Hybridized Ant Colony System for Tasks to Workstations Assignment

Adrian SERBENCU, Viorel MINZU
Department of Automation and Electrical Engineering,
“Dunarea de Jos” University of Galati, Romania
viorel.minzu@ugal.ro, adrian.serbencu@ugal.ro

***Abstract*— Ant Colony System is a well-known metaheuristic used to solve combinatorial optimization problems that is not intrinsically prepared to deal with precedence constraints. The work reported here is the continuation of the results presented in a previous paper that proposed an Ant System algorithm devoted to Tasks to Workstations Assignment problem. A special technique was developed in order to increase the effectiveness of precedence constraints treatment. On the one hand the contribution of this paper consists in the amelioration of this technique. On the other hand, the Ant System algorithm is hybridized with a local descent deterministic algorithm that contributes greatly to the avoiding of solutions bias. The results of the hybridized Ant System algorithm have proved the effectiveness of the proposed way to treat the precedence constraints**

Keywords*— *Ant Colony System, metaheuristic, combinatorial optimization, Tasks to Workstations Assignment

I. Introduction

Many optimization problems have been solved using an Ant Colony System (ACS) that is a well-known nature-inspired metaheuristic proposed in ([1], [2]). The common part of ACS that solves optimization problems can be identified as Ant System (AS) algorithm. This one is an iterative process that constructs solutions using components whose nature depends on the treated problem. This construction is based on pheromone information that accumulates the experience of the algorithm and represents the communication "tool" between virtual ants. The problem of ACS is the fact that it isn't intrinsically prepared to deal with precedence constraints. That is why ACS was initially applied to task-scheduling problems without precedence constraints like single machine scheduling problem [3].

Papers like [4] and [5] have analyzed the impact of precedence constraints over the algorithm's performances. The paper [4] shows that the performances of Ant System algorithm are reduced for some combinations between problem model and AS algorithm. In these cases, some *search bias* can be reported related to the problem model used in solutions' construction. Some strategies to avoid or reduce the search bias are mentioned.

The work presented here refers simultaneously to two aspects: a specific optimization problem and a specific tool to solve it, namely ACS. The optimization problem, called Tasks to Workstations Assignment (TWA) that is practically equivalent to Simple Assembly Line Balancing problem (SALBP) has been solved in the last decades using different techniques (see for example [11]), including Ant Colony Optimization (ACO). There are a lot of papers in the literature treating special cases of Assembly Line Balancing Problem (ALBP). Many of these papers have used ACO. For example, in paper [6], an ACO algorithm is proposed for solving simple and U-shaped ALBP. The paper is one of the first attempts to show how ACO can be used to solve U-shaped ALBP. Paper [7] has proposed few ACO algorithms to solve the single-model U-type ALBP. A comprehensive experimental study is presented, in which the performance of the proposed algorithms is compared with the best algorithms reported in the literature. The mixed-model ALBP has been considered in [8]. In this case, the optimization problem consist in minimizing the balance delay and the smoothness index for a given cycle time (MALBP-I). A multi-objective ACO algorithm has been proposed to solve this problem.

But a real assembly process has more specific technological constraints. Generally speaking, the tasks duration and the precedence constraints between tasks are not sufficient data to define a real ALBP. That is why; a real ALBP needs more data to describe the assembly process, like in paper [9]. This paper has proposed the Time and Space constrained Assembly Line Balancing Problem (TSALBP). One of its variants was considered and a basic model has been studied. Moreover, an ACO algorithm has been proposed including some ideas that have offered good results with simple balancing problems.

Because, more often, a manufacturing line doesn't carry out only assembly tasks, including also machining tasks, the authors of this work prefer to refer to the problem defined bellow as TWA. In this way the problem can express more general situations.

The work reported here is the continuation of the results presented in [10] that proposed an Ant System algorithm devoted to Tasks to Workstations Assignment problem. Taking into account the two aspects mentioned before, i.e. the optimization problem and the ACS, one can ask "what the emphasized aspect is". This paper deals with the amelioration of the ACS effectiveness in treating the precedence constraints, while the TWA problem offers only the opportunity to do this.

A contribution of this paper consists in a specific technique to treat the precedence constraints within ACS. On the other hand, the main contribution is the hybridization of Ant System algorithm with a local descent deterministic algorithm that contributes greatly to the avoiding of solutions bias.

