tends to decline. It can be seen from ARPD values on large instances are above 1.0. It is happened because for larger number of jobs processed, the number of search space will increase as factorial, which is unable to be followed by search sample number increasing. For example, to increase from 10 jobs to 15 jobs, search space will increase from 10! to 15! or 15x14x13x12x11=360360 times larger than 10 jobs. This increase is of course unable to be followed by the increase in the number of samples, considering that computing time will also increase drastically. However, by using the number of sample n^3, we can say that this algorithm performance still on good tolerable value, which is indicated by the presence ARPD values below 1.0 (for example in La10 and La21) and most of ARPD value are still below standard error 5.0 (except La12).

Figure 3: Computational Time Graph Based on Job Size

When it is reviewed from the average and standard deviation value of the makespan obtained of from all repetitions, the average values obtained tend to approach the reference makespan value, while the standard deviation values tend to be large (greater than 10.0, except in certain cases). That means this algorithm tends to produce non-uniform makespan value at each repetition, but with the probability of getting the best result is relatively higher. We can say that such performance is good enough, considering that these results show the existence of an effort to not get stuck on just one objective value (in this case, local optimum). But of course, this algorithm actually needs to be optimized again for better performance value, by reducing the standard deviation value with better or at least same average value (which means the makespan values obtained at each repetition will be more uniform but still tend to approach best value of reference).

Although this algorithm produces better overall makespan, but the calculation time is relatively long and will increase significantly when the job size are larger. This fact can be seen on Figure 2, where the average time needed by this algorithm will increase drastically as the addition of job size, followed by standard deviation value increase. This is alleged to be caused by timetabling process to calculate the objective function which is relatively time consuming. As explained in the previous that simple shift timetabling method used requires checking for the presence of overload on the machine for each new job scheduled. Every will-be-scheduled-job have a specific machine overload when it will be scheduled, it must be shifted as far as the relevant value of time of overload on that machine. When after being moved the other machines are overload, then the shifting must be performed again

until all of machines have no overload. This mechanism certainly takes a long computing time, especially if the size of the jobs that must be scheduled is larger. In addition, the use of computer and software specifications is also influencing the calculation speed of the algorithm.

Then, when compared with another algorithms, Genetic Algorithm-Simulated Annealing (Schuster, 2003) and Hybrid Tabu Search (Bożejko, 2009), CEGA performance can be shown in Table 4 and Table 5.

Table 4: Makespan Comparison of GASA, HTS, and CEGA in Small Instances

Instance	Ref	GASA		HTS		CEGA	
		Best	ARPD	Best	ARPD	Best	ARPD
ft06	73	**73**	**0,0**	**73**	**0,0**	**73**	**0,0**
la01	971	1037	6,4	**975**	**0,4**	**975**	**0,4**
la02	937	990	5,4	975	4,1	**961**	**2,5**
la03	820	832	1,4	**820**	**0,0**	**820**	**0,0**
la04	887	889	0,2	889	0,2	**887**	**0,0**
la05	777	817	4,9	**777**	**0,0**	781	0,5
ft10	1607	1620	0,8	**1607**	**0,0**	**1607**	**0,0**
orb01	1615	1663	2,9	**1615**	**0,0**	**1615**	**0,0**
orb02	1485	1555	4,5	1518	2,2	**1485**	**0,0**
orb03	1599	1603	0,2	**1599**	**0,0**	**1599**	**0,0**
orb04	1653	**1653**	**0,0**	**1653**	**0,0**	**1653**	**0,0**
orb05	1365	1415	3,5	**1367**	**0,1**	1370	0,4
orb06	1555	**1555**	**0,0**	1557	0,1	**1555**	**0,0**
orb08	1319	**1319**	**0,0**	**1319**	**0,0**	**1319**	**0,0**
orb09	1445	1535	5,9	1449	0,3	**1445**	**0,0**
orb10	1557	1618	3,8	1571	0,9	**1557**	**0,0**
la16	1575	1637	3,8	**1575**	**0,0**	**1575**	**0,0**
la17	1371	1430	4,1	**1384**	**0,9**	**1384**	**0,9**
la18	1417	1555	8,9	**1417**	**0,0**	1507	6,0
la19	1482	1610	8,0	**1491**	**0,6**	**1491**	**0,6**
la20	1526	1693	9,9	**1526**	**0,0**	**1526**	**0,0**
Average			**3,5**		**0,5**		**0,5**

