Two-Stage ACO to Solve the Job Shop Scheduling Problem

Amilkar Puris[1], Rafael Bello[1], Yaima Trujillo[1], Ann Nowe[2], and Yailen Martínez[1]

[1] Department of Computer Science, Universidad Central de Las Villas, Cuba
[2] CoMo Lab, Department of Computer Science, Vrije Universiteit Brussel, Belgium
{rbellop,ayudier,yaimatr,yailenm}@uclv.edu.cu, ann.nowe@vub.ac.be

Abstract. In this paper, a multilevel approach of Ant Colony Optimization to solve the Job Shop Scheduling Problem (JSSP) is introduced. The basic idea is to split the heuristic search performed by ants into two stages; only the Ant System algorithm belonging to ACO was regarded for the current research. Several JSSP instances were used as input to the new approach in order to measure its performance. Experimental results obtained conclude that the Two-Stage approach significantly reduces the computational time to get a solution similar to the Ant System.

Keywords: Ant Colony Optimization, Ant System, Job Shop Scheduling Problem.

1 Introduction

This paper introduces a multilevel approach of Ant Colony Optimization to solve the Job Shop Scheduling Problem (JSSP). In the static JSSP, a finite number of jobs need to be processed by a finite number of machines. A job is defined as a predetermined sequence of tasks, each one of those needs to be processed without interruption for a given period of time on a specified machine. The tasks belonging to the same job cannot be processed in parallel and, additionally, each job must be carried out in each machine exactly once. A feasible schedule is an assignment of operations to time slots on a machine without violation of the job shop constraints. A makespan is defined as the maximum completion time of the jobs. The main goal is the accomplishment of a schedule that is able to minimize the JSSP's makespan. Such optimum schedule is the one that minimizes the total idle time for the set of machines. According to the complexity theory [1], the JSSP is characterized as NP-hard combinatorial optimization problem. Since the achievement of exact solutions for such sort of problems is computationally unfeasible [1], different heuristic methods have been applied for solving JSSP.

Ant Colony Optimization (ACO) is a metaheuristic used to guide other heuristics in order to obtain better solutions than those generated by local optimization methods; this meta-heuristic has been successfully applied to various hard combinatorial

L. Rueda, D. Mery, and J. Kittler (Eds.): CIARP 2007, LNCS 4756, pp. 447–456, 2007.

optimization problems. In ACO, a colony of artificial ants cooperates in the search of good solutions to discrete problems. Artificial ants are simple agents that incrementally build a solution by adding components to a partial solution under construction. This computational model was introduced by M. Dorigo. Further information about this procedure can be found in [2], [3] and [4].

In the Scheduling field, ACO has effectively dealt with the Flow-shop [5], Resource Constraint Project Scheduling [6] and the Single Machine Total Tardiness problems [7]. ACO has also proven to be profitable in finding out the solutions of other permutation scheduling problems such as the Traveling Salesman [8, 9] and Vehicle Routing problems [10]. However, the application of the ACO to Shop scheduling such as the JSSP and open shop scheduling has demonstrated to be quite difficult [11] and very few papers about the ACO implementation for the JSSP can be found. The first ant system (AS) coping with the JSSP appeared in 1994 [12]; more recently, C. Blum et al. have done significant research on the application of ACO to shop scheduling problems including the JSSP [11, 13]; in 2004, M. Ventresca and B. Ombuki introduced an application of the Ant Colony Optimization metaheuristic to the job shop scheduling problem[14].

It is worthwhile to note that ACO algorithms are appropriate for discrete optimization problems that can be characterized as a graph $G = (C, L)$. Here, C denotes a finite set of interconnected components, i.e. nodes. The set $L \subseteq C \times C$ describes all of the connections (i.e. edges) at the graph (see [6] for a complete description). Every solution of the optimization problem may be expressed in terms of feasible paths across the graph.

In this paper, a new approach of ACO is developed where the underlying idea is to have ants perform the heuristic search as a two-stage process; we focus on the Ant System algorithm belonging to the ACO family because previous studies have shown that it attains the best results [14]. Several JSSP instances were used as input to the new approach in order to measure its performance. Experimental results showed the two-stage approach significantly lowers the computational time to get an Ant System similar solution.