Section II gives a brief statement of the well-known TWA problem that is interesting from two different points of view. First of all it is a problem having many applications in the field of production system, manufacturing and assembly system as well. It is one of the more simple statement of TWA, but is a very difficult problem because involves precedence constraints, the second point of interest. Because ACS optimization is described in many papers and all the implementations have many points in common, Section III describes the specific points of ACS used by TWA problem and precedence constraints treatment. Emphasize is placed on *tasks treatment list* (technique introduced in [10]), the definition of heuristic information and the techniques usually used to avoid the solutions bias. Section IV is devoted to the description of the local descent deterministic algorithm used to hybridize ACS. The computational results are presented in section V that is mainly a comparison between the two versions of ACS, the original one and the hybridized one. The two versions are used to solve a number of TWA problems of different dimensions. The Conclusion section underlines the effectiveness of the presented approach.

II. Tasks to Workstations Assignment

The *Tasks to Workstations Assignment* (TWA) problem [11] consists in determining the workstation to which any task of an assembly process is affected. A general statement of this problem, already presented in a previous paper, uses the following elements:

- The production line may be considered as being a sequence $W_i, i=1,\ldots,M$ of operational workstations (machines)

- For a given product, the assembly process is characterized by a set of n tasks: $\Theta=\{\theta_1,\theta_2,\ldots\theta_n\}$. For each task θ_i, its execution time $t_i, i=1,\ldots,n$ is known and it is considered independent of the workstation to which it is assigned.

- The set of tasks assigned to the workstation W_i is denoted by P_i. A proper assignment means to establish the sequence $P_i, i=1,\ldots,M$ that is equivalent to a partition of T that meets the precedence constraints given by an oriented graph $G(\Theta, U)$, with $U \subset \Theta\times\Theta$.The partition and the precedence constraints are expressed by (1) and (2).

$$\Theta=\cup_{i=1}^{M}P_i, \quad P_i \cap P_{j\neq i}=\varnothing; \tag{1}$$

$$(\forall(\theta_i,\theta_j)\in U, \text{ with } \theta_i\in P_i \text{ and } \theta_j \in P_j) \Rightarrow i\le j \tag{2}$$

For known input data, $(M, n, t_1,t_2,\ldots,t_n, \theta_1,\theta_2,\ldots,\theta_n, U)$, there are many proper assignments. Usually, the assignment that minimizes an objective function is chosen. In this paper, as in many others, the optimum criterion is to minimize the production line's cycle time. We consider that the work content of a workstation is the sum of its execution times. The line's cycle time is the maximum work content of its workstations. Because it is a graph-partitioning problem [11] that satisfies an optimum criterion, TWA is an optimization problem that is obviously NP-hard.

III. Some aspects of ACS for solving TWA

The Ant System (AS) algorithm that is the practical expression of ACS was originally proposed to solve combinatorial optimization problem [1],[2], for which the solution can be represented as a path in a graph. The algorithm uses virtual ants that build up problem's solutions, at any iteration. At each step of solution construction, an ant selects a solution component and adds it to partial solution. The construction stops when a feasible solution is obtained. When selecting a component to add to partial solution, the ant uses two kinds of information [1][2]. The first one is specific to AS algorithm and is related to the "pheromone", the specific means of communication between ants. The second one is the heuristic information, which is specific to each problem.

When solving TWA problem, a solution is a complete assignment of tasks to workstations. A possible strategy of AS algorithm is that who constructs the sequence of workstations. At each step, the ant #*k* has to select a task θ that will be assigned to the current workstation j. In order to build up a feasible solution, all the predecessors of task θ should have already been assigned to workstations i with $i \le j$. To select such a task θ can be an operation taking much time owing to the check of precedence constraints. *The other possible strategy of AS algorithm is to examine all the tasks in a specific order and assign to each one a workstation*. This specific order established before the assignment guarantees that the precedence constraints are met. The selected workstation index must be greater than or equal to those of its predecessors. In the paper [10], the authors have presented a way to engender and use such a specific order called *tasks treatment list* (TTL). This second strategy was used in the work presented by this paper. For the selection of the workstation *j* that will be affected to the task θ, the ant #*k* calculates and uses the probabilities $p_k(\theta, j)$ given by (3):

$$p_k(\theta,j)=\frac{[\tau(\theta,j)]^{\alpha}\cdot[\eta(\theta,j)]^{\beta}}{\sum_{u\in J_k(t)}[\tau(\theta,u)]^{\alpha}\cdot[\eta(\theta,u)]^{\beta}} \tag{3}$$

