SWARM INTELLIGENCE-ACO IN JOB-SHOP SCHEDULING PROBLEMS

Abstract. The optimum solution of the production scheduling problem for manufacturing processes at an enterprise is crucial as it allows one to obtain the required amount of production within a specified time frame. Optimum production schedule can be found using a variety of optimization algorithms or scheduling algorithms. Ant colony optimization is one of well-known techniques to solve the global multi-objective optimization problem. In this report I present a solution of the classic production scheduling problem by means of an ant colony optimization algorithm. A case study of the algorithm efficiency estimated against some others production scheduling algorithms is presented. Advantages of the ant colony optimization algorithm and its beneficial effect on the manufacturing process are provided.

1.Introduction

Modern science-based and high-tech product consists of a large number of parts and assembly units; a wide range of materials nomenclature and purchased components are used in its manufacture. The number of types of parts in the product can reach several tens of thousands, which greatly complicates the manufacture and requires a special approach to its organization. The production scheduling problems are set for optimum organization of control of the production process. They are characterized as problems of providing the specified amounts of output, in the required tempo and high quality.

From the mathematical point of view, the problem of production scheduling is a complex combinatorial problem that has multiple solutions, among which it is necessary to find the solution, which is optimum in terms of some criteria. This problem can be solved exactly (mathematical programming, Gomory's cut) or approximately (heuristics, Monte Carlo method) . The main optimality criteria are: total throughput time minimization,readjustment time minimization,the minimum cost of schedule execution criteria , etc. The optimum production schedule searching can be performed using linear programming, dynamic programming, combinatorial or evolutionary algorithms .

Combinatorial optimization problems naturally arise in the industrial world all the time. Often, having suffcient solutions to these problems can save companies millions of dollars. We know of many examples in which optimization problems arise; for example: bus scheduling, telecommunication network design, travelling, and vehicle routing problems. Perhaps the most famous combinatorial optimization problem is the travelling salesman problem (TSP). The TSP can be simply thought of as the problem of figuring out a tour of cities a salesman

must travel (visiting each city exactly once) so that the total distance travelled is minimized. With the evergrowing arise of combinatorial optimization problems and their intrinsic link to industrial problem, many researchers, mathematicians and computer scientists, have developed a plethora of algorithms to solve these problems. We now distinguish the two types of algorithms; complete and approximate. Complete algorithms are able to solve these problems in such a way that in the end, the optimal solution is given. However, since many of these problems are N P -hard the optimal solution may take a long time to obtain. This is the reason why there are approximate algorithms. Approximate algorithms are designed, as expected, to give approximately the optimal solution. The great thing about using an approximate algorithm is that they can obtain good results in a relatively short amount of time.

Ant colony optimization (ACO) algorithms are some of the most recent class of algorithms designed to approximate combinatorial optimization problems. The algorithm behaves similar to real ants and their biological abilities to ı nd the nearest food source and bring it back to their nest. The main source of communication between ants is the depositing of chemically produced pheromone onto their paths. It is with this key idea that Marco Dorigo and colleagues were inspired to create this new class of algorithms.

ACO is a class of distributed algorithms used for solving NP-hard combinatorial optimization problems. Its introduction can be found in (Dorigo *et al*, 1996, 1999), (Dorigo and Gambardella, 1997a, 1997b), and (Dorigo and Di Caro, 1999).

The first form of ACO, Ant System (AS), was introduced by Dorigo *et al* (1991) and is based on the foraging behaviour observed in a real ant colony. The cooperation of ants and how they efficiently find the shortest routes have been formulated into an algorithm used to solve combinatorial optimization problems.

The first improvement of the initial AS is called the ***elitist strategy*** for AS (EAS) (Dorigo *et al*, 1996), where only the best-so-far solution is used to update the pheromone trails. The idea is to enhance the promising search space. Another improvement is called the ***rank-based*** AS (ASrank), proposed by Bullnheimer *et al* (1999). The amount of pheromone that each ant deposits on the trails decreases

according to its rank. Meanwhile, the best-so-far ant still deposits pheromone at each iteration. The results of an experimental evaluation suggest that ASrank performs slightly better than EAS and significantly better than AS.

A ***MAX-MIN Ant System*** is another improvement proposed by Stützle and Hoos (1997, 2000). It limits the possible range of pheromone trail values to an interval $[\tau_{min}, \tau_{max}]$ in order to avoid

stagnation by exploring solutions; all trails are initiated with the upper pheromone value and the pheromone evaporation rate is small; finally, pheromone trails are reinitiated whenever stagnation is met or a solution has not been improved for a certain number of consecutive iterations.

There are also a few extensions of AS, for example, the Ant Colony System (ACS) by Dorigo and Gambardella (1997a, b), Approximate Non-deterministic Tree Search (ANTS) by Maniezzo (1999) and population-based ACO (P-ACO) by Guntsch and Middendorf (2002a). Some local search methods can also be combined with ACO to improve the solutions.

ACO has been used to solve the traveling salesman problem, the quadratic assignment problem, data network routing problem (Schoonderwoerd *et al*, 1996), and scheduling problem (flow shop or job shop). It has been successful in finding near-optimal solutions comparable to those found using the state-of-the-art approaches in most of those problems except JSSP (Dorigo and Stüzle, 2004, pp.168). The following review presents results obtained from previous work of ACO related to scheduling problems and dynamic problems which may give insights for reactive scheduling in a dynamic job shop.

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 . Every solution of the optimization problem may be expressed in terms of feasible paths across the graph.

The generalized scheme of the algorithm of production schedule construction using the ant colony optimization method is narrowed down to the construction of an assembly process model as a directed graph, the ant colony formation and solution search considering the limits specified.

2. JOB SCHEDULING PROBLEMS

2.1 MANUFACTURING ENVIRONMENT

Manufacturing environments can be classified into five types: job shop, project shop, cellular system, flow line and continuous systems . In a *job shop* (Fig.), machines with the same or similar material processing capabilities are grouped together in workcenters. A part moves through the system by visiting the different workcenters according to the part's process plan. In a *project shop* (Fig.), a product's position remains fixed during manufacturing because of its size and/or weight and materials are brought to the product as needed

One lot of jobs refers to a batch of jobs which are simultaneously released to a manufacturing shop floor and the lot size directly affects inventory and scheduling. Generally, the lot sizes that can be processed by a discrete manufacturing system, which works on discrete pieces of products like metal parts, are related to the types of manufacturing systems. Normally, job shops and project shops are most suitable for small lot size production, flow lines are most suitable for large lot size

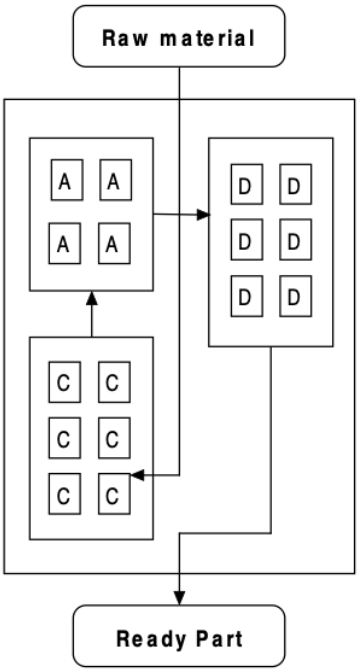
production, and cellular systems are most suitable for production of lots of intermediate size. It can be seen from Fig. 1.2 that lot sizes in job shops range from 1 to 100 jobs

The production management and control activities in a manufacturing system can be classified as strategic, tactical and operational activities, depending on the long, medium or short term nature of their tasks (Hopp and Spearman, 2000; Chryssolouris,
2006).

The strategic production management decides issues related to the determination of products according to the market demands or forecasts, the design of the manufacturing systems to produce those products, the generation of master schedule to meet the capacity requirement, etc. The tactical production management decides issues relating to the generation of detailed plans according to the master schedule. The results of this stage, such as shop orders with release and due dates are passed to the lower control level, i.e., the operational production management, which decides the processing of those orders on the shop floor in order to fulfill the order requirements, and at the same time, optimizes the performance of the manufacturing system. It needs proper scheduling strategies to meet those requirements. After scheduling, the schedule is transferred to the shop floor and the implementation of a schedule is often referred to as dispatching (Vollmann et al, 1992).

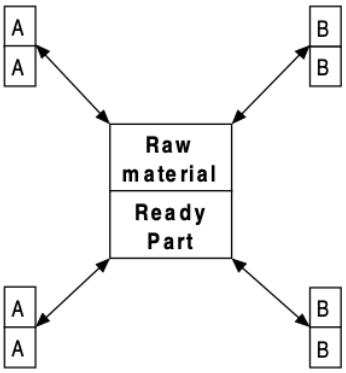
Scheduling deals with the allocation of scarce resources to tasks over time. It is a decision-making process with the goal of optimizing one or more objectives (Pinedo, 2002). The result of a scheduling procedure generates one or several schedules, which are defined as plans with reference to the sequence of and time allocated for each item or operation necessary to complete the item (Vollmann et al, 1992). A schedule can be represented as a Gantt Chart, which is a two-dimensional chart showing time along the horizontal axis and the resources along the vertical axis.

The main goal of manufacturing production management is to meet demands in a timely and cost-effective manner. In most manufacturing environments, especially in those with a wide variety of products, processes, and production levels, the construction of advance schedules is recognized as central to achieving this goal. Scheduling in manufacturing systems is very important for its roles in maximizing throughput and resource utilization, meeting due dates of orders, reducing inventory levels and cycle time, etc. Even small improvements in those measures can lead to considerable profit and thus increase the competitiveness of a factory.