2 Job Shop Scheduling Problem

The JSSP is made up by a finite set $\boldsymbol{J}$ of n jobs to be processed on a finite set $\boldsymbol{M}$ of m machines. Each job J_i must be executed on every machine and consists of m chained operations $o_{i1}, o_{i2},\ldots,o_{im}$ that are to be scheduled in a predetermined given order (precedence constraint).

There is a total of $\boldsymbol{N} = n*m$ operations where o_{ik} is the operation corresponding to job $\boldsymbol{J}_i$ that is to be run on machine M_k during an uninterrupted processing time p_{ik}. The workflow of each job throughout the machines is independent of the other jobs'. At a time, each machine is able to carry out a single job and, besides, each job is to be processed by a single machine simultaneously (capacity constraints). The parameter C_{max} points out the performance measure that should be minimized (longest time required to complete all jobs).

The objective is to determine the starting times ($t_{ik} \geq 0$) for each operation so as to entail a minimization of the makespan in such a way that all of the constraints are met:

$$C^*_{\max} = \min\{C_{\max}\} = \min_{feasible\ \ schedules}\{\max\{t_{ik} + p_{ik}\} : \forall \mathrm{J_i} \in \mathrm{J}, \forall \mathrm{M_k} \in M\} \quad (1)$$

Table 1 depicts an example of a JSSP instance whose graphical representation is portrayed in figure 2. Notice that each node represents an operation. Thus, node 1 stands for the first operation of job 1, node 2 symbolizes the second operation and so on. In a general way, a node *i* represents the *(i mod (m+1))* operation of the *job (i div (m+1)) +1*.

Table 1. An example of a simple JSSP instance holding two jobs that must be processed on four machines. The data format is (machine, duration); numbers in bold refer to Figure 2.

(2,10) **1**	(1, 2) **2**	(4,7) **3**	(3, 5) **4**
(1,12) **5**	(4,6) **6**	(3,5) **7**	(2,2) **8**

2.1 Types of Schedules

According to the schedule properties, any feasible schedule can be categorized into four major kinds: inadmissible, semi-active, active, and non-delay schedules. The number of inadmissible schedules is infinite and most of them contain excessive idle times. A semi-active schedule can be obtained by shifting a schedule forward until no such excessive idle times appears.

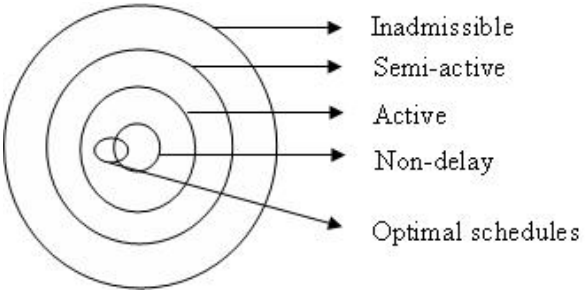


Fig. 1. The hierarchy of feasible schedules

Further improvements on a semi-active schedule can be reached by skipping some operations ahead without bringing about the latter start of other operations regarding the original schedule. However, active schedules allow no such displacement. Thus the optimal schedule is guaranteed to fall within the active schedules. Non-delay schedules build a subset of active schedules. In a non-delay schedule, a machine is never kept idle if some operation is able to be executed. It is remarkable fact that the

best schedule is not necessarily a non-delay one. However, it is easier to generate a non-delay schedule than an active one. The former may be closer to the optimal schedule even if it is not an optimal one. Additionally, there is strong empirical evidence that non-delay schedules bear solutions whose mean quality is higher than those produced by active schedules. Nevertheless, typical scheduling algorithms browse the space of all active schedules in order to assure that the optimum is taken into consideration.

3 Ant Colony Optimization (ACO)

Artificial ants are straightforward agents that incrementally make up a solution by adding components to a partial solution under construction. They are the main component in Ant Colony Optimization (ACO). In such methodology, the ants cooperate in order that good solutions to discrete problems can be found.