The "pheromone" specific to ACS, denoted here by $\tau(\theta, j)$, is an effectiveness measure of the decision to affect the workstation *j* to task θ. Another measure of the effectiveness in taking this decision is $\eta(\theta, j)$, that is the heuristic information. The set $J_k(\theta)$ is the set of candidate workstations that meet the precedence constraints, taking into consideration the tasks already treated. The relative balance between pheromone and heuristic information is tuned by two parameters of the algorithm α and β (α>0 and β>0).

The heuristic information may aggregate some different aspects: the distance between the first workstation that meets the precedence constraints and the current workstation *j*, the current work content of workstation *j*, and other information concerning the partial constructed solution.

A. Tasks Treatment List

In accordance with the chosen strategy, in the solution construction process, the tasks are treated in a previously computed sequence such that the precedence constraints are met implicitly. This strategy of treating the tasks can save a lot of computational time. The paper [10] gives the details concerning the generation and the way to use generic TTL by AS algorithm. The notion of rank of a node in graph G is the key of TTL generation. In Appendix 1, the details of the complete definition and generation of these lists are given (using some elements extracted from paper [10]). A specific TTL is randomly derived from the generic TTL.

Such a list has a useful property for our algorithm that is *any task has a place in the list after all its predecessors*. For the graph depicted in Fig. 1, a TTL may be:

[1, 17, 6, 18 | 3, 7, 2| 4| 5| 8| 10, 11, 9| 12| 13| 14| 15| 16].

Obviously, there are many TTL for a given graph G. In the same paper it is shown that the use of a unique TTL may induce a search bias, because current choice influences a later one in two ways. Firstly, when the current task θ is placed on workstation *j* than all its successors must be assigned to workstations with indices at least *j*. Secondly, the probability of placing other task, in a following step, on the same workstation *j* is depending on the heuristic information.

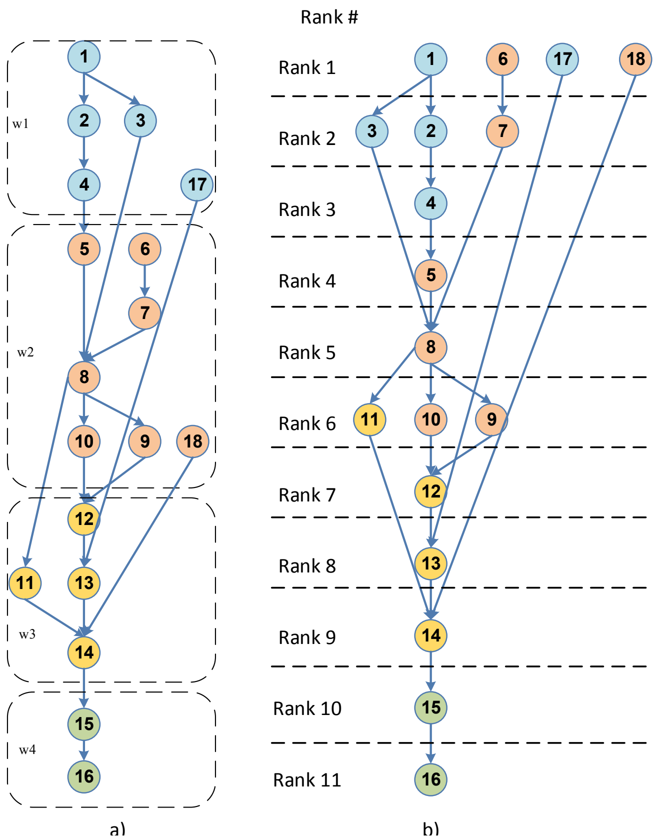

Fig. 1. a) Example of optimal tasks allocation ; b) Rank of nodes

In its turn, this one depends on the current workstation load, which includes the duration of the task *j*. Moreover, this type of influence involves also pairs of tasks that are indifferent from precedence constraints point of view.

This bias can be partially eliminated, if every ant chooses its own TTL. In the proposed AS algorithm, we used a procedure that randomly engenders a TTL. The ant family works iteratively and engenders solution generations for TWA problem. *At each generation, every ant establishes randomly its own TTL by calling this procedure*. Hence, the bias is avoided even at the level of each ant.

