An Adjusted Intelligent Water Drop (AIWD) Algorithm in Flow Shop Scheduling Problem

Suprapto and Kurnia Damarenjang Pintati
Departement of Computer Science and Electronics
Faculty of Mathematics and Natural Sciences, Universitas Gadjah Mada
Yogyakarta, Indonesia
sprapto@ugm.ac.id and kurnia.damarenjang@mail.ugm.ac.id

***Abstract*— The intelligent water drops (IWD) algorithm was introduced in 2007 by Hamed Shah Hosseini, and has been successfully implemented to solve many types of optimization problems, especially traveling salesman problem (TSP). The algorithm works by imitating the behavior of the natural river water drops. Principally, the algorithm finds solution of the problem by constructing it iteratively in certain number of stages.**

In this paper, we adjusted (or slightly modified) the IWD algorithm in order to be properly implemented in the flow shop scheduling problem. The testing was conducted on the Taillard (1993) benchmark, and the results were compared with the ones obtained by MOACSA (multi *objective ant colony system algorithm)*. Based on the comparison especially on the makespan values between both algorithms, it is seen that the adjusted IWD algorithm gave better solution than MOACSA on solving flow shop scheduling problem with the objective to optimize the makespan.

***Index Term*-- *Intelligent* water drops, scheduling problem, flow shop scheduling, optimization problem, makespan, adjusted IWD**.

I. INTRODUCTION

Scheduling is a decision making process to allocate resources to tasks in a certain time interval in order to optimize one or more objectives [12]. Scheduling problems are becoming more complicated as the increase of tasks number to be scheduled, and constraints number to be met. Flow shop scheduling is one of the most widely studied scheduling problems [1]. The scheduling problem is to schedule n tasks in a system with m machines in the equal order of machine processing for each task. Every job (task) must be processed in m machines. And the objective of this scheduling is to find the order of tasks processing for each machine that minimize makespan, respect to some specified criteria (or constraints).

There have been many approaches used to solve flow shop scheduling problems, such as branch and bound and beam search [2], exact calculation method [11] that require many calculations and times. Flow shop scheduling problem may involve a large number of jobs and machines such that, it can be classified as an optimization combinatorial problem. The approach of meta-heuristic has been widely used to solve the optimization combinatorial problems, including the flowshop scheduling problems. Ant colony optimization, bee colony optimization, genetic algorithm, particle swarm optimization, etc., are examples of metaheuristics algorithms that have been successfully implemented to solve flow shop scheduling problems.

The intelligent water drop (IWD) algorithm was first time introduced by Hamed Shah-Hosseini in 2007, and in his research he successfully implemented it in solving the traveling salesman problem. At the same time, he also stated that IWD algorithm worked very much like ant colony did. Therefore, the success of the research about ant colony algorithm in solving flow shop scheduling algorithm have motivated to implement IWD algorithm in solving the flow shop scheduling problem. The capability of this algorithm to change their properties (i.e., $velocity$ and $soil$); it is expected that the algorithm capable of finding a solution for flow shop scheduling problem better than what had been found by ant colony algorithm.

II. RELATED WORKS

The two ant-colony optimization algorithms – *max-min* ant system are developed by Stuetzle, and continue to be developed to solve the flow shop scheduling problems [4]. The flow shop scheduling problem being solved in this re-search was specialized on flow shop permutation scheduling to minimize makespan and total flow time. Another research introduced IWD algorithm [7] inspired by the flow of water in the river that can be a good flow path from one source to certain target. Some properties of water flow in the river are adopted into algorithm to solve optimization problem. It was also mentioned that this algorithm has some equalities in the way to find a solution with ant colony algorithm. In this research, IWD algorithm is used to solved the traveling salesman problem.

Further implementation of IWD algorithm in *n*-queen puzzle, and multiple knapsack problems [8]. The research results proved that IWD algorithm was able to find optimal solution for several optimization problems. Flow shop scheduling problem was solved by using ant colony algorithm [2]. In this research, ant colony algorithm was paired with local search algorithm in order to find solution for flowshop scheduling problem that minimize makespan and total of flow time. Discrete artificial bee colony algorithm was implemented in solving flow shop scheduling problem [10]. This research compared the quality of solution and CPU time obtained from discrete artificial bee colony algorithm, and evolution differential hybrid algorithm with several algorithms that have good performance – iterated greedy algorithm, genetic algorithm hybridized with a tabu search, insertion search with cut and repair, variable neighborhood search, and estimation of distribution algorithm. The testing results proved that both algorithms were highly competitive with the

mentioned algorithms.

It is different than those of previous mentioned researches, this research implements IWD algorithm by adjusting it (slight modification) to the problem, which is flow shop scheduling. The research is highly motivated by the statement that IWD algorithm worked similar to ant colony, and by considering the success of ant colony algorithm in solving flow shop scheduling problems. In the end, the solution obtained by adjusted IWD algorithm was compared with the one obtained by ant colony algorithm. Based on the comparison of the makespan values resulted by both algorithms, proven that adjusted IWD algorithm was better than ant colony algorithm.