Ant System (AS) is the first ACO algorithm; it was introduced by means of the Traveling Salesman Problem (TSP) [7] and [9]. In TSP, we have a group of edges fully connecting the set of N cities $\{c_1, \ldots, c_n\}$; each edge is assigned a weight d_{ij} whose meaning is the distance between cities i and j. The goal is to find the shortest possible trip which comprises each city only once before going back to the starting city. When ACO is used to solve these problems, pheromone trails (τ_{ij}) are associated to the edges and denote the likeliness of visiting city j coming from city i. Initially, the ants are randomly positioned into cities. Throughout the subsequent steps, ant k computes a random proportional rule to decide which city will be visited next according to expression (2):

$$p_{ij}^{k} = \frac{(\tau_{ij})^{\alpha} * (\eta_{ij})^{\beta}}{\sum_{l \in N_i^k} (\tau_{il})^{\alpha} * (\eta_{il})^{\beta}} \qquad if \quad j \in N_i^k \quad (neighborhood \quad of \quad ant\ k) \tag{2}$$

where α and β are leveling parameters of the relative importance of the pheromone trail and the heuristic information, respectively. AS ants have a memory (tabu list) that stores visited components of their current path for preventing the chance of returning to an already visited city.

After all ants have made up their tours, the τ_{ij} values are updated in two stages. Evaporation as a fading factor of the pheromone trail is considered in stage 1, yielding lower τ_{ij} which are calculated as shown in expression (3) by using the parameter ξ, $(0 < \xi < 1)$; this step is needed to avoid the unlimited accumulation of pheromone.

$$\tau_{ij}=(1-\xi)*\tau_{ij} \tag{3}$$

Secondly, all ants increase the value of τ_{ij} on the edges they have traversed in their tours according to the expression below:

$$\tau_{ij}=\tau_{ij}+Inc_{ij} \tag{4}$$

where Inc_{ij} is the amount of pheromone dropped by all ants walking across the edge (i, j). Usually, the amount of pheromone dropped by ant k equals to $1/C_k$, where C_k is the length of the tour for ant k.

4 Ant System for JSSP

An instance of the JSSP in the ACO algorithm is represented as a graph where the nodes are connected by two kinds of edges; the nodes represent operations, that is, for N jobs and M machines, the graph will include N*M nodes; the oriented edges represent the precedence between operations belonging to the same job and the dashed edges stand for a likely path that ants can go through if the problem constraints are satisfied (See Figure 2).

In order to apply the AS algorithm, a graphical representation ***G*** of the optimization problem must be built up at first.

The meta-heuristic begins initializing the amount of pheromone in each edge of ***G*** with some positive real value c. Each ant is then placed into an initial position, which is added to its tabu list; such initial positions are randomly chosen from the possible ones, which are the first operations to execute in each job.

Every agent will independently set up a solution following the probabilistic rule 1, where the heuristic value associated to an operation j is $d_{ij} = 1/Ctime_j$, $Ctime_j$ symbolizes the completion of operation j. After the tabu list of all ants is full (a valid solution has been found), its path length will be determined and the best solution found so far will be recorded.

Next, the pheromone values are recomputed via expressions 2 and 3, where Inc_{ij} in this problem is the better scheduling found in the current cycle. This process is repeated during a given number of cycles.

In the graph displayed in Figure 2, the possible initial positions for the ants are nodes 1 and 5. If ant k chooses node 5, then the likely moves that meet the problem's constraints are either to nodes 1 or 6 and alike.

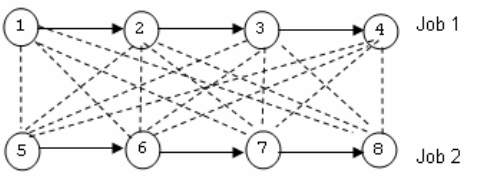
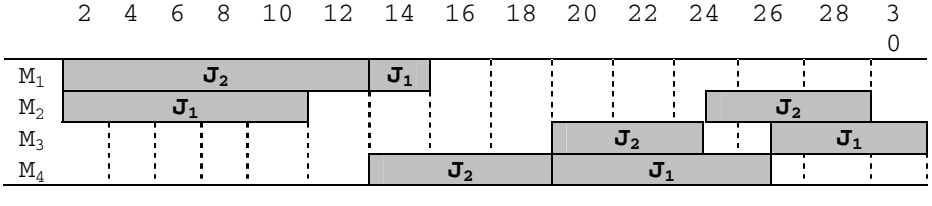


Fig. 2. A graphical representation for the 2-job and 4-machine problem instance shown in Table 1

For this problem we can find the following feasible schedules (solutions):

A: {1-2-3-5-6-7-4-8}, *B*: {5-6-7-1-2-8-3-4}, *C*: {1-2-5-6-3-4-7-8}

Table 2. Starting from solution *B*, an active schedule having a makespan of 30 is built from scratch



5 Two-Stage Ant System (TS-AS) in the JSSP

The Two-Stage Ant Colony Optimization (TS-ACO) proposed in this investigation is based on the following idea: to split the search process performed by the ants into two stages so that in the first stage, preliminary results are reached (partial solutions) that behave as initial states for the ulterior search realized during the second stage.