B. Search Biases generated by AS Algorithm

Another type of bias that influences AS performance, in problem with constraints, is induced by the model of pheromone and the model of problem. The concept of *competition-balanced system* (CBS) was defined in [4]. In our case, the combination of AS with the model of TWA is not CBS. That is why the AS algorithm manifest the *effect of second-order deception* (*sode*) introduced in [10], [4]. This effect was proved on AS algorithm that doesn't use heuristic information. When the solution construction is guided by the usage of heuristic information, this kind of bias diminishes significantly. Another way to reduce *sode* is to introduce a local search procedure at the end of the solution construction. In this work, both ways to reduce *sode* were used. The heuristic information proposed in our implementation of AS is defined hereafter:

$$\eta(\theta, j) = \begin{cases} c/(j+1-j_{\min}), \text{if } \theta \text{ can be placed on } W_j \\ \tau_0 \end{cases}, \quad (4)$$

In equation (4), the value j_{min} is the minimum index of a workstation containing a predecessor of task θ. The values of *c* and τ_0 are constant. The task θ can be placed on workstation *j*, if the work content of this one allows adding the value t_j, without exceeding an established maximal value. The maximum work content is a very important parameter for the solutions' construction.

The proposed algorithm to solve TWA is in fact a hybridized AS algorithm, because at the end of the current generation, a local search procedure is applied to each solution constructed by a virtual ant. The local search procedure is a deterministic one, trying to diminish the cycle time of a given solution.

IV. LOCAL DESCENT DETERMINISTIC ALGORITHM

The AS algorithm works iteratively and engenders, at each generation, *N* solutions (*N* being the number of virtual ants) for the given TWA problem. As it is well-known, the cyclic part of AS algorithm has the phases given below:

1. Construction of the solutions;
2. Evaluation and sorting of solutions;
3. Pheromone updating;

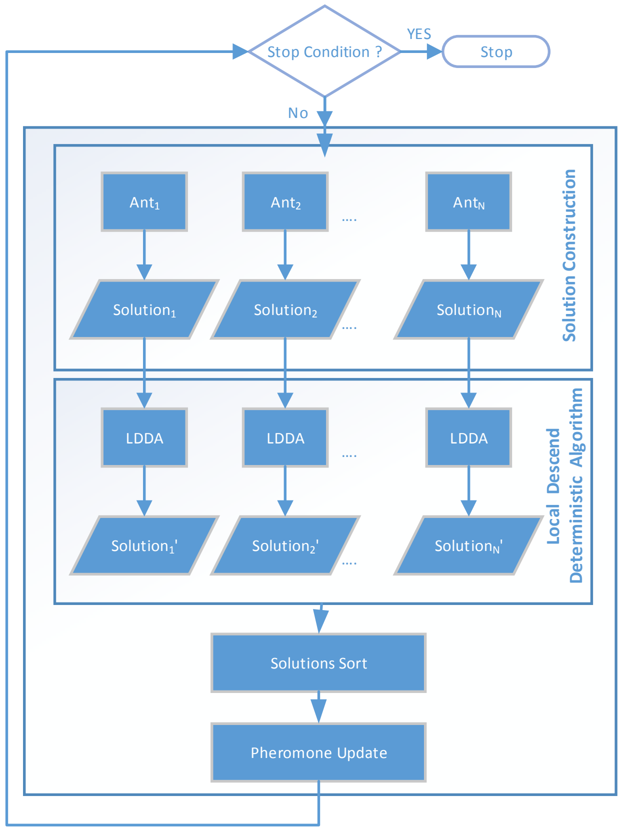

Fig. 2. Hybridized Ant Colony System

In the proposed implementation, a local descent deterministic algorithm (LDDA) is integrated between the first and the second phases. Fig. 2 shows the structure of the hybridized ACS (see also [12]). The LDDA has as input a solution constructed by an ant and tries to modify this solution in a deterministic way, in order to decrease its cycle time. These eventually modified solutions are the subject of the next phases of ACS.

The main idea of LDDA is to decrease the cycle time of a given solution - that is a TWA - by moving a task from a workstation to a neighbor one, to left or to right.

