Ant Algorithms for Assembly Line Balancing

Joaquín Bautista and Jordi Pereira

ETSEIB, Universitat Politécnica de Catalunya
Avda. Diagonal 647, Planta 7, 08028 Barcelona, Spain
{Joaquin.bautista,jorge.pereira}@upc.es

Abstract. The present work is focused on the assembly line balancing design problems whose objective is to minimize the number of stations needed to manufacture a product in a line given a fixed cycle time, equivalent to a fixed production rate. The problem is solved using an ACO metaheuristic implementation with different features, obtaining good results. Afterwards, an adaptation of the previous implementation is used to solve a real case problem found in a bike assembly line with a hierarchical multi-objective function and additional constraints between tasks.

1 Introduction

The assembly line balancing problem ALBP consists on assigning the tasks in which a final product can be divided to the work stations in its assembly line. Using the classification given by Baybards [4], the assembly line balancing problems can be divided in two general categories: SALBP (Simple Assembly Line Balancing Problems) and GALBP (Generalized Assembly Line Balancing Problems).

The first set of problems (SALB) can be formulated as: find an assignment of tasks to a set of work stations (each one consisting on a worker, a workgroup or robots) with identical cycle time or production rate, from a set of elementary tasks with previously defined duration. Each task can only be assigned to one work station and a set of precedence relationships must be fulfilled. The problem is a generalization of the Bin Packing problem where precedence constraints are added. Three formulations of the objective are found in the literature:

Minimizing the number of stations given a fixed work rate, cycle time of the line; problem known as SALBP-1 which will be the problem studied in this work.

Minimize the cycle time given to each workstation to do their tasks given a fixed number of workstations, problem known as SALBP-2.

To minimize the total idle time of the assembly line, equivalent to maximize the line efficiency, with a lower and upper bound to the number of stations. This problem is known as SALBP-E

The second category, GALBP is composed by the rest of problems, including those problems showing extra constraints, different hypothesis and different or multiple objective functions. Some models in this category have been studied in the literature including problems with parallel workstations [6], task grouping, [7] and incompatibilities between tasks [1]. As the objective function for these problems are partially or completely the same function from their SALBP counterparts, we will keep the same notation for the objective.

It's important to remark that all SALBP and GALBP instances have the property of reversibility, a solution to an instance where precedence relationships between tasks have been reversed is also a solution to the original problem. Reversing an instance consists on substituting all precedence constraints of the original instance by their reverse precedence constraints; for each pair of tasks *i* and *j* where task *i* is a predecessor of task *j*, the reverse instance will have a precedence relationship where task *j* is a predecessor of task *i*, while maintaining all other constraints untouched. This property has been used in some exact approaches, [17].

Several exact approaches for assembly line balancing problems have been proposed. The most effective approaches are those based on implicit enumeration, [12], [13] and [17]. Even though branch and bound procedures perform well, problems with big dimensions found in industrial settings are usually solved using heuristics, allowing smaller running times to obtain good solutions and facilitating the adaptation to real industrial concerns not found in the academic models. Heuristic approaches have been formulated in the literature for the SALBP-2, [20], and SALBP-1 family of problems, [3], [5] and [19].

A first class of heuristic procedures applied to the problem are greedy heuristics based on priority rules [11]. Priority rules usually refer to or combine characteristics of the instances, as the processing time of tasks, the number of successors of a task, lower and upper bounds on workstations, etc. to decide which tasks should be assigned first. Annex I show a list of different priority rules found in the literature.

The greedy procedure assign tasks to a station using the priority rule to determine the most appropriate task between a set of tasks compatible with tasks already assigned until the station is filled. The set of candidate tasks is composed by those tasks whose predecessors has already been assigned, its processing time is not bigger than the remaining available time in the station and fulfill any problem specific constraints. Once the set of candidate tasks becomes empty, no other task can be assigned to the current station, the procedure creates a new station and begins to assign tasks to it.