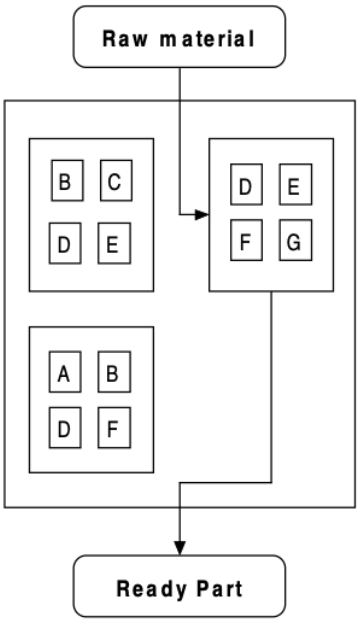
Machines/Resources are grouped according to the process they perform

(a) A job shop



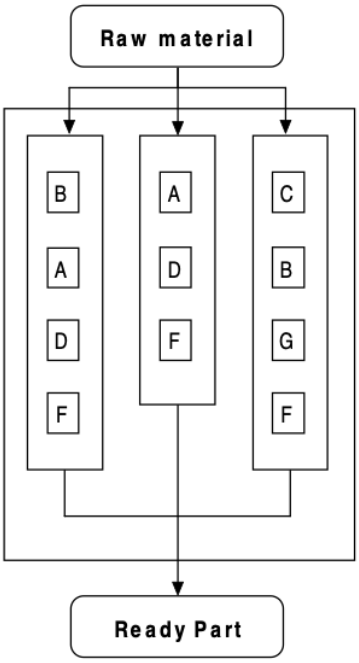
Machines/Resources are brought to and removed from stationary part as required

(b) A project shop

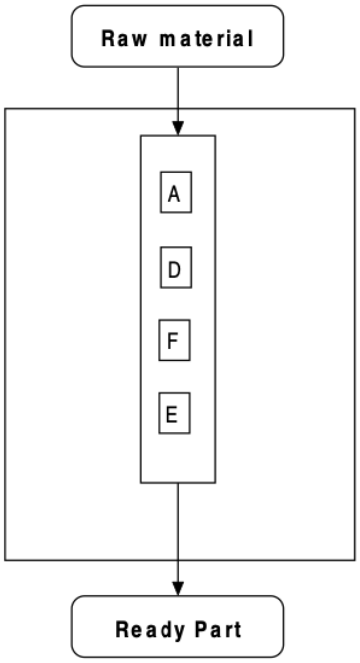


Machines/Resources are grouped according to the processes required for part families

(c) A cellular system



Machines/Resources are grouped in lines according to the operation sequence of one or more part types



Processes are grouped in lines according to the process sequence of the products

2.2 CLASSICAL SCHEDULING PROBLEMS

IMPORTANT NOTIONS ADOPTED

An ***operation*** (***O_{ij}***) refers to the j^{TH} elementary task of job i to be performed on a machine.

A ***job*** (***J_i***) refers to the i^{TH} job which has a set of operations that are interrelated by precedence constraints derived from technological restrictions.

The ***processing time*** (***p_{ij}***) of an operation is the amount of time required to process operation $\mathbf{O_{ij}}$

The ***setup time*** refers to the time required by a machine to shift from the current status to the next one in order to process the next operation. In the current studies, setup times are independent of operation sequence and are included in the processing time.

A *machine* (*M*) is a piece of equipment, a device, or a facility capable of performing an operation.

The ***due date*** (***di***) of job i is the time by which the last operation of the job should be completed.

The ***completion time*** (***Ci***) of job i is the time at which processing of the last operation of the job is completed.

2.3 CLASSIFICATION OF SCHEDULING PROBLEMS

A scheduling problem can be described based on the *n/m/A/B* classification scheme of Graham *et al* (1979). n is the number of jobs; m is the number of machines; the A field describes the machine environment. The B field describes the objective to be optimized and usually contains a single entry.

MACHINE ENVIRONMENTS

In a ***job shop*** (***G***), there are m machines , M_1, ... M_m, which are different from each other, and a set of n jobs J_1, ... J_n, which are to be processed on those machines subject to the sequence constraints of their operations. Job J_i ($1 \le i \le n$) consists of m_i operations O_{i1} , ... O_{im} ($0 \le m_i \le m$) and their respective number of machines can be given in a vector v_i , where $v_i(k)$ is the number of the machine that processes operation O_{ik} . The processing times of those operations are p_{i1} , ... p_{im} . A schedule has to be found so that all jobs are routed in the shop floor in a manner that the performance measures of the system can be optimized. The schedule decides the starting time t_{ik} for each operation O_{ik} of job J_i and the following formula holds:

$$t_{ik} = \max(t_{i,k-1} + p_{i,k-1}, t_{hl} + p_{hl})$$

t_{hl} is the starting time of job J_h , which is the job processed on the same machine immediately before job J_i . t_{ik} is decided by either the completion time of its direct preceding operation or the earliest available time of its machine.

OBJECTIVES

The objectives to be optimized are always a function of the completion times of the jobs. The objective criteria considered in this study include ***makespan***, mean flowtime, and mean tardiness, which are most commonly used in the literature of job shop scheduling. Performance measures related to inventory status like throughput, work-in-process and the size of jobs in a queue are also considered.

Makespan (C_{max}) is the "length" of the schedule, or an interval between the time at which the schedule begins and the time at which the schedule ends. Thus, the makespan of a schedule equals to max[C_i] , where i = 1, ..., m.

Given a measure of performance Z , which is defined as a function of the set of job completion times, and

Z=f(C1,C2 ,...Cn0) Z I**S** ***r***egular .

1. scheduling is to minimize Z , and 2) Z can increase only if at least one of the completion times in the schedule increases (Baker, 1974). Makespan is a regular performance measure while mean tardiness-related objectives are ***non-regular***.

A scheduling problem given as n/m/G/T refers to a job shop scheduling problem(JSSP) with n jobs and m machines

RESEARCH GOALS AND METHODOLOGIES

1. To analyze a dynamic JSSP, identify the systematic manners of research in this
   field, and define the domains of the dynamic JSSP.
2. To present the effectiveness of ACO in solving dynamic JSSPs, and demonstrate
   the effectiveness of its adaptation mechanism.

Literature Review

Scheduling as a research discipline dated back to early 1900s but serious analysis of scheduling problems did not begin until the advent of computer age in the 1950s and 1960s. Since then, a great amount of theoretical work has been reported. A good historical overview of the different approaches was given by Froeschl (1993) and an early introductory work on scheduling was reported by Baker (1974), French (1982),

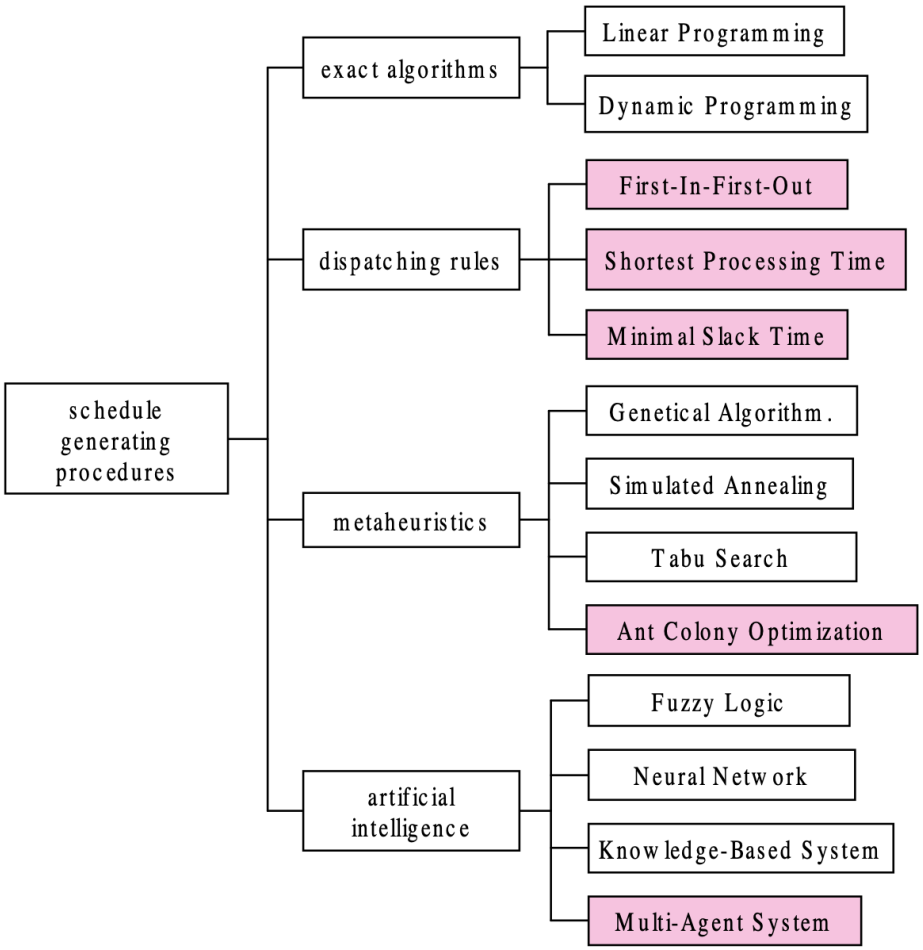
Buxey (1989), and Sule (1997). Literature reviews on static deterministic scheduling can be found in (Graves, 1981, Jain and Meeran, 1998, 1999, MacCarthy and Liu, 1993, Blazewicz et al, 1996, Sellers, 1996, Weirs, 1997, Jones and Rabelo, 1998, and Pinedo, 2002). Nowicki and Smutnicki (1995) provided an excellent review of minimum makespan job shop problems. Suresh and Chaudhari (1993) reviewed the dynamic scheduling literature.

This review starts with the approaches for static JSSPs; then the emphasis is put on the approaches for handling dynamic environments. Furthermore, the applications of ACO in the scheduling related fields are reviewed in detail to give a background of the current research.

3. Approaches for the classical job shop scheduling problems

3.1 An overview

The main approaches to solve the classical JSSPs include exact mathematical algorithms, dispatching rules, metaheuristics, and artificial intelligence methods.



3.1.1 Exact mathematical algorithms

Balas (1965, 1967) developed modern integer programming, which allows rather realistic JSSPs to be formulated in a manner that would theoretically permit them to be solved exactly. Two popular solution techniques for integer-programming problems are branch-and-bound and Lagrangian relaxation. Branch-and-bound is an enumerative technique, which systematically curtails undesired solutions by dynamically setting lower bounds through modeling the JSSP as a decision tree. Lagrangian relaxation solves integer-programming problems by omitting specific integer-valued constraints and adding the corresponding costs to the objective function.

Another exact mathematical algorithm reported is dynamic programming, which enumerates in an intelligent manner all the possible solutions. During the enumeration process, schedules which are not optimal are eliminated.
However, both integer programming and dynamic programming are computationally intensive. Thus large problems remain intractable although very small problems can be solved with optimal solutions. Subsequently, the majority of scheduling problems has to be solved using heuristics, which are techniques seeking good solutions instead of the optimal ones at a reasonable computational cost (Voß, 2001). Main heuristic approaches include dispatching rules, metaheuristics, and artificial intelligence.