Table 5: Makespan Comparison of GASA, HTS, and CEGA in Large Instances

Instances	Ref	GASA		HTS		CEGA	
		Best	ARPD	Best	ARPD	Best	ARPD
la06	1248	1339	6,8	**1248**	**0,0**	1304	4,3
la07	1172	1240	5,5	**1172**	**0,0**	1221	4,0
la08	1244	1296	4,0	1298	4,2	**1274**	**2,4**
la09	1358	1447	6,2	1415	4,0	**1382**	**1,7**
la10	1287	1338	3,8	1345	4,3	**1299**	**0,9**
la11	1671	1825	8,4	**1704**	**1,9**	1722	3,0

Instances	Ref	GASA		HTS		CEGA	
		Best	ARPD	Best	ARPD	Best	ARPD
la12	1452	1631	11,0	**1500**	**3,2**	1538	5,6
la13	1624	1766	8,0	1696	4,2	**1674**	**3,0**
la14	1691	1805	6,3	**1722**	**1,8**	1749	3,3
la15	1694	1829	7,4	**1747**	**3,0**	1752	3,3
la21	2048	2182	6,1	2191	6,5	**2054**	**0,3**
la22	1887	1965	4,0	1922	1,8	**1910**	**1,2**
la23	2032	2193	7,3	2126	4,4	**2098**	**3,1**
la24	2015	2150	6,3	2132	5,5	**2056**	**2,0**
la25	1917	2034	5,8	2020	5,1	**1994**	**3,9**
Average			**6,5**		**3,3**		**2,8**

Highlighted-red value: The best makespan results of comparison for each instance

- *Ref*: Reference makespans, for small instances are the optimum value obtained by branch and bound algorithm, otherwise for large instances are the best known makespan ever obtained by researches until present. (Zhu, 2009).
- *Best*: The best makespan value obtained by CEGA with 30 replications.
- *Avg* and *StDev*: Statistical value of average and standard deviation of CEGA's makespan and computation time for 30 replications
- *ARPD* (*Average Relative Percentage Deviation*) are standard of percentage deviation between the best obtained makespan (Best) and reference makespan (Ref), that can be formulated as:

$$ARPD = \frac{Best - Ref}{Ref} \times 100$$

From the comparison results above, we can see that when compared to GASA, the performance of CEGA based on the best makespan obtained is absolutely much better, proven by the large amount of difference between the average of ARPD of GASA and CEGA, and by overall makespan value obtained where CEGA absolutely superior.

Meanwhile, when compared to HTS, they have almost same level of performance but CEGA is slightly better. It can be seen from the difference between the averages of ARPD, which in small instances valued 0 almost all of them so that it can be considered to insignificant difference, while for the large instances it only valued 0.5. In addition, the makespan value obtained by CEGA are not absolutely superior compared to HTS, for example for La06 and La07 that the best makespan obtained from CEGA is worst than HTS.

6. CONCLUSIONS

As the result showed, it can be stated that CEGA can be used as an alternative of NWJSS problem solving approach, and can be applied widely on many industries with NWJSS characteristics, remembering that CEGA performance is actually better than the comparing method (GASA and HTS), especially for small size instances.

It's suggested that at the future researches, CEGA for NWJSS must be modified to get its performance better especially for the large size instances. On the other hand, this algorithm application on the other problems is also suggested.

7. REFERENCES

[1] Baker, Kenneth R. 1974. *Introduction to Sequencing and Scheduling*. New York: John Wiley & Sons, Inc.

[2] Beasley, J. E., 1990. *OR-Library: Distributing Test Problems by Electronic Mail*. Journal of the Operational Research Society 41: 1069 – 1072.