Determining an initial state in which the search process starts has been an interesting issue in heuristic search. Due to the well known influence the initial state has in the search process, the algorithm aims to approximate the initial state to the goal state as much as possible. Of course, it is necessary to regard a fitting balance between the computational cost of achieving the initial state and the overall cost; in other words, the sum of the costs of approximating the initial to the goal state plus the cost of finding the solution beginning at the "enhanced" initial state should not be greater than the cost of seeking the solution from a random initial state.

More formally, the purpose is described as follows: let E_i be an initial state either randomly generated or computed by any other method without a meaningful computational cost; E_i^* is an initial state generated via approximation to the goal state by some method *M;* $C_M(E_i^*)$ indicates the cost of reaching state E_i^* from E_i through method *M* and $CC_{HSA}(x)$ is the computational cost of finding a solution from state *x* utilizing a Heuristic Search Algorithm (HSA). Hence, the objective expression is held so that $C_M(E_i^*) + CC_{HSA}(E_i^*) < CC_{HSA}(E_i)$.

In the TS-AS proposed here, the procedure for calculating E_i^* and the *HSA* are both the AS algorithm, so the objective is $C_{AS}(E_i^*) + CC_{AS}(E_i^*) < CC_{AS}(E_i)$. Since AS is used in both stages, the difference between them is computed by assigning different values to some parameters of the model during each stage. A ratio *(r)* is introduced in order to measure the relative assignment of the values to the parameters of the algorithm in both stages; *r* indicates the portion of the overall search to be realized at the first stage. For instance, if $r = 0.3$, it means that the first stage will comprise 30% of the overall search and during the second stage, the remaining 70% shall be carried out (an example of the application of this ratio is exhibited in the next section).

Setting the value of *r* exercises a high influence in the overall performance of the algorithm. The higher value of *r,* the closer the state E_i^* will be to the goal state. As an outcome, $C_{AS}(E_i^*)$ increases and $CC_{AS}(E_i^*)$ decreases. In addition to this balance between the costs of $C_{AS}(E_i^*)$ and $CC_{AS}(E_i^*)$, the question of how much the search space was explored arises; the greater the rate *r* is, the lower the search in the second stage is due to several reasons: (I) there are less ants working, (II) the amount of cycles becomes smaller and (III) although the number of possible initial states for the second stage should become greater when *r* increases, such number is already upper-bounded by the result of the previous stage.

Therefore, a key point is to study what value of rate r is the best in order to reach the best balance between the searches performed during both stages. This value must allow:

- The minimization of $C_{AS}(E_i^*) + CC_{AS}(E_i^*)$.
- An exploration of the search space that guarantees to find good solutions.

When applying AS algorithm to the JSSP, the ants begin the search starting from random initial states; that is, in each cycle an ant commences its trip in a randomly

chosen operation and picks up the next operation through rule (1). At the beginning, no pheromone information is available to lead the search; the heuristic information alone is present.

On the contrary, the TS-AS builds up partial trips (they do not include all the nodes) in the first stage; this information behaves as an initial state for ants during the second stage of the search algorithm. In other words, instead of rebooting the search from scratch, the agents use the information available after the execution of the first stage as the starting point in the second stage.

In JSSP, the parameters whose values depend on the ratio *r* are: the number of ants (*m*), the number of cycles (*nc*) and the number of operations ($co = N * M$) that will be included in each stage.

The parameters values are assigned as illustrated right now: Let be 6 jobs and 5 machines (*J = 6, M = 5, N=30 total operations*) and the following parameters for the traditional AS algorithm: *m = 100*, *nc = 100* and *co = 30*. Setting *r = 0.3* implies that the values of these parameters for the two stage ACS are computed accordingly as: $m_1 = 100 * 0.3 = 30$, $nc_1 = 100 * 0.3 = 30$ and $cc_1 = 30 * 0.3 = 9$ for the first stage; and $m_2 = 100 * 0.7 = 70$, $nc_2 = 100 * 0.7 = 70$ and $cc_2 = 30$. It means that 30 ants will execute the AS algorithm for the time of 30 cycles, building a sequence of 9 operations. In the second stage, 70 ants will run the AS algorithm throughout 70 cycles shaping the sequence of 30 operations. This signifies that in the first stage, 30% of the ants will be seeking size-lessened solutions (because the sequence comprises only 30% of the nodes) in the 30% of the total number of cycles. In the second stage, the remaining 70% of the ants are used; they will perform the search for the 70% of the total number of cycles so as to discover full problem solutions (including all operations). Once the first stage has finished, a subset of partial solutions is picked up (denoted by *EI*) holding *cs* out of the best solutions (sequences with the best values of the objective function) found during the first stage.