3.1.2 DISPATCHING RULES

The simplest heuristic to find a solution is using dispatching rules, where a schedule is constructed in one iteration with generally a very light computational effort even for a large problem. A dispatching rule is used to prioritize jobs waiting for processing at the time that the waited machine/resource becomes available. The job with the highest priority is selected to be processed on the machine. An early survey can be found in Panwalkar and Iskander (1977).
Common dispatching rules employ processing times and due dates as deciding factors in simple rules or their complex combinations. Some dispatching rules are extensions of policies that work well on simple machine scheduling problems, for example, First-In-First-Out (FIFO), Shortest Processing Time (SPT), Minimal Slack Time (MST), and Earliest Due Date (EDD). In FIFO, the first operation coming into a workcenter has the highest priority; in SPT, the operation with the shortest processing time has the highest priority; in MST, the operation with the shortest slack time has the highest priority. The slack time indicates the temporal difference between the due time, the current time and the remaining computation time. In EDD, the operation with the earliest due date has the highest priority.

Other dispatching rules can be found in Panwalkar and Iskander (1977), which provides an extensive list of dispatching rules and their classification includes five categories: simple dispatching rules, combinations of simple rules, weighted priority indices, heuristic scheduling rules, and similar findings by others, such as Blackstone et al (1982), Ramasesh (1990), and Morton and Pentico (1993).
Owing to their inexpensive computational effort and robustness, dispatching rules are widely adopted, especially, in dynamic environments (Li et al, 1993). However, they

do not guarantee the realization of the full potential of a shop floor as they do not aim at optimization. Scheduling systems using algorithms, especially metaheuristic algorithms, have continuously been studied to provide optimized solutions.

3.1.3 METAHEURISTIC

A Metaheuristic is a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of problems (Voß et al, 1999). It refers to an iterative master process that guides and modifies subordinate heuristics in order to efficiently produce high-quality solutions. There may be a complete (or incomplete) single solution or a collection of solutions per iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method. A local search algorithm is a metaheuristic iteratively moving from solution to solution in the space of candidate solutions (the search space) until a solution deemed optimal is found or a time bound has elapsed. A construction method generates a schedule by adding in an operation one at a time until all operations are considered.

Examples of metaheuristics include genetic algorithms (Goldberg, 1989), simulated annealing (Kirkpatrick, et al, 1983), Tabu search (Glover, 1989, 1990; Glover and Laguna, 1997), ACO (Dorigo and Di Caro, 1999), and their hybrids. Each has its own perturbation methods, stopping rules, and methods for avoiding local optimum. The use of metaheuristics has significantly increased the ability of finding very high-quality solutions to hard, practically relevant combinatorial optimization problems in a reasonable time (Dorigo and Stützle, 2004).

3.1.4 ARTIFICIAL INTELLIGENCE

The approaches to solve scheduling problems in the artificial intelligence field are based on the inspirations from either human society or natural phenomena (Weiss, 1999). Many sophisticated procedures have been proposed including fuzzy logic, neural network, knowledge-based systems and MAS (Kusiak, 2000)

4. ANT COLONY OPTIMIZATION

1.BEHAVIOUR OF REAL ANT

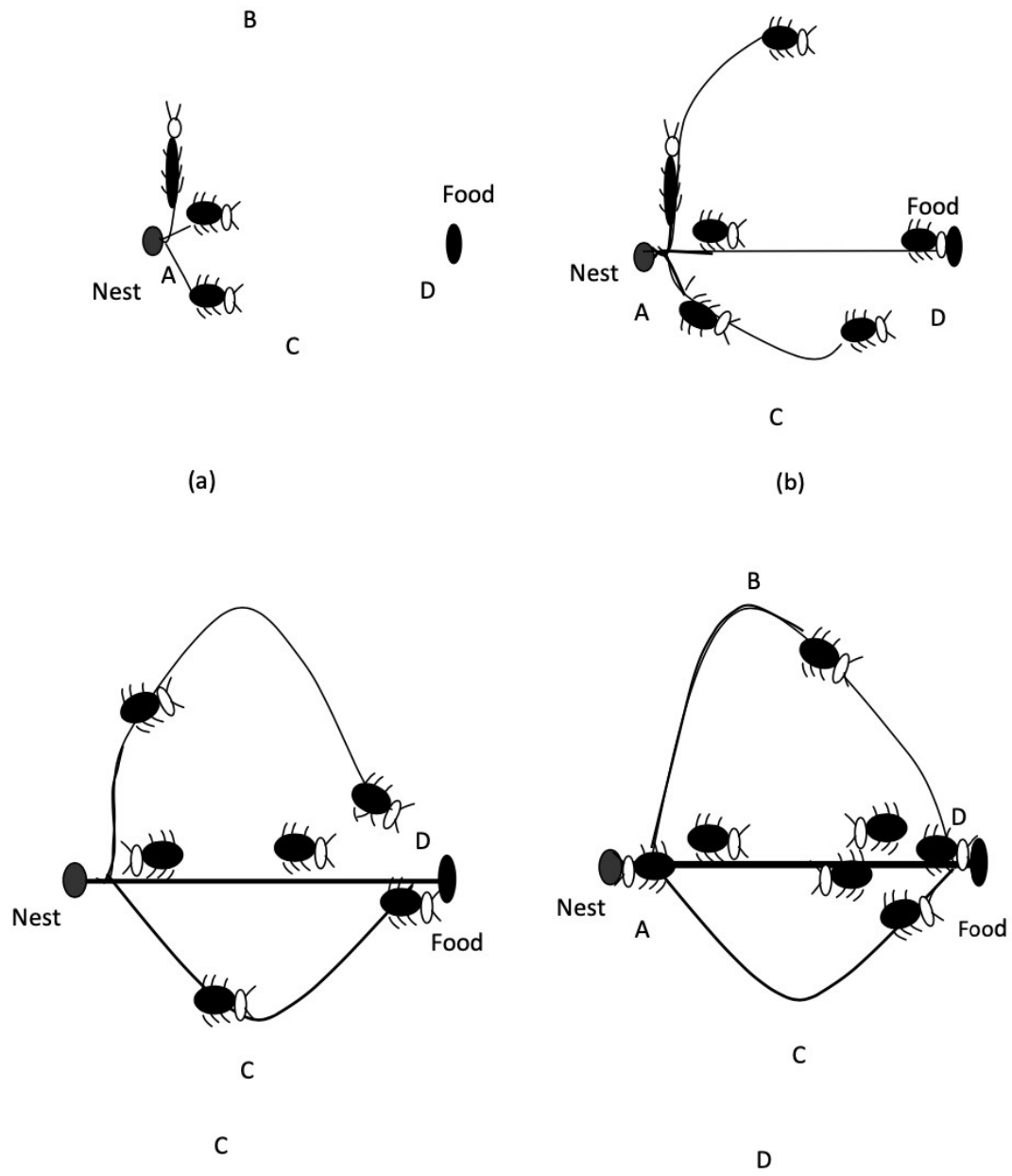
Ants have a limited awareness of their surroundings. They are not able to see a food source that is available in a long way away and also are not having the knowledge to plan a correct route to reach there and come back. The ants work collectively to search the area around their nest. They need to communicate in some way so as to work collectively and this communication is carried out by marking the ground with pheromone. The nature of the ants varies considerably. Some types of ants are capable of finding the best route between a food source and the nest (Li and Gong 2003). A simplified theory of how ants find

the best path to a food source is given in this thesis and key concepts relevant to ant algorithms are highlighted. All ants wander back and forth between the food source and the nest after finding the food source. Initially, they take a route by a random wander between the two sites. They also deposit pheromone on the ground during the wandering. This pheromone will affect paths of future ants, because the ants have a tendency to follow paths marked with pheromone. If the pheromone level on a path is more, the more likely, an ant is to wander in that path. When ants first leave the nest, they choose paths randomly.

Ants that have returned from the nest using shorter routes will arrive more quickly to the nest and will deposit pheromone sooner on that path. That is, the shorter routes will be more strongly marked with pheromone and ants in future will be more likely to use those routes. After some time, the shorter routes become very strongly marked with pheromone and thus, all ants will be following the shorter routes.

Ants are capable of finding the shortest path from a food source to the nest (Dorigo and Gambardella 1997a, Colorni et al 1993) without using visual cues. Hence, the inspiring source of ACO is the “pheromone trail laying and following” behavior of real ants, which use pheromone (Holldobler and Wilson 1990, Beckers et al 1992) as a communication medium. The probability for choosing the next path by an ant will be directly proportional to the amount of pheromone on that path. Referring to Figure 4.1(a), three ants start from a point A, where nest is available, travel randomly to reach food at a point D and deposit some amount of pheromone during their traveling. First ant selects a path through point B. Second ant travels directly to D and third ant travels through C. In Figure 4.1(b), second ant reaches the food first, since it travels through the shortest path and other two ants are still traveling towards the food in their paths. At the same time, three more ants start to travel from the nest. Since the pheromone level for all paths traveled by ants are same, new ants follow the paths already traveled by the previous ants. Since the ants travel at constant speed, more number of ants travel through path A-D than number of ants through paths A-B-D and A-C-D as denoted in Figures 4.1(c) and 4.1(d). In turn, pheromone is deposited on the paths according to the number of ants traveling in the respective paths.

Figure 4.1(e) shows thickness of the line indicating pheromone level of three paths after some time. Eventually, all ants choose the path A-D that is having the highest pheromone level.



(a) Ants Select Paths Randomly,
(b) Ant Selecting the Shortest Path Reaches
(c) and (d) food first,change of pheromone level while travelling

Pheromone is deposited and evaporated in paths of ants during the traveling of ants. Since the paths A-B-D and

A-C-D are not used by ants, the pheromone is evaporated on these paths.

After some time the pheromone is completely evaporated in longer paths, which are not available long time and all ants use
only one path having the shortest distance.

Ants are also capable of adapting to changes in the environment. For example, the shortest path found so far is no longer feasible due to a new obstacle on the way. Figure 4.3 shows the behavior of real ants. Ants are moving on a straight line, which connects a food source to the nest. As stated, each
ant probabilistically prefers to follow a direction rich in pheromone rather than a poorer one. This elementary behavior of real ants can be used to explain how they can find the shortest path, which reconnects a broken line after the sudden appearance of an unexpected obstacle interrupting the initial path .
Once the obstacle has appeared, those ants, which are just in front of the obstacle, cannot continue to follow the pheromone trail and therefore they have to choose between turning right or left . In this
situation, half number of ants may be expected to turn right and the other half to turn left.