III. Discussion

This section gives an overview of flow shop scheduling, and IWD algorithm, and then presents the solution of flow shop scheduling problem by the adjusted IWD algorithm.

A. Flow shop scheduling

The shop contains *m* different machines, and each job consists of *m* operations, each of which requires a different machine. The flow shop is characterized by a flow of work that is unidirectional. When machines are numbered, so that if the *j*th operation of any job precedes its *k*th operation, then the machine required by the *j*th operation has a lower number than the machine required by the *k*th operation. The machines in a flow shop are numbered 1, 2, ..., *m*; and the operation of job_i correspondingly are numbered (*i*,1),(*i*,2),...,(*i*,*m*). Each job can be treated as if it has exactly *m* operations, in case there is job_i; for some *i* has fewer than *m* operations, the corresponding processing times can be set to zero [9]. The flow of work in pure flow shop is shown in Figure 1 (a), and (b) shows the flow of work in a more general one.

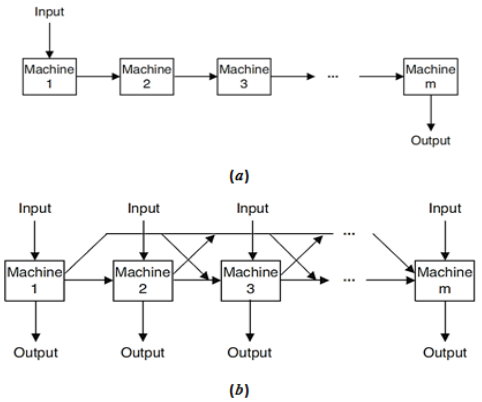

Fig. 1. (a) The flow of work in pure flow shop, (b) the flow of work in a more general flow shop

B. Flow shop Representation

Flow shop scheduling is represented by a disjunctive graph [2] [13], which is generally described as graph *G* = (*V, C, D*); with *V* is a set of nodes representing all operations of all jobs, *C* is set of directed conjunctive arc representing precedence relation among jobs (job precedence), and *D* is set of undirected disjunctive arc relating operations processed in the same machine. Figure 2 shows an example of disjunctive graph for flow shop scheduling problem with 4 jobs and 4 machines.

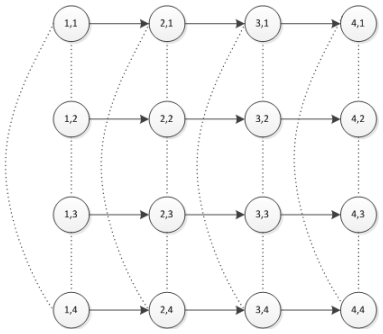

Fig. 2. Example of disjunctive graph for flow shop scheduling problem with 4 jobs and 4 machines

As scheduling problems in general, their solution (i.e., schedules) can be represented by Gantt chart [9] [13]. The chart is shown in Figure 3.

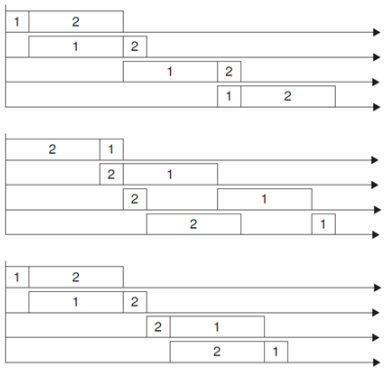

Fig. 3. Example of Gantt chart for solution representation of flow shop scheduling

Flow shop scheduling problem optimizing makespan can be detonated with triplet $F|\beta|C_{max}$; where F describes machine environment, β is a field for constraint and characteristic process, and C_{max} is the problem objective – minimize makespan.

Makespan (C_{max}) of flow shop scheduling is amount of time of the last job in a machine had been processed. Therefore, if C_j is completion time of job_j, makespan for flow shop scheduling with m machines and n jobs is defined as $C_{max} = max\{C_j | 1 \leq j \leq m\}$. Hence, the definition of C_{max} indicates that the smaller makespan value of schedule the more optimal the solution.

C. Intelligent water drop

The IWD algorithm was first time introduced in 2007 by Hamed Shah-Hosseini. It is built by two properties : $velocity$, the speed of algorithm; and $soil$, the amount of soil is carried by algorithm. These properties may change during its life time.