These greedy procedures usually use a single priority rule and provide good mean results, usually improving as the number of characteristics from the instance taken into account by the rule increases, but no single rule dominates all other rules for all instances of the problem.

A second class of heuristic procedures is constituted by the GRASP (*Greedy Randomized Adaptive Search Procedure*) [3], and ACO metaheuristics, that allow the generation of several solutions due to the incorporation of randomness in the procedure. On each iteration of the greedy procedure, a task is chosen from a subset of the candidates using a probabilistic rule which may take into account a priority rule and information obtained by previous iterations.

Finally, a third class of heuristics are local search procedures. Between them: the hill-climbing procedures (HC), the simulated annealing (SA), the tabu search (TS), [16], and the genetic algorithms (GA) [3]. This class of heuristics provide alternative ways to find new solutions in a solution space limited by the neighborhood definition.

This work is divided in the following sections. In section 2 a constructive heuristic and a local search procedure for SALBP-1 problems are shown. Section 3 proposes an ACO heuristic approach to the SALBP-1 problem based on the AS heuristic [9], for the traveling salesman problem, analyzing different models of trail information maintenance and the hybridization with a local search heuristic shown in section 2. Section 4 covers a computational experiment with a benchmark instance set to the problem, [16], for the previous proposed heuristics. Finally section 5 details the adaptation of the implemented heuristics to a special real GALBP application found in the bike assembly industry to finish in section 6 with the conclusions of this work.

2 Greedy Heuristics for SALB-1 Problems

Figure 1 shows a schematic model of a constructive, greedy, heuristic to generate solutions for SALB and GALB problems.

The heuristic works as follows: tasks are selected one by one, using the associated priority rule (or rules) which identify the algorithm, from a set of tasks satisfying precedence relationships, time limitations and incompatibilities with the already constructed part of the solution. After a complete solution is found, a local search procedure can be applied to the solution to improve it.

Obviously, to define the heuristic at least a priority rule must be selected: Annex I shows a list of thirteen heuristics rules found in the literature and an example of its usage.

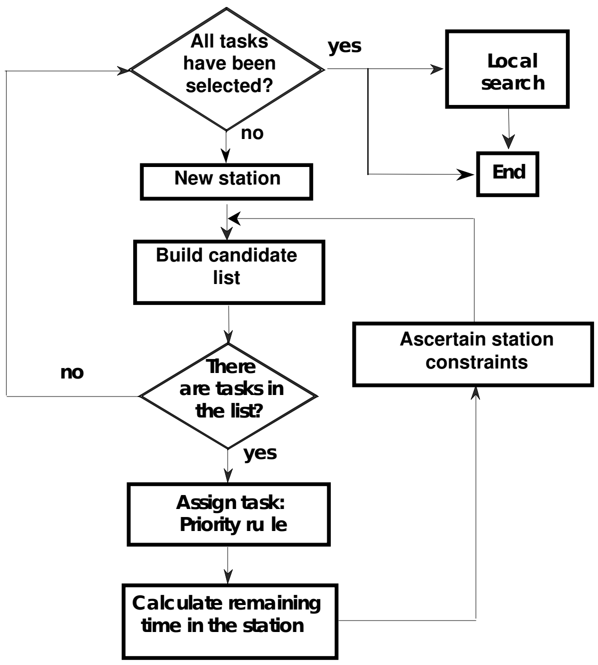
All tasks have been selected?
yes
Local search
no
End
New station
Build candidate list
Ascertain station constraints
no
There are tasks in the list?
yes
Assign task: Priority ru le
Calculate remaining time in the station

Fig. 1. Procedure scheme for obtaining solutions to SALB-GALB problems.

The final results obtained are the number of stations needed, an assignment of tasks to each station and, if needed by any other part of the algorithm, an ordered list containing when the tasks were selected.