The same situation can be found on the other side of the obstacle. It is interesting to note that those ants, which choose, by chance, shorter path around the obstacle will more rapidly reconstitute the interrupted.

pheromone trail compared to those, which choose longer path. Hence, shorter path will receive a higher amount of pheromone in the time unit and this will in turn cause a higher number of ants to choose shorter path. Due to this positive feedback,
very soon all ants will choose the shorter path. The most interesting aspect of this process is that finding the shortest path around the obstacle seems to be an emergent property of interaction between the obstacle shape and distributed behavior of ants. Although all ants move at approximately the same speed and deposit a pheromone trail at approximately the same rate, it is a fact that it takes longer to contour obstacles on their longer side than on their shorter side, which makes the

pheromone trail accumulate quicker on the shorter side. The ant's preference for higher pheromone trail levels makes this accumulation still quicker on the shorter path.

Key features

1. If a new shorter route than the current route is found, ants will continue to use the older and longer route for some time, because ants tend to follow the heavily marked trail that they had previously been using
2. Ants have difficulty with paths having many sharp corners. The ants reach these corners and take some time finding the way to continue. They may follow the trail back again, instead of the trail onwards, since both are equally well marked.
3. Objects marked with pheromone also affect the behavior of ants for some time after they are marked. However, the effect of trail diminishes, if the future ants will not deposit more pheromone.
4. If the pheromone level between the food source and the nest changes, the ants will try to find a new path using any elements of the old trails, which are also relevant at the current time. This often greatly reduces the time taken to find a new path.

- Ants co-operate by marking the search space they explore with pheromone.
- Ants wander at random, but are more likely to follow routes that have been marked with pheromone and the probability of following that route is greater, if the pheromone level is more.

2. ACO for static scheduling problems

The AS was first applied to JSSP by Colorni et al (1994). It was successful in finding solutions within 10% of the optima for both instances of 10x10 and 10x15 job shop scheduling problems (Dorigo et al, 1996). However, despite showing the viability of

the approach, the computational results were not competitive with state-of-the-art algorithms for classic JSSPs (Stützle and Dorigo, 1999).

EAS was applied to three benchmark JSSPs in 1999 (Zwaan and Marques, 1999). The results were within 8% and 26% of the best known optima for the 10/10/G/Cmax Muth-Thompson problem and the 20/10/G/Cmax Lawrence problem (OR-Library), respectively. The authors considered the results promising since the tests were only partially executed with an iteration number of 2000. The study also presented the importance of parameter settings.

There are also reports of other forms of JSSPs. Blum (2002) applied MMAS to solve the Group Shop Scheduling Problem (GSSP), which is a general Shop Scheduling problem covering JSSP and Open Shop Scheduling (OSSP). Several versions of MMAS were compared and the proposed algorithm could find optima for the tested benchmark JSSP (15x15) and OSSP. Stützle (1998) applied MMAS integrating a local search for a series of benchmark flow shop problems (FSP). The results were compared with several other heuristics and showed that the MMAS gave high quality solutions to FSP in a shorter time, performing better or at least comparable to other state-of-the-art algorithms.

3.1 ANT SYSTEM FOR THE TSP: THE FIRST ACO ALGORITHM (1991)

The transition probability from city i to city j for the k-th ant is defined as

3.1.1 **TRANSITION PROBABILITY**

$$p_{ij}(t) = \begin{cases} [\tau_{i,j}(t)]^{\alpha} [\eta_{i,j}(t)]^{\beta} / \sum_{i \varepsilon N_i^k} [\tau_{i,j}(t)]^{\alpha} [\eta_{i,j}(t)]^{\beta} & \text{if } i \varepsilon N_i^k \\ 0 & \text{otherwise} \end{cases}$$

and we note that the fraction ensures that

Variables in the formula are de ned as follows:

- $\tau_{i,j}(t)$ = amount of pheromone trail on edge (i, j) at time t.
- $\eta_{i,j}$ = the visibility; expressed as vertices.

$1/d_{i,j}$

where $d_{i,j}$ is the euclidean distance between two

- α = parameter controlling the relative weight of phermone trail
- β = parameter controlling the weight of visibility

Data-strutures that are used are:

- t abu $_k$ – dynamically growing vector containing the tabu list of the k-th ant (list of

vertices visited by the k-th ant)

• N_i^k = V \ t abu $_k$ – list of vertices not visited by the k-th ant

3.1.2 PHEROMONE UPDATE

After each ant has completed their tour, pheromone evaporation on all arcs is triggered, andthen each ant k deposits a quantity of pheromone $\varphi\tau^k_{i,j}$ (t) on each arc that it has used.

$$\Delta\tau_{xy}^k = \begin{cases} Q/L_k & \text{if ant } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

Where Q is a constant[2] and L^k (t) is the length of the tour done by ant k at iteration t .

One can implement the addition of new pheromone and pheromone evaporation to all arcs. I,j = x,y

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_k^m \Delta\tau_{xy}^k$$

where is the amount of pheromone deposited for a state transition , is the *pheromone evaporation coefficient,* is the number of ants and is the amount of pheromone deposited by th ant, typically given for a TSP problem (with moves corresponding to arcs of the graph)

ANT COLONY SYSTEM

Improved from the Ant System in 1996 [Dorigo, 1999]

argmax $_{i \varepsilon Ni^k}$ $\{\tau_{i,l}[\eta]^\beta\}$ if $q \leq q_0$ (exploitation)

j=

J otherwise (exploration)

where q is a random variable uniformly distributed in [0, 1], q o ($0 \leq q_o \leq 1$) is a parameter,

and J is a random variable selected according to the probability distribution

$p_{ij}(t) = [\tau_{i,j}(t)]^\alpha [\eta_{i,j}(t)]^\beta / \xi_{i \varepsilon Ni^k} [\tau_{i,j}(t)]^\alpha [\eta_{i,j}(t)]^\beta$ if $i \varepsilon N_i^k$

0 otherwise

LOCAL TRAIL PHEROMONE UPDATE

Similar to evaporation we update pheromone on the trail by

$$\tau_{i,j}(t) \leftarrow (1-\rho)\tau_{i,j}(t) + \varphi\tau_{i,j}(t)$$

but we let $4\tau_{i,j} = (nL_{nn})^{-1}$; where L nn is the length of a nearest-neighbor tour.

[By doing the local pheromone update (results were improved and made great use of positive-feedback. –improved results, etc.)

GLOBAL PHEROMONE UPDATE

$$\tau_{i,j}(t) \leftarrow (1-\rho)\tau_{i,j}(t) + \rho\, 1/L^{ba} \text{ for } (i,j) \xi T^{ba}$$

ANT-Q(1995)

Before ACS, there was Ant-Q. Essentially Ant-Q was an algorithm that attempted to merge main properties of both AS and Q-learning [Dorigo, 1999].

4. Ant Colony System

AS was initially applied to solve traveling salesman problem. However, it was not able to compete against the state-of-the art algorithms in the field. Authors of AS, on the other hand, have the merit to introduce ACO algorithms and show the potentiality of using artificial pheromone and artificial ants to search of better solutions for complex optimization problems. Then research on AS was carried out in order to achieve the following goals:

• To improve the performance of the algorithm.
• To investigate and better explain the behavior of the algorithm.

Gambardella and Dorigo (1995) proposed an extension of AS called Ant-Q algorithm, which integrates several ideas from Q-learning (Watkins and Dayan 1992). They also proposed Ant Colony System (ACS)

(Gambardella and Dorigo 1996, Dorigo and Gambardella 1997b), which is a simplified version of Ant-Q. ACS maintains the same level of performance as Ant-Q algorithm in terms of the complexity and the computational results. The following three aspects make ACS to differ from AS: (i) State transition, (ii) Pheromone updating and (iii) Hybridization and performance improvement.

JSSP AND ACO

The new agent, scheduler, can communicate with the job, shop floor and workcenter agents. A job agent contacts both the shop floor and the scheduler right after it has been generated by the job releaser agent. The scheduler agent then prepares to reschedule to include this new incoming job according to its states. A shop floor agent proactively requests the scheduler to update the schedule when necessary and suspends its actions. The scheduler then updates all the workcenters with new schedules. All workcenters confirm to the scheduler regarding to the reception of schedules; then the scheduler replies to the shop floor agent that its request has been fulfilled. At this time, the job shop resumes its work.

In ACO algorithms a finite size colony of artificial ants with the above described characteristics collectively searches for good quality solutions to the optimization problem under consideration. Each ant builds a solution, or a component of it, starting from an initial state selected according to some problem dependent criteria. While building its own solution, each ant collects information on the problem characteristics and on its own performance, and uses this information to modify the representation of the problem, as seen by the other ants. Ants can act concurrently and independently, showing a cooperative behavior. They do not use direct communication: it is the stigmergy paradigm that governs the information exchange among the ants.

Structure of basic ant colony optimization algorithm

Initialization

Initialize parameters like α, β, ρ, q_0, and Q.

Store a maximum value to the solution.

Calculate initial pheromone value

Construction and Improvement

While termination condition not satisfied do

Construction

For each ant k do

Choose a state i with a probability and
add i to partial solution

Update pheromone trial for the current move

End For

Update pheromone trail for the best ant’s path

Improve the solution

End While

Output

Print the best solution found.