[3] Boubezoul, Abderrahmane; Paris, Sébastien; dan Ouladsine, Mustapha. 2008. *Application of the Cross Entropy Method to the GLVQ Algorithm*. Pattern Recognition 41: 3173 – 3178

[4] Bożejko, Wojciech dan Makuchowski, Mariusz. 2009. *A Fast Hybrid Tabu Search Algorithm for the No-Wait Job Shop Problem*. Computers & Industrial Engineering 56: 1502–1509

[5] Caserta, M.; Rico, E. Quiñonez; dan Uribe, A. Márquez. 2008. *A Cross Entropy Algorithm for the Knapsack Problem with Setups*. Computers & Operations Research 35: 241 – 252

[6] de Boer, Pieter-Tjerk; Kroese, Dirk P.; Mannor, Shie; dan Rubinstein, Reuven Y. 2003. *A Tutorial on the Cross-Entropy Method*. Haifa: Department of Industrial Engineering, Technion – Israel Institute of Technology

[7] Dewi, Dian Retno Sari. Pengembangan Algoritma Penjadualan Produksi Job Shop untuk Meminimumkan Total Biaya Earliness dan Tardiness. Jurnal Ilmiah Teknik Industri 4 no. 2: 57 – 65

[8] Ganduri, Chandrasekhar V. 2004. *Rule Driven Job-Shop Scheduling Derived from Neural Networks Through Extraction*. Department of Industrial and Manufacturing Systems Engineering, Ohio University

[9] Kroese, Dirk P. 2009. *Cross-Entropy Method*. Brisbane: Mathematics Department, University of Queensland

[10] Liaw, Ching-Fang. 2008. An Efficient Simple Metaheuristic for Minimizing The Makespan in Two-Machine No-Wait Job Shops. Computer & Operations Research 35: 3276 – 3283

[11] Mascis, Alessandro, dan Pacciarelli, Dario. 2002. *Discrete Optimization: Job-Shop Scheduling with Blocking and No-Wait Constraints*. European Journal of Operational Research 143: 498 – 517

[12] Pan, Jason Chao-Hsien, dan Huang, Han-Chiang. 2009. *A Hybrid Genetic Algorithm for No-Wait Job Shop Scheduling Problems*. Expert Systems with Application 36: 5800 – 5806

[13] Rera, Gladiez Florista. 2010. Penerapan Metode Cross Entropy dalam Penyelesaian Capacitated Vehicle Routing Problem – Studi Kasus: Distribusi Koran Jawa Pos Surabaya. Surabaya: Institut Teknologi Sepuluh Nopember

[14] Rubinstein, Reuven Y., dan Kroese, Dirk P. 2004. The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning. New York: Springer Science+Business Media, Inc.

[15] Schuster, Christoph J. dan Framinan, Jose M. 2003. *Approximative Procedures for No-Wait Job Shop Scheduling*. Operations Research Letters 31: 308 – 318

[16] Widyarini, Tiananda. 2009. *Aplikasi Metode Cross Entropy untuk Support Vector Machines*. Surabaya: Institut Teknologi Sepuluh Nopember

[17] Waterloo Manufacturing Software. *Job Shop Scheduling.* http://www.waterloo-software.com/planning-and-scheduling/job-shop-scheduling.html. Accessed on February 24th, 2010

[18] Wikipedia. *Cross Entropy Method*. http://en.wikipedia.org/wiki /Cross_Entropy_Method. Accessed on February 19th, 2010

[19] _____. *Crossover* (*Genetic Algorithm*). http://en.wikipedia. org/wiki/Crossover_(Genetic_Algorithm). Accessed on June 18th, 2010

[20] _____. *Genetic Algorithm*. http://en.wikipedia.org/wiki/ Genetic_Algorithm. Accessed on June 18th, 2010

[21] _____. *Kullback-Leibler Divergence*. http://en.wikipedia. org/wiki/Kullback-Leibler_Divergence. Accessed on February 19th, 2010

[22] _____. *Mutation* (*Genetic Algorithm)*. http://en.wikipedia. org/wiki/Mutation_(Genetic_Algorithm). Accessed on June 18th, 2010

[23] _____. *Selection (Genetic Algorithm*). http://en.wikipedia. org/wiki/Selection_(Genetic_Algorithm). Accessed on June 18th, 2010

[24] Zhu, Jie; Li, Xiaoping; dan Wang, Qian. 2009. *Complete Local Search with Limited Memory Algorithm for No-Wait Job Shops to Minimize Makespan*. European Journal of Operational Research 198: 378–386