2.1 Hill Climbing Procedure

The proposed local search procedure starts with a solution to the problem and tries to reduce the objective function of the SALBP-1 minimizing the number of stations needed. Unfortunately due to the structure of the solution space of SALBP-1, many solutions exists with the same objective function value, and usually the whole solution neighborhood have the same objective function value, making direct approaches not useful. Instead of this, the algorithm tries to achieve the goal using a different strategy consisting on minimizing the idle time in the first stations of the solution, and thus maximizing the idle time in the later stations. Eventually, and as a side effect, the last station might end up with no assigned tasks and thus it can be eliminated, reducing the objective function value.

The neighborhood is defined by two possible operators: (1) exchange the station of two tasks if the task duration which is moving to the station nearer to the initial station is greater than the task duration moving to the station nearer to the final station, and (2) moving a task from a station to a previous station. Both operators are forced to construct feasible solutions discarding any not feasible solution they might reach to, and not to accept any exchange whose solution is worse than the previous one. The algorithm is applied in such a way that once an improvement is found it is automatically applied on the solution. At the end of the procedure if the final station is completely idle the objective function is decreased.

A different approach is given by [16], consisting on a transformation of the SALBP-1 instance in a SALBP-2 instance with a fixed number of stations equal to one less than the number of stations of the original solution. Tasks assigned to the last station in the original solution are assigned to the last station of the transformed solution, and the objective is to minimize the cycle time under the cycle time constraint given to the original SALBP-1. If we obtain a solution for the transformed problem with a cycle time equal to or inferior to the cycle time given in the SALBP-1 problem, this assignment of tasks is a valid solution for the original SALBP-1 problem with smaller objective function than the original solution. The process can be repeated until no improvement is obtained.

3 Ant Colony Algorithms for SALBP-1

During the development of the ACO metaheuristic from its beginning [9], up to the last implementations [18], different variations, some slight some other more important, of the original approach have been formulated, and several different algorithms have appeared. Our approach takes the AS heuristic as presented by [10] as its starting point. In each iteration *m* ants, we will call them subcolony, build solutions to the problem. Each ant iteratively chooses tasks using the constructive approach shown in section 2, where a probabilistic rule based on a priority rule and the trail information deposited by previous ants are used to guide the search. The heuristic information, indicated as η_i, where i is a task, and the trail information, indicated as τ, are in advance indicators of how good seems to assign the task i in the current decision. As more than a single priority rule are available in the literature for this problem, we opt for assigning a different rule to each ant of the subcolony, and additionally to use the inversability property, some ants will solve the direct instance while others the reverse instance. As there are thirteen rules available from the literature and two ants of each subcolony use the same priority rule, one for the direct problem and the other for the reverse problem, each iteration, or subcolony, is formed by twenty-six ants.

The different policies to manage in which characteristics of the solution the trail information will be left and two techniques to read the information have been tested. The three trail information usage policies are: (1) The trail information is deposited between correlative tasks (i,i+1) in an ordered assignment list, (τ_{ij} meaning the existing trail information between task j and task i), called trail between pair of tasks, (2) the trail information is deposited between the task and the iteration in which the task was selected (meaning the trail information existing between the task i and the position in an ordered list of decisions j) called trail between task and position, and (3) the trail information is left between the task and its assigned station (where τ_{ij} mean the existing trail information between task j and station i), and called trail between task and station.

To keep information in such a way that it's possible to use it for ants solving the direct and the reverse instance, the trail information for reverse instances is deposited in the following ways: (1) The trail information is left between correlative pairs of tasks (i,i-1), (2) The task is left between a task and the position (number_of_tasks - chosen_position), and (3) the trail information is left between a task and the station (upper_station_bound – chosen_station).

The two ways used to read information are: (1) direct trail information reading, called direct evaluation, and (2) summed trail information reading, as [15], called accumulative evaluation.