-> the algorithm initializes various parameters and assigns a maximum value for the current solution. It also calculates initial pheromone value during the initialization phase. In the construction phase, the algorithm finds a solution and updates the pheromone

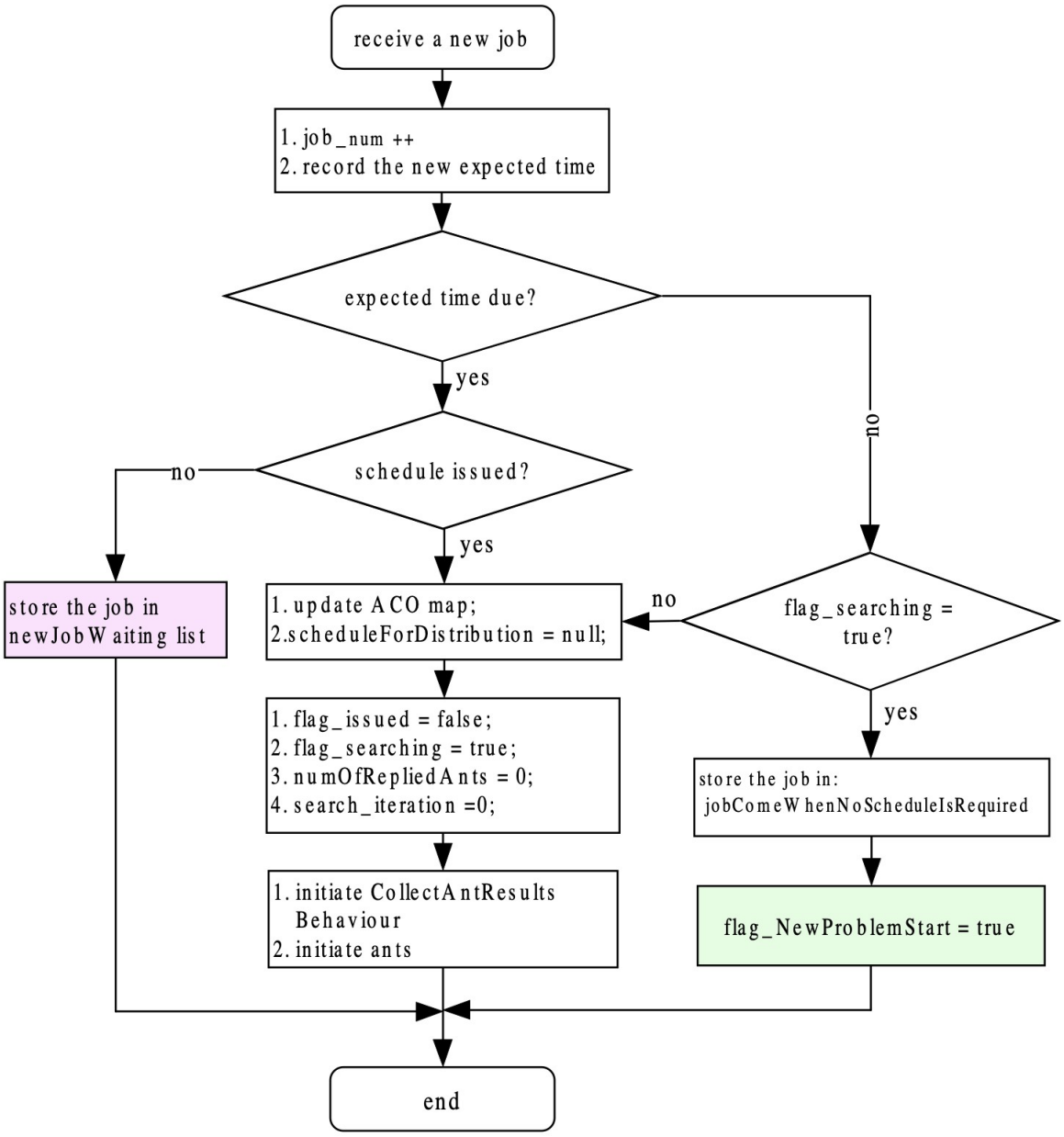
trial, which is used to improve the solution. The algorithm repeats the construction phase until termination condition is met and finally prints the best solution found.

Coordination among behaviours in the scheduler agent

The scheduler agent can be in either one of two states: idle or searching. It is idle when all jobs are scheduled and the schedule is issued; otherwise, it is in a searching state. It should be able to receive new jobs and react to schedule requests anytime. These two abilities are supported by the two independent and concurrent behaviours: receive a new job and receive schedule requests. The former behaviour is initiated by the arrival of a new job agent and the latter is initiated by the job shop agent. Meanwhile, solutions from the ant agents are collected through "collect ant results" behaviour. The following sections present the flowcharts of those behaviours and the coordination among them.

Behaviour of receiving a new job

flowchart for the behaviour of receiving a new job, which triggers the rescheduling procedure of the scheduler when it comes to the shop floor at the reception/shipping section. The schedule should have been updated by the time a new job arrives at its first workcenter. This point of time is called expected due time of rescheduling and the operations scheduled before this moment by the previous schedule should not be considered in the new scheduling problem.



Upon receiving a new job, the scheduler agent increases the number of its jobs by one, records the new expected due time and checks whether a previous schedule request, if any, is due. The new job should be stored temporarily in the list called jobComeWhenNoScheduleIsRequired if it has not reached the expected time for releasing a schedule and the scheduler is seeking a schedule. A flag named flag_NewProblemStart is then raised and marked in green color in Fig. 5.1. It will be handled in Fig. 5.3 in the location with the same color. This mechanism is to synchronize the actions between two concurrent behaviours: receiving a new job and collecting ant results. However, the scheduler starts seeking schedules if it is idle at the time a new job arrives.

A new job may come at the same time that a schedule is due to be issued. The scheduler should guarantee that the due schedule is issued before the new job is considered for rescheduling. If the schedule is not issued, the new job has to be stored temporarily in the list called newJobWaiting and the procedure is colored pink in Fig. 5.1. It will be included to generate a new schedule right after the due schedule is issued indicated in Fig. 5.3 in the procedure highlighted with the same color. This

mechanism is to synchronize the two concurrent behaviours: receiving a new job and receiving a schedule request. Otherwise, the rescheduling procedure is executed immediately if the previous schedule is issued.

Behaviour of receiving a schedule request

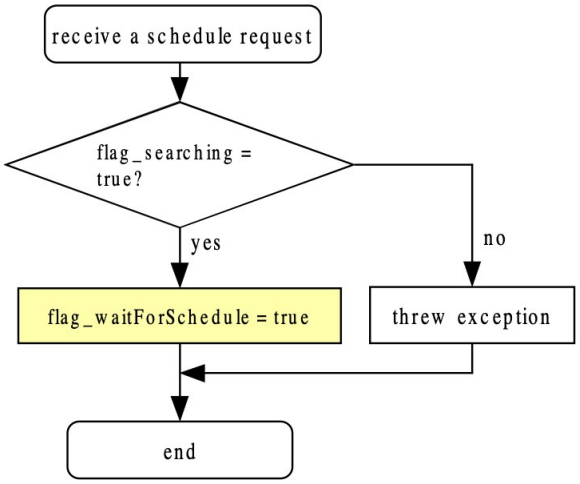


Fig. 5.2. The behaviour of the scheduler agent receiving a schedule request

Fig. 5.2 presents the flowchart of the behaviour of receiving a schedule request. The scheduler basically checks whether it is in the correct state of searching a schedule and raises a flag called flag_waitForSchedule, which is marked in yellow color, to wait for a schedule. Then the behaviour of collecting ant results can immediately dispatch a new schedule once it is ready. The procedure is indicated in the procedure marked in the same color in Fig. 5.3. This flag synchronizes the behaviours of requesting schedule and collecting ant results

The main goal of this behaviour is to collect all ant results, update the best solution and the pheromone matrix, and initiate ants to search schedule for the next round of searching (Fig. 5.3). The behaviour checks the flag of new job coming (flag_NewProblemStart) when all the ant results have been collected. If it is raised, the record of the best solution is removed and the pheromone matrix/ACO map is updated. A new problem is then formed and rescheduling starts.
However, searching continues if the problem is not changed until the minimum number of iterations is met. At that time, the schedule agent checks whether a schedule request (flag_waitForSchedule) is waiting. It should dispatch schedules to all the workcenters if a request is made, otherwise, it will continue to search to find better solutions until a maximal number of iterations is reached. The list containing the waiting jobs (newJobWaiting list) is checked after the schedules are dispatched in order to synchronize the concurrency between a new job event and a schedule request.

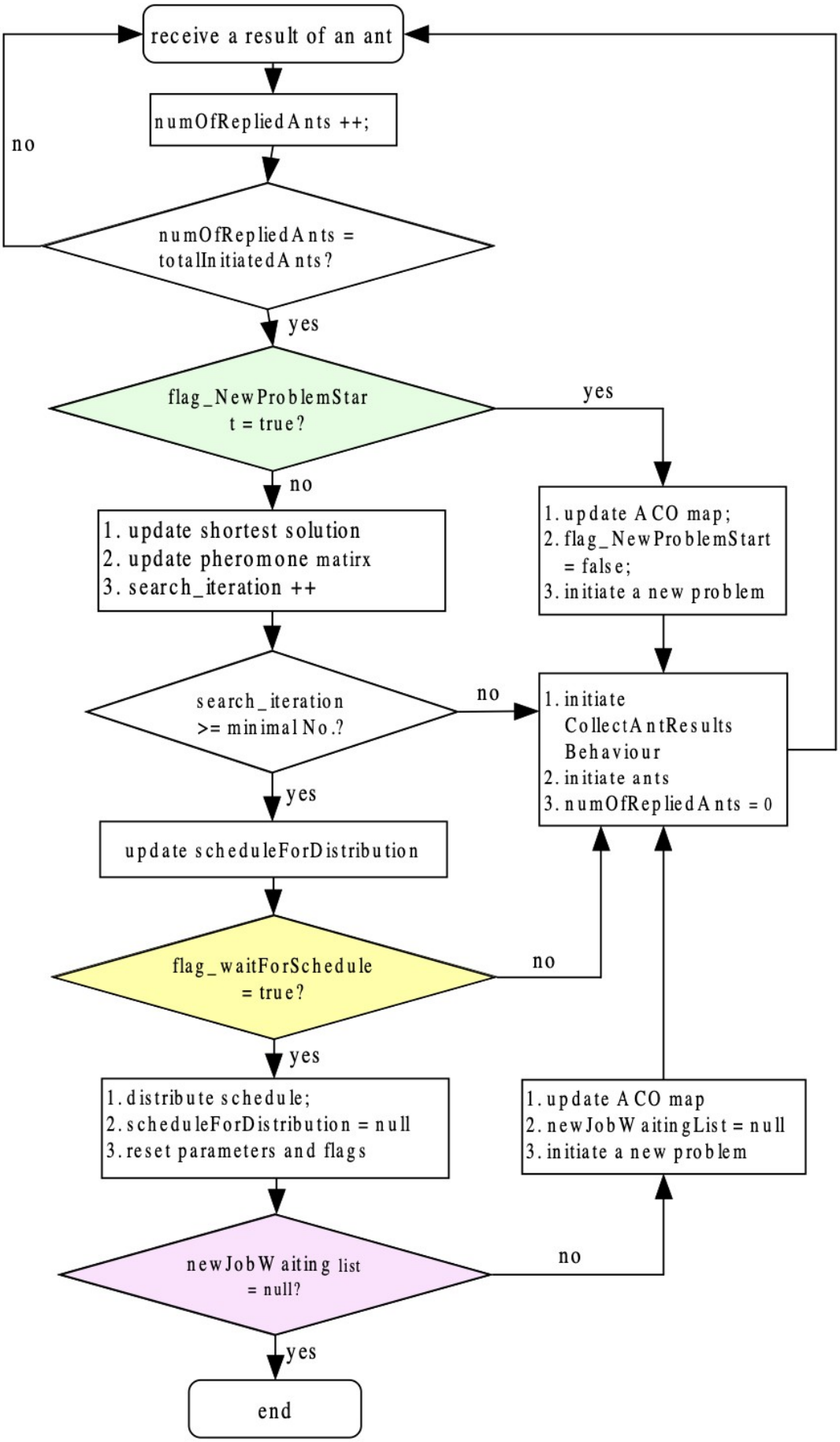


Fig. 5.3. The behaviour of collecting ant results in the scheduler agent

In this section, the flowchart of the ACO algorithm is first illustrated; the representation of the JSSP as well as the application of ACO for dynamic JSSPs is then described.