D. Intelligent water drop algorithm

The IWD algorithm employs a number of *IWD* to find optimal solution for a given problem. The problem will be represented by a graph (N, E) with *N* is set of nodes, and *E* is a set of edges. Every *IWD* started building its solution iteratively by visiting nodes of graph via its edges until the

condition of termination is achieved. End of each iteration is indicated by the completion of solution generation of all *IWD*. At the end of each iteration, the best iteration solution (T^{IB}) is chosen, and it is used to update the best solution (T^{TB}). The number of soil on edges that generates T^{IB} will be subtracted based on quality of the solution, and the algorithm will perform next iteration with new values of parameters. The algorithm will stop when either the maximum iteration or T^{IB} with the required quality had been achieved.

The IWD algorithm has two kinds of parameters: statical parameter, one that has constant value during algorithm is running; and dynamical parameter, one that its value will be reinitialized at the end of each iteration. The steps of IWD algorithm can be described as follows.

1) Initialization of statical parameters, such as graph (N, E), the best total solution T^{TB} initialized by the worst value, number of iteration $iter_{max}$ determined by user, iteration counter variable $iter_{count}$ initialized by 0. The number of water drop N_{IWD} initialized by a positive integer value, usually equal to number of node N_C in graph of problem. Parameters $a_v = 1$, $b_v = 0.001$, and $c_v = 1$ are used to update *velocity* variable of *IWD*. Parameters $a_s = 1$, $b_s = 0.001$, and $c_s = 1$ are used to update $soil$ variable of *IWD*. Parameters $\rho_n = 0.9$ $\rho_n = 0.9$ is used to update local $soil$ variable of *IWD*, and $\rho_{IWD} = 0.9$ is used to update global $soil$ variable of *IWD*. Parameter $InitSoil$ used to initialized $soil$ for each path between two nodes *i* and *j*, $soil(i,j) = InitSoil$. Initialization of $velocity$ for each IWD is done by giving a value corresponds tothe value of $InitVel$ parameter, while the values of $InitVel$ and $InitSoil$ parameters are determined by user.
2) Initialization of dynamical parameters. Every IWD has $V_C(IWD)$ that records nodes that had already been visited by that IWD, $V_C(IWD)$ is initialized by empty set {}. Besides, it also has $velocity$ that is initialized by a value corresponds to $InitVel$, and *soil* for each IWD is initialized by 0.
3) Randomly distribution of all of IWD to nodes in graph as the first nodes that will be visited by that IWD.
4) Insertion of first node for every IWD to corresponding $V_C(IWD)$ of that IWD.
5) Repetition of steps from 5.a) through 5.d) for every IWD until all nodes had been visited.
   a. For IWD is currently in $nodei$, choose $nodej$ that satisfy constraints of the problem, and hasnot been in $V_C(IWD)$ with probability $p_i^{IWD}(j) = \frac{f(soil(i,j))}{\sum_{k \notin V_C(IWD)} f(soil(i,k))}$, with $f(soil(i,j)) = \frac{1}{s_s + g(soil(i,j))}$, and $g(soil(i,j)) = soil(i,j)$ if $min_{l \notin V_C(IWD)}(soil(i,l)) \geq 0$, and $g(soil(i,j)) = soil(i,j) - min_{l \notin V_C(IWD)}(soil(i,l))$ for others. Node *j* (i.e., $node_j$) is then inserted into $V_C(IWD)$.
   b. For every moving IWD from $node_i$ to $node_j$, variabel $vel^{IWD}(t)$ is updated with (according to) $vel^{IWD}(t+1) = vel^{IWD}(t) + \frac{a_v}{b_v + c_v soil^2(i,j)}$.
   c. Computation of $soil$ obtained by IWD from path between $node_i$ and $node_j$ travesed, i.e., $\Delta\, soil(i,j)$ according to the formula $\Delta\, soil(i,j) = \frac{a_s}{b_s + c_s time^2(i,j:vel^{IWD}(t+1))}$ where $time^2(i,j;vel^{IWD}(t+1)) = \frac{HUD(j)}{vel^{IWD}(t+1)}$ and $HUD(j)$ is determined heuristic undesirability.
   d. Updating $soil(i,j)$ from path connecting $node_i$ to $node_j$, and also $soil^{IWD}$ (one that is brought by IWD) according to formula $soil(i,j) = (1-\rho_n)soil(i,j) + \rho_n \Delta\, soil(i,j)$ and formula $soil^{IWD} = soil^{IWD} + \Delta soil(i,j)$.
6) Selecting the best iteration solution T^{TB} among all solutions T^{IWD} obtained by IWD according to the formula $T^{IB} = argmax_{\forall T^{IWD}}\, q(T^{IWD})$, where q is quality function of solution.
7) Updating $soil$ on selected path by the best iteration solution T^{TB} according to the formula $soil(i,j) = (1+\rho_{IWD})soil(i,j) - \rho_{IWD}\frac{1}{N_{IB}-1} soil_{IB}^{IWD}, \forall (i,j) \epsilon T^{TB}$, where N_{IB} is the number of nodes in T^{IB}.
8) Updating the best total solution T^{TB} for this iteration, according to $T^{TB} = T^{TB}$ if $q(T^{TB}) \geq q(T^{IB})$, $T^{TB} = T^{IB}$ otherwise.
9) Updating number of iteration, $Iter_{count} = Iter_{count} + 1$, if $Iter_{count} < Iter_{max}$, then return to step 2.
10) The IWD algorithm terminates with the best solution T^{TB}.