Under the previous hypothesis, the probability of assigning task j from a set D of candidate tasks is as follows:

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha}[\eta_j]^{\beta}}{\sum_{h\in D}[\tau_{ih}]^{\alpha}[\eta_h]^{\beta}} \tag{1}$$

Where τ_{ij} and τ_{ih} will depend on the trail information updating policies and η_j, η_h the priority of each task for the given rule. Due to the differences between the range of values each priority rule can have, the heuristic information η_j is linearly normalized between 1 and the number of candidate tasks for each task i. This approach was firstly proposed in [14].

In case of accumulative evaluation, the probability to choose a task is as follows:

$$p_{ij} = \frac{\left(\sum_{k=1}^{i}[\tau_{kj}]\right)^{\alpha}\cdot[\eta_j]^{\beta}}{\sum_{h\in D}\left(\sum_{k=1}^{i}[\tau_{kh}]\right)^{\alpha}\cdot[\eta_h]^{\beta}} \tag{2}$$

Where k represent, depending on the trail information updating policy, all previously selected tasks, all previous assignment positions or all previous stations.

In both cases, α and β are two constants determining the relative influence of the heuristic information and the trail information in the ants decision process.

Additionally, the management policy to read the trail information takes into account if the direct or reverse instance of the problem is being solved, eg. if the trail information policy is to keep trail information between tasks and stations in an accumulative fashion, the reading of trail information for the reverse instance to assign the task i to the station j τ_{ij} will be the summation of trails between the station (upper_bound-j) and station (upper_bound).

The trail information is updated after each subcolony have generated and evaluated all their ants, and only the best ants for the direct and reverse problem leave trail information, following the elitist ant scheme from [8] for the ACS algorithm. For each task j, the amount of trail information left equals (3) which take into account the objective function of the solution (noted as solution) and the best solution previously obtained by the procedure (noted as upper bound of the problem).

$$\tau_{ij} = \tau_{ij} + \frac{upper\_bound\_problem}{solution} \cdot \rho \tag{3}$$

Where ρ is a parameter related to the evaporation factor used to harden premature convergence problems.

The trail information is left between task j and i, where i is the previously assigned task in case of trail between tasks policy is used, the position in which the task has been assigned in case the trail information is left between tasks and positions, or the station to which the task has been assigned in case the trail information between tasks and stations is used. To harden convergence problems from appearing, before depositing the trail information an evaporation occurs following the formula (4).

$$\tau_{ij} = (1-\rho) \cdot \tau_{ij} \tag{4}$$

The algorithm is run until the final condition is reached, which may be a fixed number of iterations or a maximum running time.

3.1 Hill Climbing Optimization Strategy

The procedures have been tested with, and without, an additional local search procedure inserted. The local search procedure is shown in the section 2.1, but as the time required to apply the local search procedure is bigger than the time spent to build solutions, the local search is only applied to the best solutions of each subcolony for the direct and reverse instances and randomly to 1% of all solutions constructed. A similar strategy was also used by [2], [8] and [15].

The number of tested heuristics is, thus, twelve depending on the three trail information management policies, the two trail information reading procedures and the use or not of the local search procedure.

4 Computational Experience

To compare the results given by the proposed procedures, we compare their results with those present in the webpage www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/

alb/index.htm by Scholl for his instance set [16], and composed of a total of 267 instances. The next table shows the differences between the best know solution and the number of optimal solutions found by each procedure, with a time limit of one and five minutes and parameters α=0.75, β=0.25 and ρ=0.1. After the results are shown, the differences between the presence of a local search procedure or not, the trail information management policy and the reading policy and running times are analyzed.

Table 1. Results obtained by each procedure. The number of optimal solutions, the mean variation between the optimum solution and the obtained solution of the optimal or best solution known is reported. The trail information management policy used is referred as task-station (TS), task-position (TP) and task-task (TT). The reading trail information procedure is refered as direct (D) or accumulative (A) and the use of the local search procedure (LS) or not (NLS).