The TS-AS-JSSP algorithm is outlined below:

Input: Parameters beta, rho, epsilon, cc, factor r, number of solutions in EI (cs)

Output: The best solution found.

```
S1: Set the number of ants either by input data or by using some
method depending on the number of operations.
S1: Perform Stage 1.
S1.1: Compute the parameters for the first stage:
            m1 = r * m
            nc1 = r * nc
            cc1 = r * cc
S1.2: Run the AS algorithm that performs nc1 cycles in the first stage.
S1.3: Set of trips ← Trips generated by AS algorithm in the first
stage.
S2: Perform Stage 2.
S2.1: Compute the parameters for the second stage:
            m2 = m - m1
            nc2 = nc - nc1
            cc2 = cc
            EI ← Pick up the cs best solutions from the
            set of trips.
S2.2: Run the AS algorithm (that performs nc2 cycles in the second
stage by using the elements of EI as initial states for the ants in
the second stage).
```

6 Experimental Results

Table 3 shows a comparative study between the algorithms AS-JSSP and TS-AS-JSSP using some remarkable JSSP instances found at OR-Library [17] regarding the quality of the solution and the computational cost in time. The same parameters were used in both algorithms for running the tests; that is, the number of ants (m) equals to the number of operations, $nc = 3000$, $\xi = 0.1$, $\alpha = 0.8$ and $\beta = 0.17$. Three different ratios were used for the TS-AS-JSSP algorithm: $r = \{0.2, 0.25, 0.3\}$. For every test, 10 runs on every instance were carried out and the best solution was selected. The columns of Table 3 stand for: the dataset name, the best solution reached for that dataset, the best solution found by the Ant System algorithm, the time cost (in milliseconds) for finding the AS solution, the best solution reported by the two stage approach (including the ratio used in the computation) and finally, the time cost (in milliseconds) for finding the two-stage algorithm solution.

Table 3. A comparative study between Ant System and Two-Stage Ant System

Instance	BK	AS-JSSP	Time1	TS-AS-JSSP	Time2
la01	666	666	157502	666(r=0.3)	53513
la02	660	673	144697	672(r=0.2)	74518
la03	597	627	144107	607(r=0.25)	60210
la04	590	611	144455	594(r=0.3)	53044
la05	593	593	144531	593(r=0.25)	61224
la06	926	926	510077	926(r=0.3)	180915
la07	890	897	509708	890(r=0.25)	224793
la08	863	868	508714	865(r=0.25)	216916
la09	951	951	510802	951(r=0.3)	186744
la10	958	958	508825	958(r=0.25)	178458
la11	1222	1222	1276036	1222(r=0.3)	460834
la12	1039	1039	1269386	1039(r=0.3)	450302
la13	1150	1150	1268055	1150(r=0.3)	462080
la14	1292	1292	1288142	1292(r=0.3)	456755
la15	1207	1251	1271330	1247(r=0.25)	553566
la16	945	978	930177	978(r=0.3)	353844
la17	784	797	927918	800(r=0.2)	510641
la18	848	901	938328	868(r=0.2)	480469
la19	842	892	928723	871(r=0.3)	414511
la20	902	955	933017	936(r=0.3)	354534

These experimental results prove that the two-stage approach algorithm achieved better solutions than the classic Ant System algorithm, additionally lowering the time cost for over 50%. Also, the qualities of the solutions reached by TS-AS- JSSP are higher than those exposed in [14], where the algorithms Max-Min Ant System, PFS and NFS, introduced in [14], are used.