The notations used in the ACO algorithm are listed as follows
h is the index of iteration number
p_{ij} is the probability for an ant to travel from node i to node j at h_{th} iteration
τ_{ij} is the quantity of pheromone on the edge connecting nodes i and j at h_{th} iteration;
d_{ij} is the heuristic distance between nodes i and j;
ρ is the evaporation coefficient, which can be a real number between 0 and 1.0.
$\tau_{ij}(h)$ is the quantity of increased pheromone on the edge connecting nodes i and j at h_{th} iteration;
Q is a constant representing the total quality of pheromone on a route;
$f_{evaluation}(best\_so\_far)$ is the best value obtained so far optimizing the given objective.

ACO FLOWCHART

The flowchart of the ACO algorithm is given in Fig. 5.4. The basic idea is to repetitively initiate a set of ants, which walk in a common environment (problem graph) comprised of all the operations in a JSSP. The operations are modeled as nodes in a graph, which is described in detail . Each ant walks through all of those operations (nodes) one by one and thus forms a route, which can be interpreted as schedules and its length can represent the value of some performance measures like makespan, flowtime, or tardiness. The goal of each ant is to find a shortest route.

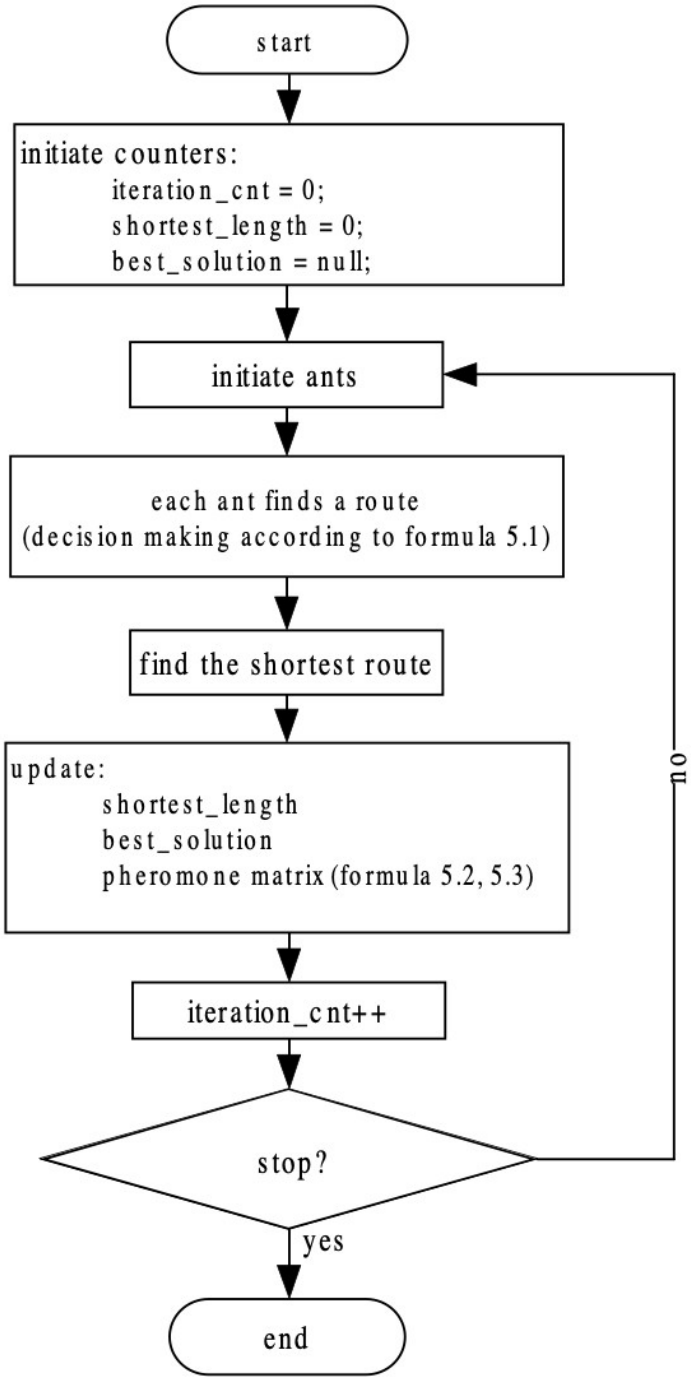


Fig. 5.4. The flow chart of the ACO algorithm

A walking ant leaves behind on its route some amount of pheromone, which changes the global environment. The probability for an ant to choose its next node is directed by both the amount of pheromone on the route and the distance from its current location to the targeted one. Ant i chooses the next node according to the State Transition Rule in formula (Dorigo et al, 1996)

$$p_{ij}(t) = \begin{cases} [\tau_{i,j}(t)]^{\alpha} [\eta_{i,j}(t)]^{\beta} / \xi_{i \varepsilon N_i^k} [\tau_{i,j}(t)]^{\alpha} [\eta_{i,j}(t)]^{\beta} & \text{if } i \varepsilon N_i^k \\ 0 & \text{otherwise} \end{cases}$$

The heuristic distance d_{IJ} in this study is the sum of traveling time between the current workcenter to the target workcenter and the processing time of the operation in the target workcenter. The environment is represented by a pheromone matrix, which is updated by the best solution at each iteration. The updating can be described in formulae (5.2) and (5.3) (Dorigo et al, 1996).

Pheromones on all edges evaporate at the rate of $\cup$ so as to diversify the search procedure into larger solution spaces and jump out of local optima. The information

of the best solution can be used to intensify certain search areas by strengthening the pheromones on all the edges of the best route by an amount of φτij (t+1) through
formula (5.2).
The centralized actions include choosing and keeping the best solution, as well as deciding whether or not to continue solution seeking.

ACO for job shop scheduling problems

Each job in a classical JSSP is comprised of several operations to be processed on different machines. Generally, their technical orders and the processing times are represented in a technical matrix TM and a processing time matrix PM, respectively. Each row of TM indicates the order of machines that all the operations of one job will visit while each row of PM indicates the processing times that all those operations will take on their processing machines. Simple examples of these are given as

$$TM = \begin{bmatrix} M1 & M2 & M3 \\ M3 & M1 & M2 \end{bmatrix} \qquad PM = \begin{bmatrix} t(O_{11}) & t(O_{12}) & t(O_{13}) \\ t(O_{21}) & t(O_{22}) & t(O_{23}) \end{bmatrix}$$

Fig. 5.5. The technical matrix TM and the processing matrix PM for a 2 x 3 JSSP

Fig. 5.5 presents a technical matrix and a processing matrix of a JSSP with two jobs and three machines. The first job has three operations O_{11} , O_{12} , and O_{13}that will be processed on machines M1, M2 and M3, in that order, and its three operations need processing times of) t(O_{11}),t(O_{12}) and t(O_{13})respectively.

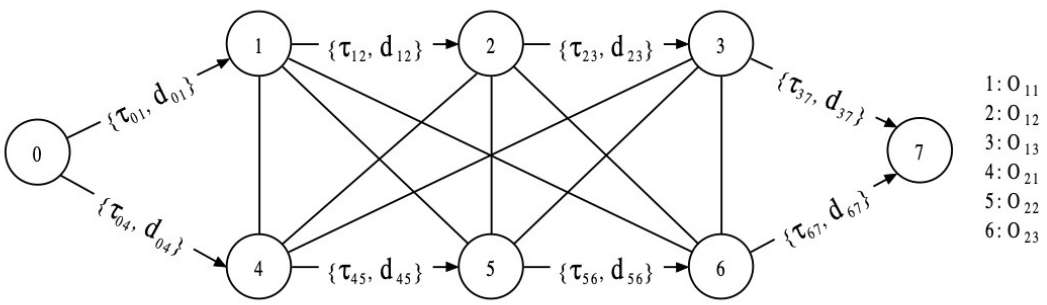


Fig. 5.6. The graph representing a 2 x 3 JSSP

The JSSP above can be represented as a graph (Fig. 5.6). Nodes 1 to 6 represent operations O_{11} ,O_{12} , to O_{13}, and O_{21} , O_{22} , to O_{23}. They are connected by horizontal directional edges indicating the precedence constraints given in matrix TM. The bi-directional edges indicate no ordering constraints among those operations. Dummy nodes 0 and 7 representing the source and the sink of the graph are the starting and the
ending points of routing. They are connected by directional edges to the first and the
last operations of all jobs, respectively.

Each edge is associated with a pair of values $\{\tau_{ij}, d_{ij}\}$ representing the amount of pheromone on it and the heuristic distance between the two nodes it connects. The value of d_{ij} can be easily looked up from matrix PM while the value for τ_{ij} should be found in the pheromone matrix, which is updated by the ants who found the best solutions . Pheromone matrix for the previous JSSP is shown in which records the pheromone values of all the edges connecting
every two nodes.