E. An adjusted IWD for Flow shop scheduling problem

In order to conveniently implement IWD algrithm to find solution for flow shop scheduling problem, first is a certain number of adjustments performed. The adjusted IWD algorithm creates a number of intelligent water drops IWD, places them on a number of points (or nodes) to start performing their flows and building the solution of the problem as IWD's are flowing. Therefore, the problem to be solved must be represented by a graph. The flow shop scheduling problem to be solved by IWD algorithm is a flow shop with m machines and n jobs in order to minimize makespan.

The schedule is represented as a digraph $G = (N, E)$ with N is set of nodes (i, j) – operation of job j in machine i, and E is a set of edges as a union of disjunctive arcand conjunctive arc of flow shop scheduling denoted by $G = (V, C, D)$. Adjusted IWD algorithm constructs the solution for flow shop scheduling problem in certain number of stages, the solution is constructed by every created IWD, so that one IWD will construct one schedule. Every IWD flows through every node in the problems' digraph, and the flows of IWD to a node indicates that operation represented by that node is running.

The following are some adjustments made in order to IWD algorithm can be implemented to find solution of flow shop scheduling problem.

1) Adding statical parameter $NIWD$ denotes the number of created IWD, and initialized by number of jobs in scheduling, $NIWD = n$. Only nodes that represent first operation of each job $(1, j)$ will be processed at the first time, and will be considered as initial nodes.
2) Adding statical parameter of processing time $p(i,j)$, for operation (i,j).
3) Adding statical parameter V_C^{TB} will store the list of $V_C(IWD)$ of the best total solution.
4) Adding dynamical parameter $t_{machine}$ and t_{job} for every IWD used to compute makespan of schedule built by every IWD. tmachine records time where machine finish processing, while t_{job} records time where job finish being processed. The initial value ofboth $t_{machine}$ and t_{job} are zero.
5) Adding dynamical parameter stage for every IWD and every job, used to store machine number where every job the last time being processed. Besides, stage parameter also used to determined which nodes are visitable by every IWD during building the solution. The initial value of stage for each job are initialized by 0 – all of jobs have not been processed, and the value will be updated by last machine number where the corresponding job being processed at the last time.
6) For step 3 of IWD (original) algorithm, the distribution of IWD to a number of nodes did not perform randomly. Each of IWD will be placed at every first operation node of each job.
7) Adding step to step 4 of IWD (original) algorithm.This additional step updates values of $t_{machine}$, t_{job} and stage parameters for every IWD. For every node (i,j) that has just been visited, $t_{machine}(i) = p(i,j)$, $t_{job}(j) = p(i,j)$, and $stage(j) = 1$.
8) At the step 5.1 of IWD (original) algorithm, the selection of next visited nodes by IWD must meet the job precedence rule in flow shop scheduling. The following are the rules for selecting those nodes.
   - node $(1,j)$, if $stage(j) = 0$,
   - node $(i+1,j)$, if $stage(j) = i$, for $1 \leq i \leq m$.
9) First condition denotes the nodes of first operation every job that has not been in $V_C(IWD)$ list. First operation (the operation in machine 1) of each job can be performed any time. This means that these nodes can be visited by IWD providing they are not in the list of $V_C(IWD)$. While the second makes sure thatjob precedence in flow shop scheduling is satisfied – operations of job must done sequentially, starting from operation $(1, j)$, then $(2, j)$, etc. Hence, operation $(i + 1, j)$ can be done only if operation (i, j) had been done. In IWD algorithm, this means that nodes $(i + 1, j)$ can be visited only if node (i,j) had been visited (or already been in $V_C(IWD)$ list). When a job has $stage = m$ means the job is last time processed in machine m, then nodes representing this job will not be put in the list of next visitable nodes.
10) Heuristic undesirability (HUD) for flow shop scheduling is assigned by the same value as idle time of machines in this scheduling process. This HUD value measures how big the unwillingness of IWD not to move from one node to the next. The value of machine idle time is chosen as the value of HUD, since the value of HUD will be used to determine time required by IWD to flow to the next visited node, and this time value must be proportional with its HUD value. Hence, the bigger value of idle time (or HUD) the longer time required by IWD to move from a node. For an IWD that will move from node (i,j) to node (x,y), the value of HUD can be determined by taking the value of idle time of machine x – how long this machine should be idle until node (x,y) is visited. The value of HUD for IWD that move from node (i,j) to node (x,y) is given by : $HUD(i,j)(x,y) = t_{job}(y) - t_{machine}(x)$, if $x \neq 1$ and $t_{machine}(x) < t_{job}(y)$, and $HUD(i,j)(x,y) = 0$, for others. The value of HUD will be zero if $x = 1$, it means that the next node is an operation that will be processed in machine 1 (first machine). It happened since any operation processed in the first machine will immediately be started providing the previous operation for this machine had been done, so that the first machines will never have idle time. While idle time of other machines are computed by comparing the values of $t_{machine}(x)$ and $t_{job}(y)$. If $t_{machine}(x) < t_{job}(y)$ – job_y has not been completely processed in machine $(x-1)$ when machine x is ready to process job_y, then according to job precedence in the flow shop scheduling, operation (x,y) will start once job_y had been processed in machine $x-1$. In this case, machine x must be idle until job_y had been done with machine $(x-1)$ – the value of idle time of machine x is equal to the last time job_y being processed in machine $(x-1)$ minus time of machine x. If $t_{macine}(x) > t_{job}(y)$, then job_y had been processed in machine $(x-1)$, when machine x ready to process job_y, then job_y is immediately processed in machine x and the value of idle time of machine x is equal to zeroHeuristic undesirability (HUD) for flow shop scheduling is assigned by the same value as idle time of machines in this scheduling process. This HUD value measures how big the unwillingness of IWD not to move from one node to the next. The value of machine idle time is chosen as the value of HUD, since the value of HUD will be used to determine time required by IWD to flow to the next visitednode, and this time value must be proportional with its HUD value. Hence, the bigger value of idle time (or HUD) the longer time required by IWD to move from a node. For an IWD that will move from node (i,j) to node (x,y), the value of HUD can be determined by taking the value of idle time of machine x – how long this machine should be idle until node (x,y) is visited. The value of HUD for IWD that move from node (i,j) to node $(x; y)$ is given by : $HUD(i,j)(x,y) = t_{job}(y) - t_{machine}(x)$, if $x \neq 1$ and $t_{machine}(x) < t_{job}(y)$, and $HUD(i,j)(x,y) = 0$, for others. The value of HUD will be zero if $x = 1$, it means that the next node is an operation that will be processed in machine 1 (first machine). It happened since any operation processed in