Procedure	Optimal Sol. (1 min.)	Deviation Rate (1 min.)	Optimal Sol. (5 min.)	Deviation Rate (5 min.)
TSD-NLS	169	0.461 %	173	0.404 %
TSD-LS	172	0.436 %	175	0.379 %
TSA-NLS	162	0.493 %	168	0.45 %
TSA-LS	161	0.493 %	164	0.475 %
TTD-NLS	172	0.454 %	175	0.426 %
TTD-LS	160	0.514 %	171	0.454 %
TTA-NLS	178	0.426 %	182	0.394 %
TTA-LS	177	0.436 %	178	0.415 %
TPD-NLS	164	0.46 %	172	0.433 %
TPD-LS	165	0.472 %	171	0.44 %
TPA-NLS	180	0.39 %	182	0.379 %
TPA-LS	177	0.418 %	180	0.401 %

4.1 Presence of a Local Search Procedure

With a limited running time of one minute, four algorithms with local search perform worse than their counterparts without local search. This number even increases when the running time is limited to five minutes.

As shown, the incorporation of this element is not very attractive for the problem. The time spent in the intensive exploration of the neighborhood of a solution doesn't help the ants very much to obtain the same results exploiting the trail information. Only for the case where trail information is left between tasks and stations, the algorithm obtains better results than their counterpart without local search, and only if the reading trail information policy is the direct one, even though this difference is reduced when the algorithm is given more computation time. This is probably a side effect of the proposed local search algorithm who tend to put tasks in the first stations helping to create a fast and good trail information, than the obtained without the local search.

A different approach like the transformation proposed by [16], may lead to an improvement in the number of optimal solutions found.

4.2 Trail Information Management and Reading Policies

Even if all procedures give similar results, the trail information management policies between tasks and positions and between tasks seem to be more fitted than not the task station policy. The accumulative task to task trail information management policy which takes into account trail information between already assigned tasks and the new assigned task is the best procedure for a direct trail information reading, while the accumulative algorithms seem to be better than their direct counterparts.

In case of task station management policy, direct reading seems to be better.

4.3 Running Time

Small improvements are obtained when the running time is increased to five minutes, indicating the high quality of one minute processing. Most solutions are only one station over the best known solution even with one minute runs, showing how robust the heuristic is, even with small computation times.

5 A Real Case Study

The following problem comes from a previous collaboration between our department and a bike assembly industry from Barcelona, who facilitated data of the production line, tasks attributes, precedence relationships and cycle times for each model produced by the factory. The problem consisted on assigning one hundred and three tasks with a cycle time of 327 seconds, one bike produced each five minutes and a half, trying to minimize the number of work stations needed with the given cycle time.

The real problem had several additional differentiated characteristics from the academic model shown before, which included constraints related to incompatibilities between tasks, some tasks couldn't be done in the same work station with other tasks which needed special working conditions. That included a division of tasks between dirty hand jobs, like some jobs related to motor assembly, clean hand jobs, like embellishment tasks, and don't care jobs, those were no special consideration was kept. Additionally, the tasks possessed an additional attribute marking them as right side jobs or left side jobs, depending on which side of the chassis the task had to be performed. This attribute didn't imposed a constraint, as the chassis is moved along the line by a platform allowing this movement, but some nuisances and wasted time was caused to the worker when turning the bike. As the number of turns was hard to calculate exactly, the worker does its assigned tasks in the order he is more comfortable with, we decided to try to maximize the sum of differences between tasks done in one side of the chassis and the other side in each station.

The final objective function was a weighted function between minimizing the number of required workstations and minimizing the number of turns in the solution. As the primary objective was minimizing the number of stations, a multiplying factor was applied to minimize the significance of turns in the objective solution compared to the primary objective.

To solve the problem, the heuristics from section 3 were used, with some special subroutines designed to keep incompatibility constraints between tasks during the construction of solutions, the evaluation of the number of turn existing in the solution and a hierarchical objective function.