REPRESENTATION
Application of ACO for Dynamic Job Shop Scheduling Problems

ACO is applied to two dynamic job scheduling problems, which have
the same mean total workload but different dynamic levels and disturbance severity. Its performances on these two problems are statistically analyzed and the effects of adaptation mechanism are next studied. Furthermore, the effects of two important parameters in the ACO algorithm, namely the minimum number of iterations and the size of searching ants per iteration, which control the computational time and the solution quality of an intermediate scheduling problem, are also investigated. The results show that ACO can perform effectively in both cases; the adaptation mechanism can significantly improve the performance of ACO when disturbances are not severe; increasing the size of iterations and ants per iteration does not necessarily
improve the overall performance of ACO.

6.1 EXPERIMENTAL DESIGN

It is assumed that the reception of a new job will trigger a rescheduling procedure to find a full schedule with makespan as performance measure within the computational timeslot. The best-so-far schedule is then dispatched to be executed in all workcenters. The rescheduling procedure repeats until the preset stop criteria are met.

6.1.1 Experimental environments

- **PROBLEM CONFIGURATION**

The dynamic job shop studied is shown in Fig. with five workcenters and one receiving/shipping station. The numbers of machines in workcenters 1 to 5 are 4, 2, 5, 3, and 2, respectively. The machines in the same workcenter are assumed identical. The distances between all the workcenters are given in Table 4.1. Jobs are transported between workcenters by MHDs and the time spent on one trip is proportional to the distance between the two locations. All MHDs are assumed to be moving at a constant speed of 5 feet per second and they are assumed to be adequate.

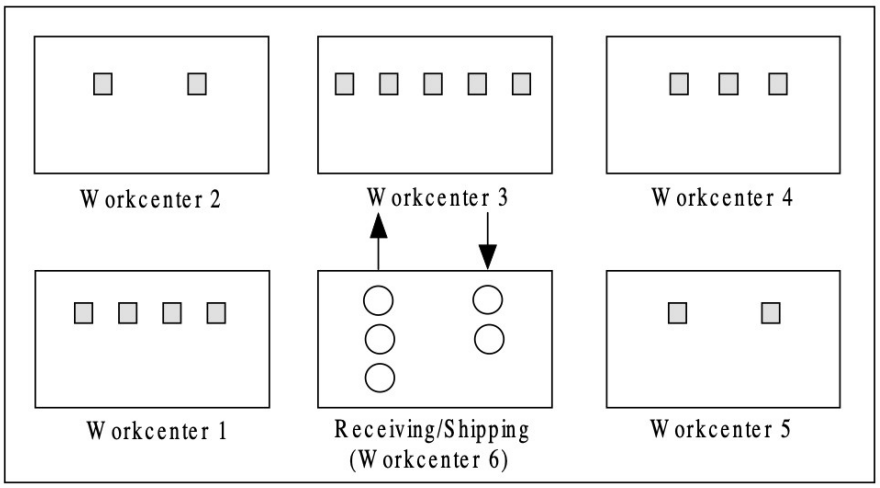


Table 4.1. Distances between workcenters (feet)

Workcenter	1	2	3	4	5	6
1	0	150	213	336	300	150
2	150	0	150	300	336	213
3	213	150	0	150	213	150
4	336	300	150	0	150	213
5	300	336	213	150	0	150
6	150	213	150	213	150	0

New jobs arrive at the receiving/shipping station (workcenter 6) and travel among workcenters according to their technical orders and finally leave the system from the receiving/shipping station. There are a total of 120!5 types of jobs and each type of job occurs with a probability of !51 and the total processing time for each job is 1 hour.

Table 4.2. Technical routes of jobs

Job type	Work stations in routing
1	3, 1, 2, 5
2	4, 1, 3
3	2, 5, 1, 4, 3

Table 4.3. Processing times of all operations

Job type	Mean service time for successive operations (hours)
1	0.25, 0.15, 0.10, 0.30
2	0.15, 0.20, 0.30
3	0.15, 0.10, 0.35, 0.20, 0.20

- **ACO PARAMETERS**

The parameters of the ACO algorithm are $\alpha = 10.0$, $\beta = 10.0$, $Q = 1.0$, AND $\tau_0 = 0.5$ tuned by Zwaan and Marques (1999) to solve several JSSP benchmarks.
They are adopted here as each intermediate JSSP is similar to those benchmarks.
It is assumed that the computation timeslot determined by S_{min} is within the time
constraint in realistic applications. The following are the default values: $s_{min} = 25$,
$s_{max} = 100$, and $u = 10$.

The performance objective of those intermediate JSSPs has to be decided in order to yield the best total throughput, overall mean flow time or overall mean tardiness. Three performance measures for those intermediate JSSPs are tested and they are the makespan, mean flowtime, and mean tardiness. Irrespective of the intermediate performance measure, evaluation values are recorded for all the three overall performance measures: total throughput, overall mean flowtime, and overall mean tardiness.

EXPERIMENTAL VARIABLES

Jobs arrive at the shop floor with inter-arrival times that are independent exponential random variables. The mean job inter-arrival time and the lot size are the two problem variables that decide the utilizations of workcenters. Two levels of job-arrival frequencies with the same mean size of total jobs are tested. In problem 1, jobs arrive one by one with the mean job inter-arrival time is nine jobs per hour. In problem 2, jobs are released in lots and arrive one lot per hour with nine jobs per lot. Jobs in one lot can be different types and will be processed job by job. In both problems, the type of a job is randomly decided so that each one of the 120 types has an equal chance to be chosen. Thus the mean total processing time demanded on each workcenter is the same.

The size of jobs in a lot determines the severity that an underlying scheduling problem is disturbed. For example, there are 16 unprocessed operations when a lot of new jobs are released to the shop floor. The size of operations for the new intermediate JSSP is 22 if there is only one job with 6 operations in the lot. The old operations take about 73% (16/22) of the total operations in the new problem. However, they take only 57% (16/28) of the total operations in the new problem if there is one additional job (also with 6 operations) in the lot. Obviously, the underlying problem is changed more severely by the larger lot with two jobs than the smaller one with one job.
The simulation for each problem runs five replications for 200 simulation hours (totally about 1800 jobs) and the warming-up time is 20 hours (about 180 jobs). Only steady-state performance is measured and the average values of five replications are listed for all the performance measures.

Computational results and analysis

Performance measures like the proportion of machine busy time, both the average and the maximum numbers of waiting jobs in queue are recorded by each workcenter agent while the average daily throughput, the average time in system, the average total time in queues, the maximum size of WIP are recorded by the shop floor agent. The maximum and the average sizes of operations in the scheduling procedure are recorded by the ACO scheduler agent.

Some general observations are as follows. Firstly, workcenters 2 and 5 are bottlenecks shown in all tables with utilizations of approximately 90%. Secondly, the machine utilization is inversely proportional to the number of the machines in its workcenter. The above two results are in accordance with the facts that the numbers of machines in both workcenters are the smallest with only 2 while having the same workload as other workcenters. Thirdly, the improvement in the average daily throughput and the machine utilization can reduce the average and maximum numbers of waiting jobs in a queue, the average time jobs spending in the system, the average total waiting times

jobs spending in queues, the maximal size of WIP, and the maximal/average size of operations of intermediate problems, which reflects the overall performance of ACO.

ACO performance analysis

The performances of ACO in two dynamic JSSPs are listed in tables 6.1 and 6.2. The 200 hourly throughputs of the five replications for both problems with the adaptation mechanism are plotted in figures 6.2 and 6.3 using moving average $Y_i(20)$ with a window of 20 (Law and Kelton, 2000) and a warming up period of l = 20 hours is obtained. Next, 90 percent confidence intervals for the steady-state mean daily throughputs of the two problems are constructed as [72.09,72.43] for Problem 1 and [73.19,74.75] for problem 2.

The effects of the ACO adaptation mechanism

The results indicate that the adaptation mechanism has greater effects in the situation where disturbances are not severe as in problem 1 and has little effect in the situation where disturbances are severe as in problem 2. The observation can be explained as follows. In problem 1, jobs arrive one by one and neighboring intermediate JSSPs are not severely different. A good solution can be found through the adaptation mechanism within a given computational timeslot. However, in problem 2, there would be not much difference between the pheromone matrices with and without the adaptation mechanism since the underlying problem can be dramatically changed by a large lot.

The effects of the number of minimal iterations

This seems to be against the initial expectation that increasing the number of minimal iterations can increase the optimality of an intermediate schedule and thus improve the overall performance of ACO.

This phenomenon could be explained as follows. The pheromone values of certain edges are increased too much as the result of increasing mins and the initial amount of pheromone on the new edges introduced by new jobs becomes trivial. Thus the pheromone matrix fails to properly represent a new scheduling problem and is called too rigid to find a new good solution. Thus, the scheduler can only produce a worse intermediate schedule each time, especially in a highly dynamic environment where the computational time is limited.

The effects of changing the number of ants per iteration

which show that the overall performance of ACO deteriorates as the size of ants per iteration increases. For example, with otherproblem parameters unchanged, the average daily throughput decreases from 72.258 (Table 6.3), 68.364 (Table 6.5), to 65.502 (Table 6.5) as the size of ants per iteration increases from 10, 20, to 40 in Problem 1 while the same performance measure decreases from 73.973 (Table 6.4), 72.978 (Table 6.6), to 71.502 (Table 6.6) in Problem 2.