the first machine will immediately be started providing the previous operation for this machine had been done, so that the first machines will never have idle time. While idle time of other machines are computed by comparing the values of $t_{machine}(x)$ and job_y. If $t_{machine}(x) < t_{job}(y)$, then job_y has not been completely processed in machine $(x-1)$ when machine x is ready to process job_y, then according to job precedence in the flow shop scheduling, operation (x, y) will start once job_y had been processed in machine $(x-1)$. In this case, machine x must be idle until job_y had been done with machine $(x-1)$ – the value of idle time of machine x is equal to the last time job_y being processed in machine $(x-1)$ minus time of machine x. If $t_{machine}(x) > t_{job}(y) - job_y$ had been processed in machine $(x-1)$, when machine x ready to process job_y, then job_y is immediately processed in machine x and the value of idle time of machine x is equal to zero.

11) Adding new step 5.5 to step 5 of IWD (original) algorithm. It is required to update values of $t_{machine}$; t_{job}, and $stage$. For IWD that moves from node (i, j) to node (x, y), updating those values was done in accordance with :
   - $t_{machine}(x) = t_{machine}(x) + p(x, y)$, if $x = 1$ and $t_{machine}(x) = \max\{t_{machine}(x), t_{job}(y)\} + p(x, y)$, if $x \neq 1$,
   - $t_{job}(y) = t_{machine}(x)$,
   - $stage(y) = stage(y) + 1$.
12) The best total solution T^{TB}, and the best iteration solution T^{IB} of IWD algorithm for flow shop scheduling problem is $makespan$ of built schedule that is computed by $makespan = t_{machine}(m)$, where m is number of machines. Once the solution was built, $makespan$ was computed for every built schedule whose value is $t_{machine}(m)$ – the last machine perform processing.
13) Quality function $q()$ is a minimal function. Therefore, the formula $T^{IB} = argmax_{\forall T^{IWD}} q(T^{IWD})$ will produce the minimal value of makespan. In accordance with the objective of scheduling, either T^{IB} or T^{TB} the solution is makespan with minimal value.
14) Adding step to reinitialize $soil(i, j)$ as done in the implementation of IWD algorithm for traveling salesman problem [7]. Based on characteristic of the flow shop scheduling, reinitialization is done every 20 iterations, according to: $soil(i, j) = InitSoil * 0.001$ for any $(i, j) \in T^{TB}$, and $soil(i, j) = InitSoil$ for others. After 20 iterations, soil of all paths are reset to $InitSoil$ except one of paths that built best total solution. For these paths, $soil$ is initialized by the value of $InitSoil$ times 0.001. This multiplication is required to make paths of best total solution has $soil$ with fewer value than one of other paths, so that these paths tend to be selected by IWD's as the flowing paths in the next iteration.
15) Termination criteria is defined by the maximum iteration.