The local search improvement method was also modified to take care of the additional constraints and to focus the minimization of turns present in the solution, once we saw the optimal number of stations, nine, was easy to obtain by the procedures without local search.

The following table shows a bound in the number of turns (calculated as a sum of each task with a chassis side associated), the solution found by a previous MILP solution to the problem, stopped after four days of computation, and the proposed heuristics for one and five minutes runs.

Table 2. Obtained results for each procedure in the real case problem. Additionally a trivial bound to the problem and the solution obtained using a mixed integer linear programming formulation is shown.

Procedure	Turns (1 min.)	Turns (5 min.)	Procedure	Turns (1 min.)	Turns (5 min.)
Bound	52	---	MILP	---	41
ESPD-NML	38	40	ESPD-ML	42	42
ESPA-NML	40	40	ESPA-ML	42	44
PRPD-NML	38	38	PRPD-ML	42	44
PRPA-NML	38	38	PRPA-ML	40	44
POPD-NML	46	46	POPD-ML	42	42
POPA-NML	40	40	POPA-ML	42	44

The results show that the result obtained by almost every heuristic is better than the obtained by the exact procedure after a longer computation time, even if no local search procedure was used, and also very near to the known problem bound that doesn't take into account task incompatibilities or precedence relationships between tasks.

6 Conclusions

The present work has proposed several procedures based on the ACO metaheuristic for the assembly line balancing problem. After showing the available greedy procedure to solve the problem and a local search procedure, several trail information management policies and trail information reading techniques are studied. Some new ideas are also described as solving the direct and reverse instance of a problem concurrently and using several priority rules together. Finally the heuristics are modified to handle a real life case found in the bike assembly industry with positive results.

Appendix: Heuristic Rules

Nomenclature

i, j	Tasks index
N	Number of tasks
C	Cycle time
t_i	Duration of task i.
IS_i	Set of immediate successors of task i.
S_i	Set of successors of task i.
TP_i	Set of predecessors of task i.
L_i	Level of task i in the precedence graph.

Assign the task z^* : $v(z^*)=\max_{i\in Z}[v(i)]$.

Name	Priority Rule
1. Longest Processing Time	$v(i) = t_i$
2. Number of Immediate Successors	$v(i) = \lvert IS_i\rvert$
3. Greatest Number of Successors	$v(i) = \lvert S_i\rvert$
4. Greatest Ranked Positional Weight	$v(i) = t_i+\sum t_j \ (j\in S_i)$
5. Greatest Average Positional Weight	$v(i) = (t_i+\sum t_j\ (j\in S_i)) / (\lvert S_i\rvert+1)$
6. Smallest Upper Bound	$v(i) = -UB_i= -N -1+[(t_i+\sum t_i\ (j\in S_i))/C]^+$
7. Smallest Upper Bound / Successors	$v(i) = -UB_i / (\lvert S_i\rvert + 1)$
8. Processing Time / Upper Bound	$v(i) = t_i / UB_i$
9. Smallest Lower Bound	$v(i) = -LB_i= - [(t_i+\sum t_j\ (j\in TP_i)) / C]^+$
10. Minimum Slack	$v(i) = - (UB_i - LB_i)$
11.Maximum Number Successors/Slack	$v(i) = \lvert S_i\rvert / (UB_i - LB_i)$
12.Bhattcharjee & Sahu	$v(i) = t_i + \lvert S_i\rvert$
13. Kilbridge & Wester Labels	$v(i) = - L_i$

Example: Let's suppose an instance with five tasks (A,B,C,D and E) with a duration of 3,5,4,1 and 11s. respectively and a cycle time of 12s. The precedence relationships between tasks are: A precedes B, C and D, C precedes E and D precedes E. In the reverse instance B precedes A, C precedes A, and E precedes C and D, while keeping the same task duration and cycle time. The heuristic weights for each task, using rule 2, are 3,0,1,1 and 0 for the direct instance and 0,1,1,1 and 2 for the reverse instance. Using the greedy algorithm and a direct lexicographic order, for the direct instance, or reverse lexicographic order, for the reverse instance to break ties, the solution to the direct instance will have 3 stations composed by (station 1) tasks A, C and D, (station 2) task B and (station 3) task E, while the reverse instance will have 2 workstations composed by (station 1) tasks E and D, and (station 2) tasks C, B and A.