The general structure of LDDA is given by the Fig. 3. In this algorithm, a solution of TWA is equivalent to a partition of Θ. LDDA uses two shift operators, shift-left and shift-right, and generates only valid solutions for TWA problem (see [10]). The set of tasks that may be shifted to left, in relation to a workstation W_m, may be considered as a left frontier of W_m and is denoted by L_l. In the same way acts the shift-right operator. A task belonging to the left frontier has all its predecessors in the workstations $W_1,\dots, W_{m-1}$. A task belonging to the right frontier, denoted by L_r, has all its successors in the workstations $W_{m+1},\dots, W_M$.

In our implementation, a recursive version of LDDA was considered. If the new cycle time has decreased, we have a new TWA of the production line to whom it can be applied once again LDDA, and so on. The process stops when the cycle time can no more be decreased.

```
**LDDA**(*solution of TWA*)
begin
- Ascertain the index $m$ of the maximum work content workstation described in solution of TWA.
- Construct the list $L_l$ of tasks belonging to $W_m$ that can be shifted to $W_{m-1}$ and the destination list $D_l$ with the same length as $L_l$ having all elements equal to $m-1$.
- Construct the list $L_r$ of tasks belonging to $W_m$ that can be shifted to $W_{m+1}$ and the destination list $D_r$ with the same length as $L_r$ having all elements equal to $m+1$.
- Construct the lists:
  $L \leftarrow [L_l\ L_r]$; $D \leftarrow [D_l\ D_r]$;
 for all the tasks $L(i)$ belonging to $L$
     - Shift the task $L(i)$ to the workstation $D(i)$;
     - Calculate the new cycle time;
     - Memorize the effect of this shifting;
 end
- Choose the task $L(i_0)$ that involves the maximum decreasing of the cycle time.
 if $L(i_0)$ exists
     - Move the task $L(i_0)$ to the workstation $D(i_0)$.
     - Replace solution by the new solution
 end
stop
```

Fig. 3. Description of LDDA

V. Computational Results

The ACO algorithm for TWA problem was implemented in two versions: without and with LDDA.

We have chosen 17 instances of TWA problem with different sizes, part of them used also in [10]. These problem instances were solved by the two versions of ACO algorithm. The best Cycle Time (CT) value actually found at the end of the running (denoted by Gb, as Global best) has to be compared with the optimal solution of the problem, in order to evaluate the efficiency of the proposed algorithms. For large size test problems having tens of tasks, knowing the optimal solution is definitely a big matter. The 17 instances of the TWA problem that we used in our tests have been generated using a small size problem ($n_o \leq 15$) whom optimal solution is known. If the latter problem has the precedence graph G_0, this is replicated several times and some connecting arrows are adequately added in order to obtain a precedence graph with n tasks (n is multiple of n_0). Obviously, the sequence $t_1,t_2,\dots,t_n$ is obtained by repeating the sequence $t_1,t_2,\dots,t_{n_o}$ the same number of times. Finally, one can conclude that the two problems have the same optimal CT and the optimal assignment is completely determined. So, a generated instance of the test problem has obviously certain regularity, but this one is not anyhow exploited by the algorithms. The computational complexity remains the same.

The problem name is coded by twaxxx-yy, where xxx = tasks number and yy = workstations number.

The two algorithms have been implemented as MATLAB programs. The running of the algorithms is characterized by the following parameters: the number of ants 20; α=1; β=2; pheromone evaporation parameter (1-ρ) =0.9. The function (4) that calculate the heuristic information uses the constant values c=2 and $\tau_0 = 0.01$. The values of all these parameters have been tuned experimentally, such that the two variants of ACS should produce good solutions in a reasonable execution time, for all the test problems. The first two ants of each generation plus the two *Best so far* are used to update the pheromone matrix.

In the first experiment, the both algorithms have run a total number of 300 generations. This value is big enough, for the two versions of ACS algorithm, to ensure the convergence toward the optimal value of the Cycle Time (CT). The TABLE I shows, for each problem, the best result among 30 runs. This table is the first comparison between the two algorithms. For each problem instance, this table indicates the following values: the minimum value of CT (denoted Q^*), the Gb value, the generation number (Gen) at which Gb is reached and the number of fitness function evaluations (Nffe) when Gb is reached. The first analysis of TABLE I points out the following remarks:

- For all the optimization problems, the two ACS versions converge to a Gb solution within 150 generations
- ACS with LDDA is faster in terms of number of generations, but it uses a much bigger number of fitness function evaluations. On the other hand, the global best value is generally speaking nearer to the optimal value.