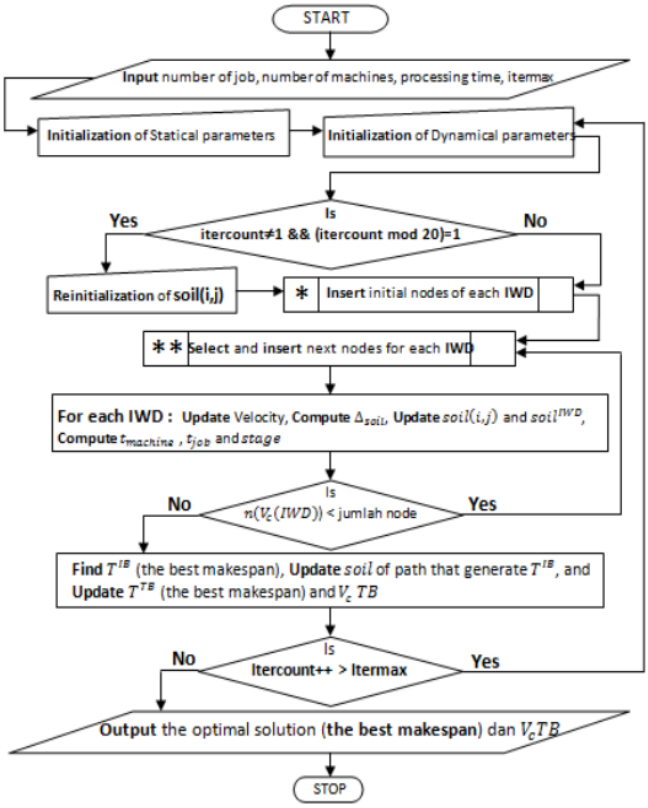

Fig. 4. Flowchart of the adjusted IWD Algorithm for Flow shop scheduling

Considering a number of adjustments made to IWD algorithm mentioned above, so that the flowchart of the complete adjusted IWD algorithm can be described in Figure 4. Subsequently, Figure 6 and Figure 5 respectively shows the flowcharts of two dominant subprocesses of adjusted IWD algorithm in Figure 4 with (*) and (**) marks.

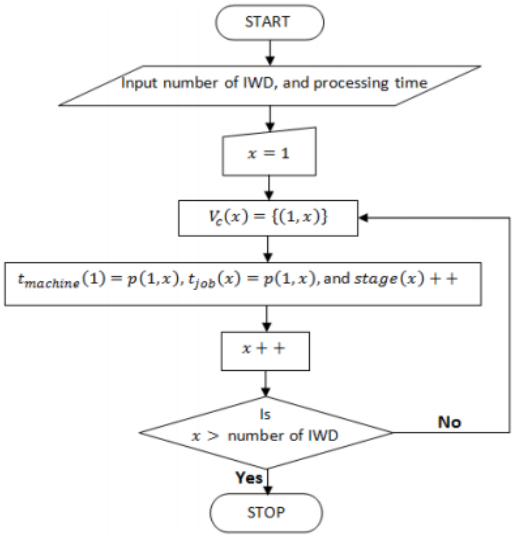

Fig. 5. Flowchart of Adding nodes

The process of adding nodes begins by accepting inputs: the number of IWD and processing time of each node. Then, initializing the counter variable x to keep track the loop. So that, the value of x is incremented by one up to the number of IWD. The first node $(1, j)$ is then added to each IWD (i.e.,

$V_C(x) = (1, j)$), and their $t_{machine}$, t_{job} and $stage$. The process will terminate whenever x is greater than the number of IWD.

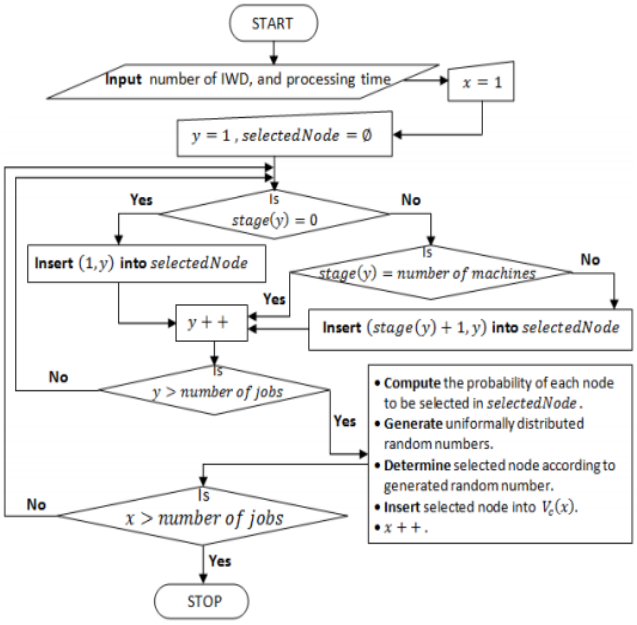

Fig. 6. Flowchart of Selecting and Inserting nodes

The beginning process for selecting the nodes is similar to the one for adding nodes. For each IWD, however, is performed several steps, such as selecting the node to be visited next, possibility of the node being selected, generatingrandom number and determinating the selected node from this generated random numbers, and the possibility of each node. Finally, inserting the selected node into $V_C(IWD)$. If all steps mentioned above had been done in all IWD's, the process terminates.