The phenomenon can be explained as follows. A schedule with a small makespan is more likely found by more ants; subsequently, a greater pheromone value is added on the related edges. The optimality found in this schedule will be fully realized if the execution of the schedule is not disturbed by any dynamic/stochastic events. However, once the execution is disturbed, the schedule’s optimality will not be able to be fully realized. Furthermore, the amount of pheromone left on edges by the

Table 6.5. Increase the number of ants per iteration – Problem 1

Mean job inter-arrival time: 1/9 hour, 1 job/lot ACO (u = 20/40) 120 types of jobs (randomly)			Number of machines: 4, 2, 5, 3, 2 Simulation time: 200 hours Warming up time: 20 hours		
performance measure	**1**	**2**	**3**	**4**	**5**
Machine utilization (w orkcenter)	0.421/0.423	0.873/0.838	0.348/0.335	0.550/0.566	0.884/0.848
Average number in queue (w orkcenter)	4.666/5.209	34.264/32.949	3.112/2.937	12.153/14.593	34.237/32.734
Maximum number in queue (w orkcenter)	18.2(34)/ 22.2(34)	75.2(143)/ 70(138)	13.2(22)/ 11.8(21)	36.2(79)/ 41.2(83)	72.8(122)/ 68.6(116)
Average daily throughput (shop floor)	68.364/65.502				
Average time in system (shop floor)	9.999/10.232				
Average total time in queues (shop floor)	8.923/9.156				
Maximal size of WIP (shop floor)	189.8(389)/200.2(382)				
maximal size of ACO operations	626.6(1320)/674.2(1332)				
average size of ACO operations	275.4(529)/262.6(531)				

Table 6.6. Increase the number of ants per iteration – Problem 2

Mean job inter-arrival time: 1/1 hour, 9 jobs/lot ACO (u = 20/40) 120 types of jobs (randomly)			Number of machines: 4, 2, 5, 3, 2 Simulation time: 200 hours Warming up time: 20 hours		
performance measure	**1**	**2**	**3**	**4**	**5**
Machine utilization (w orkcenter)	0.434/0.453	0.916/0.903	0.319/0.355	0.614/0.605	0.929/0.914
Average number in queue (w orkcenter)	4.940/9.140	31.686/37.849	3.372/6.538	8.251/16.221	33.826/37.819
Maximum number in queue (w orkcenter)	19.8(32)/ 30.0(41)	73.4(84)/ 77.2(97)	17.6(23)/ 27.4(36)	25.2(36)/ 38.8(45)	72.4(87)/ 75.6(94)
Average daily throughput (shop floor)	72.978/71.502				
Average time in system (shop floor)	9.759/12.349				
Average total time in queues (shop floor)	8.683/11.273				
Maximal size of WIP (shop floor)	166.6(213)/177.6(282)				
maximal size of ACO operations	599.8(679)/718.2(963)				
average size of ACO operations	305.4(443)/358.6(507)				

The effects of pheromone adaptation – Problem 1

Mean job inter-arrival time: 1/9 hour, 1 job/lot ACO (with/without pheromone adaptation) (10 ants) 120 types of jobs (randomly)			Number of machines: 4, 2, 5, 3, 2 Simulation time: 200 hours Warming up time: 20 hours		
performance measure	**1**	**2**	**3**	**4**	**5**
Proportion machines busy (workcenter)	0.404/0.421	0.902/0.830	0.355/0.334	0.564/0.563	0.916/0.839
Average number in queue (workcenter)	0.703/5.764	5.558/36.115	0.404/3.316	1.297/14.793	5.838/35.726
Maximum number in queue (workcenter)	7.0(10)/ 20.4(33)	21.2(28)/ 104.4(140)	5.2(6)/ 14.0(24)	9.4(12)/ 41.4(70)	21.0(31)/ 77.6(123)
Average daily throughput (shop floor)	72.258/64.871				
Average time in system (shop floor)	2.524/11.022				
Average total time in queues (shop floor)	1.448/9.946				
Maximal size of WIP (shop floor)	48.8(69)/216(380)				
maximal size of ACO operations	138.8(198)/734.4(1335)				
average size of ACO operations	59.8/285.8				

Table 6.2. The effects of pheromone adaptation – Problem 2

Mean job inter-arrival time: 1/1 hour, 9 jobs/batch ACO (with/without adaptation) (10 ants) 120 types of jobs (randomly)			Number of machines: 4, 2, 5, 3, 2 Simulation time: 200 hours Warming up time: 20 hours		
performance measure	**1**	**2**	**3**	**4**	**5**
Proportion machines busy (workcenter)	0.438/0.462	0.922/0.924	0.364/0.361	0.617/0.617	0.935/0.935
Average number in queue (workcenter)	3.482/3.488	27.839/29.382	2.420/2.445	5.564/5.523	30.195/29.774
Maximum number in queue (workcenter)	15.8(17)/ 17.6(21)	70.0(84)/ 80.4(86)	15.2(18)/ 15.2(17)	21.0(28)/ 22.4(26)	69.8(87)/ 71.4(91)
Average daily throughput (shop floor)	73.973/73.929				
Average time in system (shop floor)	8.426/8.545				
Average total time in queues (shop floor)	7.350/7.469				
Maximal size of WIP (shop floor)	151.4(178)/152.6(178)				
maximal size of ACO operations	565.8(669)/569.4(683)				
average size of ACO operations	275.4(364)/278.8				

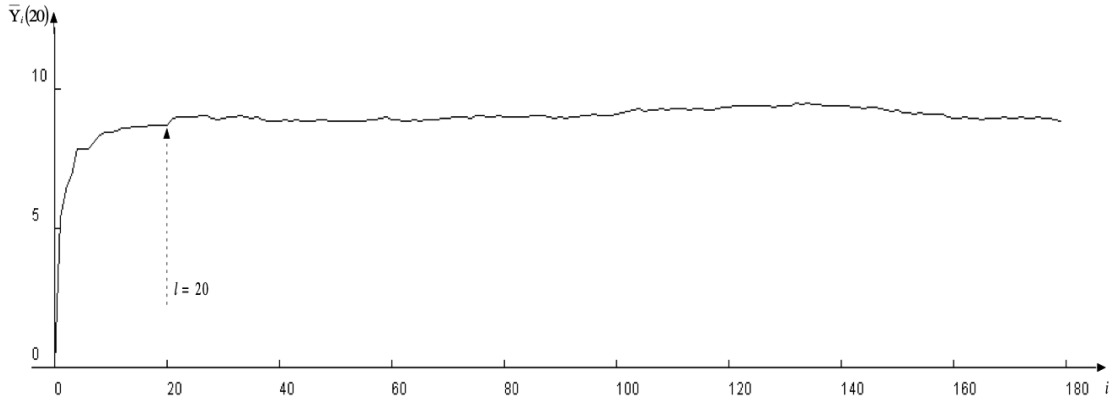


Fig. 6.2. Moving average of hourly throughputs of problem 1 with adaptation

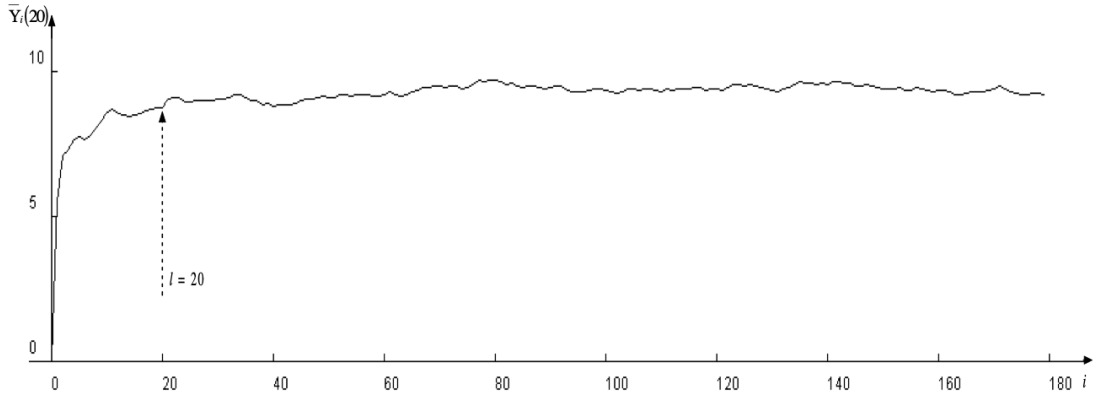


Fig. 6.3. Moving average of hourly throughputs of problem 2 with adaptation

Summary and Conclusion

A basic version of ACO has been applied to two dynamic JSSPs with the same workloads but different dynamic levels and disturbing severity. The computational results show: 1) the ACO performs effectively in both cases; 2) the adaptation mechanism of the ACO does have effects in situations where disturbances are slight but have little effects in situations where disturbances are severe; and 3) improving the optimality of immediate schedules but sacrificing the flexibility of the pheromone matrix may lead to an inferior long-term performance.Two important ACO parameters, namely the number of iterations and the size of ants per iteration, are tuned in order to improve the overall performance under the same problem settings.

Table 6.3. Increase the number of iterations – Problem 1

Mean job inter-arrival time: 1/9 hour, 1 job/lot ACO (s_{min} = 25/40 iterations) (10 ants) 120 types of jobs (randomly)			Number of machines: 4, 2, 5, 3, 2 Simulation time: 200 hours Warming up time: 20 hours		
performance measure	**1**	**2**	**3**	**4**	**5**
Machine utilization (workcenter)	0.404/0.419	0.902/0.826	0.355/0.332	0.564/0.560	0.916/0.835
Average number in queue (workcenter)	0.703/6.040	5.558/37.344	0.404/3.113	1.297/15.989	5.838/37.085
Maximum number in queue (workcenter)	7.0(10)/ 24.4(35)	21.2(28)/ 78.2(140)	5.2(6)/ 13.4(27)	9.4(12)/ 45.4(75)	21.0(31)/ 75.4(117)
Average daily throughput (shop floor)	72.258/64.693				
Average time in system (shop floor)	2.524/11.458				
Average total time in queues (shop floor)	1.448/10.382				
Maximal size of WIP (shop floor)	48.8(69)/222(383)				
maximal size of ACO operations	138.8(198)/778.2(1362)				
average size of ACO operations	59.8/300.2				

Table 6.4. Increase the number of iterations – Problem 2

Mean job inter-arrival time: 1/1 hour, 9 jobs/lot ACO (s_{min} = 25/40 iterations) (10 ants) 120 types of jobs (randomly)			Number of machines: 4, 2, 5, 3, 2 Simulation time: 200 hours Warming up time: 20 hours		
performance measure	**1**	**2**	**3**	**4**	**5**
Machine utilization (workcenter)	0.438/0.459	0.922/0.918	0.364/0.334	0.617/0.530	0.935/0.930
Average number in queue (workcenter)	3.482/4.640	27.839/31.667	2.420/3.436	5.564/8.295	30.195/32.961
Maximum number in queue (workcenter)	15.8(17)/ 19.0(30)	70.0(84)/ 73.6(90)	15.2(18)/ 16.4(24)	21.0(28.0)/ 25.2(37)	69.8(87)/ 73.6(86)
Average daily throughput (shop floor)	73.973/73.138				
Average time in system (shop floor)	8.426/9.657				
Average total time in queues (shop floor)	7.350/8.581				
Maximal size of WIP (shop floor)	151.4(178)/164(194)				
maximal size of ACO operations	565.8(669)/598.6(687)				
average size of ACO operations	275.4(364)/302.2(413)				