In TABLE II, the columns "min", "max" and "avrg" give respectively the minimum, the maximum and the average of the Global best values over the 30 independent runs. From this table, it results that ACS with LDDA gives better Gb value for 6 problems and the same Gb value for the other 11 problems. For the same algorithm, the ***max*** and ***avrg*** values are also closer to the optimal value of the problem. The ***avrg*** values for the 30 runs are smaller for 13 problems and the same for the other 4 problems. This is an empirical evidence that ACS with LDDA works better.

In order to facilitate a statistical comparison between the two algorithms based on the average errors (see [16]), we need to introduce a score (numerical measure) for a given solution, such that the score should be greater for a better solution. The value of *CT* can't be used because it decreases for a better solution. For any solution returned by our algorithms, we propose the following quality score:

TABLE I. COMPARISON BETWEEN THE BEST RESULTS OF THE TWO ALGORITHM VERSIONS

Problem	Q^*	**ACO without LDDA**			**ACO+LDDA**		
		Gb	***Gen.***	***Nffe***	***Gb***	***Gen.***	***Nffe***
twa60-15a	75	80	26	520	80	25	980
twa60-15b	100	115	57	1140	110	59	2360
twa60-15c	275	285	32	640	285	19	760
twa80-20a	200	220	1	20	220	1	40
twa80-20b	130	140	1	20	140	1	40
twa80-20c	220	250	52	1040	220	57	2280
twa80-20d	200	210	20	400	210	19	760
twa80-20e	200	210	19	380	205	6	240
twa100-20a	150	170	50	1000	160	1	40
twa100-20b	300	310	15	300	310	13	520
twa100-20c	620	690	48	960	660	64	2560
twa100-20d	150	160	33	660	155	25	980
twa120-15a	150	160	13	260	155	13	520
twa120-24a	150	170	50	1000	170	1	40
twa120-24b	300	310	19	380	310	13	520
twa120-24c	620	690	45	900	670	55	2200
twa160-20	400	420	19	380	420	2	80

TABLE II. COMPARISON USING AVERAGED VALUES

Problem	**Q***	**ACO without LDDA**			**ACO+LDDA**		
		min	***max***	***avrg***	***min***	***max***	***avrg***
twa60-15a	75	80	85	84,5	80	85	81.00
twa60-15b	100	115	115	115	**110**	115	111.0
twa60-15c	275	285	345	323,5	285	320	290.16
twa80-20a	200	220	240	223	220	230	223
twa80-20b	130	140	140	140	140	140	140
twa80-20c	220	250	250	250	**220**	250	240
twa80-20d	200	210	210	210	210	210	210
twa80-20e	200	210	215	212	**205**	215	211.6
twa100-20a	150	170	180	175	**160**	180	169.67
twa100-20b	300	310	320	311	310	310	310
twa100-20c	620	690	720	694	**660**	690	681
twa100-20d	150	160	170	167	160	170	164,3
twa120-15a	150	155	155	155	155	155	155
twa120-24a	150	170	180	175	170	180	174
twa120-24b	300	310	320	312	310	320	311.3
twa120-24c	620	690	700	695	**670**	670	670
twa160-20	400	420	420	420	420	420	420

$$\text{score} = \frac{Q^*}{\text{CT value}}. \qquad (5)$$

where the CT value may be the *min*, *max* or *avrg* value of the CT. For the optimal value, the score is equal to 1 and, for the other values, the score is less than 1. For our problem, it holds $Gb \geq Q^*$. Hence, we have:

$$0 < \text{score} \leq 1$$

Using the quality score defined by (5), the data from TABLE II is converted in TABLE III that allows us to calculate two statistic parameters. Obviously, the *min* and *max* columns of TABLE II generate respectively the *Best* and the *Worst* columns of TABLE III.