IV. EXPERIMENTATIONS AND RESULTS

In order to test the algorithm, a program was developed in Java using an application Netbeans IDE 6.9.1. The program was run on a computer with hardware specifications: processorIntel Pentium Dual-Core CPU E6600 3.06GHz, memory 4GB DDR III, and Graphic card Intel G41 Express Chipset, and the software specifications: Windows 8 (6.2) 32-bit (Build 9200) and Java Runtime Environment Version 7 update 45. Figure 7 shows an example of processing time data of each job in every machine (in this example, there are 20 jobs and 5 machines).

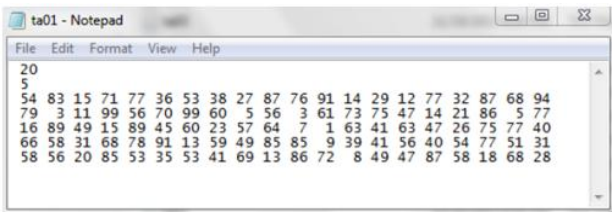

Fig. 7. An example of processing time data for 20 jobs and 5 machnes

The values of processing time can be either manually inputed or randomly generated with selected random number distribution. An illustration of this inputing stage is shown in Figure 8.

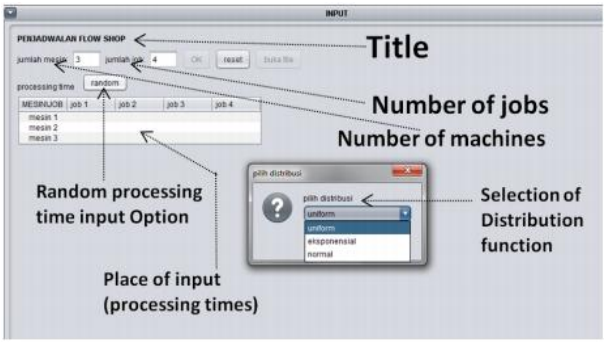

Figure 8: An illustration of inputing stage

While Figure 9 shows an example of program testing result. For simplicity reason, it uses input with 3 jobs and 2 machines (small number of job and machines).

Further testing was performed by varying the input values of scheduling – the number of machines, the number of jobs, and their processing time. In order to capable of comparing the testing result of adjusted IWD algorithm with those of results obtained from implementation of ant colony algorithm in flow shop scheduling problem done by Yagmahan and Yenisey (2010) – MOACSA (Multi-Objective Ant Colony System Algorithm), testing was conducted by dataset from Taillard (1993).

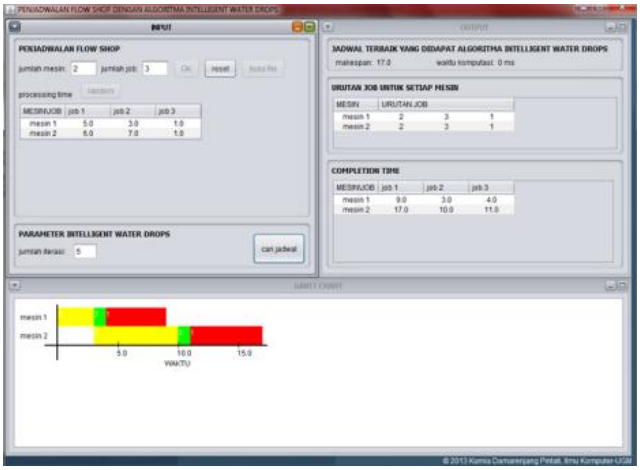

Fig. 9. An example of program testing result

To keep a fairness of comparison, parameter values of the adjusted IWD algorithm are specified similarly with ones specified in the research of IWD in TSP done by Hamed Shah-Hosseini (2009); i.e., $a_v = 1$; $b_v = 0{:}001$; $c_v = 1$; $a_s = 1$; $b_s = 0.001$; $c_s = 1$; $\rho_n = 0.9$; $IWD = 0.9$; $InitSoil = 10000$; and $InitVel = 200$. All of dataset used in the testing was downloaded from http://mistic.heig-vd.ch/taillard/problems.dir/ordonnancement.dir/ordonnancement.html.