Acknowledgements. This work has been partially funded by CYCIT grant DPI2001-2169. We would also like to thank two anonymous reviewers for their suggestions to improve this work.

References

1. Agnetis, A., A. Ciancimino, M. Lucertini and M. Pizzichella: Balancing Flexible Lines for Car Components Assembly. *International Journal of Production Research* (1995) 33: 333-350.
2. Bauer, A., B. Bullnheimer, R.F. Hartl and C. Strauss: An Ant Colony Optimization Approach for the Single Machine Total Tardiness Problem. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99).* IEEE Press, Piscataway, NJ. (1999) 1445-1450.
3. Bautista, J., R. Suárez, M. Mateo and R. Companys: Local Search Heuristics for the Assembly Line Balancing Problem with Incompatibilities Between Tasks. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation, ICRA 2000*, (2000) 2404-2409.
4. Baybars, I.: A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem. *Management Science* (1986) 32 (8): 909-932.
5. Boctor F.F.: A Multiple-rule Heuristic for Assembly Line Balancing. *Journal of the Operational Research Society*, (1995) 46: 62-69.
6. Daganzo, C.F and D.E. Blumfield: Assembly System Design Principles and Tradeoffs, *International Journal of Production Research* (1994) 32: 669-681
7. Deckro, R.F.: Balancing Cycle Time and Workstations. *IIE Transactions* (1989) 21: 106-111
8. Dorigo M. and L. M. Gambardella: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation* (1997) 1(1): 53-66.
9. Dorigo M., V. Maniezzo and A. Colorni: The Ant System: An Autocatalytic Optimizing Process. *Technical Report 91-016 Revised*, Dipartimento di Electronica, Politecnico di Milano, Italy (1991).
10. Dorigo M., V. Maniezzo and A. Colorni: The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, (1996) 26(1): 29-41
11. Hackman, S.T. M.J. Magazine and T.S. Wee: Fast, Effective Algorithms for Simple Assembly Line Balancing Problems. *Operations Resarch* (1989) 37, 916-924.
12. Hoffmann, T.R.: Eureka. A Hybrid System for Assembly Line Balancing. *Management Science*, (1992) 38 (1): 39-47.
13. Johnson R.V.: Optimally Balancing Assembly Lines with „FABLE". *Management Science* (1988) 34: 240-253
14. Daniel Merkle, Martin Middendorf: An Ant Algorithm with a New Pheromone Evaluation Rule for Total Tardiness Problems, *Proceeding of the EvoWorkshops* (2000)
15. Merkle, D., M. Middedorf, and H. Schmeck: Ant Colony Optimization for Resource Constrained Project Scheduling. *Proceedings of GECCO-2000.* (2000)
16. Scholl, A.: Balancing and Sequencing of Assembly Lines, *Physica-Verlag*, Heidelberg (1999)
17. Scholl, A. and R. Klein: Balancing Assembly Lines Effectively – A Computational Comparison. *European Journal of Operational Research* (1999) 114: 50-58
18. Taillard É. D.: Ant Systems, in: P. Pardalos & M. G. C. Resende (eds.), *Handbook of Applied Optimization*, Oxford Univ. Press, (2002) 130-137
19. Talbot,F.B., J.H. Patterson and W.V. Gehrlein: A Comparative Evaluation of Heuristic Line Balancing Techniques, *Management Science* (1986) 32: 430-454.
20. Ugurdag, H.F., R. Rachamadugu, and A. Papachristou: Designing Paced Assembly Lines with Fixed Number of Stations. *European Journal of Operational Research*, (1997) 102(3): 488-501.