To evaluate the performance of the algorithms, as in paper [16] the average errors of the best and mean score are computed as below:

$$AEB = \frac{1}{L}\sum_{i=1}^{L}(1 - best_i) \times 100; \quad AEM = \frac{1}{L}\sum_{i=1}^{L}(1 - m_i) \times 100,$$

where *AEB* is the Avg. error of Best score and *AEM* is the Avg. error of Mean score, *L* is the number of problems (17 in our case), $best_i$ and m_i are respectively the best (the score of Gb)

TABLE III. COMPARISON USING QUALITY SCORES

Problem	ACO without LDDA			ACO+LDDA		
	Best	***Worst***	***Mean***	***Best***	***Worst***	***Mean***
twa60-15a	0.937	0.882	0.887	0.937	0.882	0.9259
twa60-15b	0.869	0.869	0.869	0.909	0.869	0.9009
twa60-15c	0.964	0.797	0.850	0.964	0.859	0.9478
twa80-20a	0.909	0.833	0.896	0.909	0.869	0.8969
twa80-20b	0.928	0.928	0.928	0.928	0.928	0.9286
twa80-20c	0.880	0.880	0.880	1.000	0.880	0.9167
twa80-20d	0.952	0.952	0.952	0.952	0.952	0.9524
twa80-20e	0.952	0.930	0.943	0.975	0.930	0.9452
twa100-20a	0.882	0.833	0.857	0.937	0.833	0.8841
twa100-20b	0.967	0.937	0.964	0.967	0.967	0.9677
twa100-20c	0.898	0.861	0.893	0.939	0.898	0.9104
twa100-20d	0.937	0.882	0.898	0.937	0.882	0.9130
twa120-15a	0.967	0.967	0.967	0.967	0.967	0.9677
twa120-24a	0.882	0.833	0.857	0.882	0.833	0.8621
twa120-24b	0.967	0.937	0.961	0.967	0.937	0.9637
twa120-24c	0.898	0.885	0.892	0.925	0.925	0.9254
twa160-20	0.952	0.952	0.952	0.952	0.952	0.9524
Avg. error of Best quality	7.35 %			5.55 %		
Avg. error of Mean quality	9.10 %			7.29 %		

and the mean score for the problem #*i*. The values of the two statistic parameters, indicated in the last lines of TABLE III, show that ACS with LDDA yields better results with less average errors.

VI. CONCLUSION

In this paper, an ACS algorithm for solving the TWA problem was proposed. The precedence constraints have imposed special techniques usage, in order to avoid or reduce the search bias of the solutions. The first one was the implementation of a random TTL for each ant and each generation. This technique has a general character and can be used in other problems involving precedence constraints. The second one was the hybridization of the ACS algorithm with a proposed LDDA. The computational results prove that the hybridized ACS is more efficient than the original version. A statistical analysis shows that ACS with LDDA yields better results with less average errors.

REFERENCES

[1] Dorigo M, Maniezzo V, Colorni A. Ant System: Optimization by a colony of cooperating agents. *IEEE Trans Syst Man Cybernet* Part B 1996;26(1), pp. 29–41.

[2] Dorigo M, Gambardella LM. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Trans Evolutionary Comput* 1997;1(1), pp. 53–66.

[3] Serbencu, A., Minzu, V., and Serbencu, A., An ant colony system based metaheuristic for solving single machine scheduling problem, *The Annals of Dunarea De Jos University of Galati*, 3, pp. 19–24, 2007

[4] Blum C, Dorigo M. Search bias in ant colony optimization: On the role of competition-balanced systems. IEEE *Trans Evolutionary Comput* 2005;9(2), pp. 159–74.

[5] Blum C. and Sampels M., “Ant colony optimization for FOP shop scheduling: A case study on different pheromone representations,” in Proc. Congr. Evol. Comput. (CEC), vol. 2, Los Alamitos, CA, 2002, pp. 1558–1563.

[6] Baykasoğlu Adil and Dereli Türkay, “Simple and U-type Assembly Line Balancing by Using an Ant Colony Based Algorithm”, Math. Comput. Appl. 14*(1)*, 2009, pp. 1-12.

[7] Sabuncuoglu Ihsan, Erdal Erelb, Arda Alpc , “*Ant Colony Optimization for the single model U-type assembly line balancing problem*”, IJPE, Volume 120, Issue 2, August 2009, pp. 287–300

[8] Betul Yagmahan, “*Mixed-model assembly line balancing using a multi-objective ant colony optimization approach*”, Expert Systems with Applications, Vol. 38, Issue 10, September 2011, pp. 12453–12461

[9] Joaquín Bautista, Jordi Pereira, “Ant algorithms for a time and space constrained assembly line balancing problem, European Journal of Operational Research, Vol. 177, Issue 3, 2007, pp. 2016–2032

[10] Serbencu, A., Minzu, V., and Serbencu, A., "Precedence constraints treatment in ant colony optimization"; 18th International Conference System Theory, Control and Computing (ICSTCC), 2014, Sinaia, Romania, pp. 87 - 92; ISBN 978-1-4799-4602-0

[11] Minzu V; Henrioud J.M; Stochastic algorithm for the tasks assignment in single or mixed- model assembly lines- *European Journal on automation* Vol. 32 No 7-8 October 1998 pp 831-851.