Subsequently, the comparison between both **adjusted IWD** and **ant colony** algorithms in finding solution for flow shop scheduling problem, especially on computed makespans

is shown in Table I.

Table I
The comparison of makespans resulted by MOACSA and by adjusted IWD algorithm for Flow shop scheduling

Job	Machine	Datasets	MOASCA	Adjusted IWD
20	5	ta001-ta010	1.340	0.041
	10	ta011-ta020	0.386	0.073
	20	ta021-ta028	2.739	0.057
		Subaverage	1.488	0.057
50	5	ta031-ta040	0.545	0.034
	10	ta041-ta050	0.000	0.106
	20	ta051-ta060	0.050	0.135
		Subaverage	198	0.092
100	5	ta061-ta070	1.340	0.041
	10	ta071-ta080	0.000	0.086
	20	ta081-ta090	0.000	0.143
		Subaverage	0.241	0.088
	Total	Average	0.657	0.078

In addition, times required by Adjusted IWD algorithm to find a solution (i.e., computation time) for various pairs of job number and machine number is shown in Table II.

Table II
The computation time of Adjusted IWD algorithm for flow shop scheduling

Job	Machine	Datasets	Computation time
20	5	ta001-ta010	1.30
	10	ta011-ta020	2.50
	20	ta021-ta030	5.00
50	5	ta031-ta040	14.60
	10	ta041-ta050	32.20
	20	ta051-ta060	63.40
100	5	ta061-ta070	110.40
	10	ta071-ta080	226.50
	20	ta081-ta090	465.60

V. CONCLUSION

The IWD algorithm introduced by Hamed Shah-Hosseini (2009) was successfully implemented for solving flow shop scheduling problem by adjusting a certain number of steps (i.e., adding both statical and dynamical parameters, defining the updating mechanism for parameters $t_{machine}$, t_{job}, and $stage$, etc.). Based on the testing results of the adjusted IWD algorithm (especially $makespan$ of scheduling), and the comparison with ones obtained by ant colony algorithm for flow shop scheduling shown in Table I, on average the adjusted IWD algorithm gave better solution (more optimal) than ant colony algorithm had.

REFERENCES

[1] A., Rabanimotlagh, 2011. *An Efficient Ant Colony Optimi-zation Algorithm for Multiobjective Flow Shop Schedu-ling Problem*. WASET, 51, 127-133.

[2] B., Yagmahan, and M.M., Yenisey, 2008. *Ant Colony Optimization for Multi-Objective Flow Shop Scheduling Problem*. Computers and Industrial Engineering, 54, 411-420.

[3] B., Yagmahan, and M.M., Yenisey, 2010. *A Multiobjective Ant Colony System Algorithm for Flow Shop Scheduling Problem*. ESWA, 37, 1361-1368.

[4] C., Rajendran, and H., Ziegler, 2004. *Ant-Colony Algori-thms for Permutation Flowshop Scheduling to Minimize Makespan/Total Flowtime of Jobs*. EJOR, 155, 426-438.

[5] E., Taillard, 1993. *Benchmark for Basic Scheduling Pro-blems*. EJOR, 64, 278-285.

[6] H., Boukef, M. Benrejeb, and P., Borne, 2007. *A Proposed Genetic Algorithm Coding for Flow-Shop Scheduling Problems*. IJCCC, 3, II, 229-240.

[7] H., Shah-Hosseini, 2007. *Problem Solving by Intelligent Water Drops*. IEEE Congress on Evolutionary Computa-tion, Singapore.

[8] H., Shah-Hosseini, 2009. *The Intelligent Water Drops Algorithm: A Nature-Inspired Swarm-Based Optimiza-tion Algorithm*. IJBIC, 1/2, 1, 71-79.

[9] K.R., Baker, and D., Trietsch, 2009. *Principles of Sequencing and Scheduling*. John Wiley Sons, Inc., New Jersey.

[10] M.F., Tasgetiren, Q.K., Pan, P.N., Sganthan, and A.H.L., Chen, 2011. *A Discrete Artificial Bee Colony Algorithm for the Total Flowtime Minimization in Permutation Flow Shops*. Information Sciences, 181, 3459-3475.

[11] M.H.A., Ashkezari, N.S., Pour, and H.M., Andargoli, 2012. *An Ant Colony System for Solving Fuzzy Flow Shop Scheduling Problem*. International Journal of Engineering and Technology, 1, 2, 44-57.

[12] M.L., Pinedo, 2008. *Scheduling Theory, Algorithms, and Systems*. 3rd edition, Springer, New York.

[13] P., Bruker, 2007. *Scheduling Algorithms*. 5th edition, Springer, 5th edition, New York.

[14] Z., Cickova, and S., Stevo, 2010. *Flow Shop Scheduling using Differential Evolution*, Management Information Systems. 2, 5, 008-013.