The statistical analysis performed in order to compare the solution values for the mentioned algorithms using Monte Carlo Significance of Friedman's test=0.000 is displayed in Figure 3 a), whereas b) provides the comparison among the algorithms

	Mean Ranks
Optimum	1.50^{a}
TS-AS-JSSP(0.2)	2.80^{b}
TS-AS-JSSP(0.25)	3.10^{b}
AS-JSSP	3.45^{bc}
TS-AS-JSSP(0.3)	4.15^{c}

a) Statistical analysis for solutions values

	Mean Ranks
TS-AS-JSSP(0.3)	1.00^{a}
TS-AS-JSSP(0.25)	1.40^{ab}
TS-AS-JSSP(0.2)	2.10^{b}
AS-JSSP	4.00^{c}

b) Statistical analysis for time cost

Fig. 3. Statistical Analysis for solutions values (a) and computational cost (b)

with respect to the time needed to get the solution. Mean Ranks with a common letter denote non-significant difference according to Wilcoxon's test, proving that there are not significative differences between the AS and TS-AS in solutions values, and showing an important difference between them in the time cost.

7 Conclusions

This paper introduces a new approach to ant colony optimization to the job shop scheduling problem. It consists of the splitting of the search process performed by ants into two stages. The study was carried out with the use of the Ant System algorithm. In this approach, some parameters (number of ants, number of cycles, etc.) are assigned a different value in each stage according to a ratio r which signals the portion of the overall search that corresponds to each stage.

The algorithm's performance was thoroughly studied by using different ratio values. The best results came up when this value falls within the interval [0.2, 0.3].

This new ACO approach yields a significant reduction of the computational time cost yet preserving the quality of the solutions.

Acknowledgments

The authors would like to thanks VLIR (Vlaamse InterUniversitaire Raad, Flemish Interuniversity Council, Belgium) for supporting this work under the IUC Program VLIR-UCLV.

References

1. Garey, M.R., D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H Freeman and Company (1979)
2. Dorigo, M., et al.: The Ant System: Optimization by a colony of cooperating agents. Man and Cybernetics-Part B 26(1), 1–13 (1996)
3. Dorigo, M., et al.: Ant algorithms for discrete optimization. Artificial Life 5(2), 137–172 (1999)
4. Dorigo, M., Stutzle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
5. Stutzle, T.: An ant approach for the flow shop problem. In: EUFIT 1998, Verlag Mainz, Aachen, vol. 3, pp. 1560–1564 (1998)

6. Merkle, D., Middendorf, M., Schmeck, H.: Ant Colony Optimization for Resource Constrained Project Scheduling. IEEE Transactions on Evolutionary Computation 6(4), 333–346 (2002)
7. Merkle, D., Middendorf, M.: Ant Colony Optimization with Global Pheromone Evaluation for Scheduling a Single Machine. Journal of Applied Intelligence 105–111 (2003)
8. Dorigo, M., Gambardella, L.M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation 1(1), 53–66 (1997)
9. Bullnheimer, B., Hartl, R.F., Strauss, C.: A New Rank Based Version of the Ant System: A Computational Study. Central European Journal for Operations Research and Economics 7(1), 25–38 (1999)
10. Gambardella, L.M., Taillard, E., Agazzi, G.: MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Window. In: David Corne, Marco Dorigo, and Fred Glover, editors, New Ideas in Optimization, pp. 63–76. McGraw-Hill, London (1999)
11. Blum, C., Sampels, M.: An ant Colony Optimization Algorithm to tackle Shop Scheduling Problems
12. Colorni, A., Dorigo, M., Maniezzo, V., Trubian, M.: Ant system for Job-shop Scheduling. JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science 34(1), 39–53 (1994)
13. Blum, C., Sampels, M.: An ant Colony Optimization Algorithm for FOP Shop Scheduling: A case study on different pheromone representations. In: CEC 2002, pp. 1558–1563 (2002)
14. Ventresca, M., Ombuki, B.: Ant Colony Optimization for Job Shop Scheduling Problem. From Proceedings pf Artificial Intelligence and Soft Computing (2004)
15. Dorigo, M., Gambardela, L.M.: Ant colonies for the traveling salesman problem. BioSystems (43), 73–81 (1997)
16. Dorigo, M., Stützle, T.: ACO Algorithms for the Traveling Salesman Problem. Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications. John Wiley & Sons, Chichester (1999)
17. OR-library, URL http://web.cecs.pdx.edu/ bart/cs510ss/project/jobshop/jobshop/
18. Stutzle, T., Hoos, H.: The Max-Min Ant System. Future Generation Computer System 16(8), 889–914 (2000)