[12] V. Mînzu; L. Beldiman, "Some Aspects Concerning the Implementation of a Parallel Hybrid Metaheuristic, Engineering Applications of Artificial Intelligence", *Engineering Applications of Artificial Intelligence*, Volume 20 , Issue 7 (October 2007), Pages 993-999, Elsevier ISSN:0952-1976;

[13] Guangru Hua; Xiaoliang Fan, "An Approach of Obtaining Global and Near-global Optimal Process Plans Based on GA Considering Operations Precedence Constraints," *Genetic and Evolutionary*

Computing (ICGEC), 2010 Fourth International Conference on „ pp.308,311, 13-15 Dec. 2010

[14] Glover, F., Laguna, M., Marti, R., 2004. Scatter search and path relinking foundations and advanced designs. In: Onwubolu, G., Babu, B.V. (Eds.), New Optimization Techniques in Engineering. Springer, pp. 87–100.

[15] Blum C. and Dorigo M., “Deception in ant colony optimization,” in *Lecture Notes in Computer Science*, Ed. M. Dorigo et.all, Eds. Berlin, Germany, 2004, vol. 3172, Proc. 4th Int.Workshop Ant Colony Opt. Swarm Intell. (ANTS), pp. 119–130.

[16] Z. Beheshti, S. M. Shamsuddin, S. Hasan, "Memetic binary particle swarm optimization for discrete optimization problems", ELSEVIER, Information Sciences 299 (2015), pp. 58-84.

APPENDIX 1

TASKS TREATMENT LIST

(extracted from paper [10])

The set of task Θ can be covered by a list of sets *r*,

$$r=[I_0,I_1,\dots,I_m]$$

having the properties of a partition:

$$\Theta=\bigcup_{i=0,m} I_i,\quad I_i\cap I_{j\neq i}=\varnothing .$$

and the sets I_k are the Equal Rank Sets of the precedence graph G. The set I_k, k∊{0,1,…, *m*} contains all the tasks those associated nodes have the rank *k* in *G.* The rank of a node *x* of *G* is the maximum length of a path arriving in *x*. The maximum rank in *G* is denoted by *m*. Hence, the sets I_k, *k*=0,…, *m* are totally ordered in the list *r* .

Let's note that the tasks belonging to I_k, are *indifferent* i.e. any task of I_k has no predecessor or successor in I_k. That is why they may be called *indifference sets*.

The list *r* is called *generic tasks treatment list* (TTL).

The figure below describes the algorithm ***GenTTL*** that generates the generic TTL. The function S(Y) engenders the set of all the successors of the elements belonging to Y. The function *append*(*r*, X) creates a new list as a result of adding the elements X to the end of list *r*. The message "error" means the oriented graph G is not a precedence graph because it has a cycle.

The main property of the generic TTL is that

All the predecessors of a task $t\in I_k$, *k*∊{1,…, *m*} *belong to the set* $I_0\cup\dots\cup I_{k-1}$.

This property is the key aspect of the proposed method to treat the precedence constraints.

This list *r* of indifference sets gives the order the tasks are examined. When the algorithm is running, the generic TTL is exploited as follows:

1. Begin with *k*=0 i.e. I_0 is the current set.
2. Select by a random procedure a task *t* belonging to I_k.
3. Process the task *t* depending on the treated problem.
4. Erase the task *t* from I_k.
5. If I_k is empty, increment *k*.

The generic TTL is equivalent with a predefined tasks sequence. In fact, there are groups of positions reserved to the indifference sets that are strictly sequenced. Inside a group the tasks are randomly placed.

```
                    GenTTL
start
    Y← Θ; //the set of tasks
    k←0;
    $I_k$←Y \ S(Y)
    r←append(nil, $I_k$)
    while $I_k \neq \phi$
            k ← k+1;
            Y=Y \ $I_k$
            $I_k$←Y \ S(Y)
            r←append(r, $I_k$) // r is a list of sets $I_k$
    end;
    If $Y \neq \phi$ then "error"
end GenTTL
```