Metaheuristic Techniques for Job Shop Scheduling Problem and a Fuzzy Ant Colony Optimization Algorithm

Sezgin Kılıç[1] and Cengiz Kahraman[2]

[1] Industrial Engineering Department, Air Force Academy, 34807, Yeşilyurt, İstanbul, Turkey

[2] Industrial Engineering Department, Istanbul Technical University, Maçka, İstanbul, Turkey

Summary: Job shop scheduling (JSS) problem is NP-hard in its simplest case and we generally need to add new constraints when we want to solve a JSS in any practical application area. Therefore, as its complexity increases we need algorithms that can solve the problem in a reasonable time period and can be modified easily for new constraints. In the literature, there are many metaheuristic methods to solve JSS problem. In this chapter, the proposed Ant algorithm can solve JSS problems in reasonable time and it is very easy to modify the artificial ants for new constraints. In addition, it is very easy to modify artificial ants for multi-objective cases.

Key words: Job shop scheduling, ant colony optimization, metaheuristic methods, tabu search, simulated annealing.

1 Job Shop Scheduling Problem

Scheduling is the allocation of resources over time to perform a collection of tasks. It has been examined in the operation research literature since the early fifties (Conway *et al.,* 1967). Variations in problem types are mostly illustrated using the manufacturing domain. For example, a job is a term used to designate a single item or batch of items that require processing on the machines and the processing of a particular job through a particular machine is called an operation. The Job Shop Scheduling (JSS) problem is a well known NP-hard problem (Lawler *et al.,* 1982), meaning that there is no algorithm that can solve such problems in polynomial time with respect to problem size. JSS is the most general case of the scheduling problems and consists of a finite set of jobs to be processed on a finite set of machines. Each job is characterized by a fixed order of operations, each of which is to be processed on a specific machine for a specific duration. Each machine can handle only one job and each job can be processed by only one machine at a time. Each operation needs to be processed during an uninterrupted period of time on a given machine. A schedule is an assignment of operations to time slots on the machines. We can formulate a JSS problem with *n* jobs to be scheduled on *m* machines as an integer programming model.

Let c_{ik} denote the completion time of job *i* on machine *k* as the decision variable of the model and t_{ik} denotes the processing time of job *i* on machine *k*. If job *i* must be processed on machine *h* before machine *k*, we need the following constraint:

$$c_{ik} - t_{ik} \geq c_{ih} \tag{1.a}$$

On the other hand, if the processing on machine *k* comes first, the constraint becomes

$$c_{ih} - t_{ik} \geq c_{ik} \tag{1.b}$$

Thus, we need to define an indicator variable x_{ihk} as follows:

$$x_{ihk} = \begin{matrix} \text{1, if processing on machine } h \text{ precedes that on machine k for job } i \\ \text{0, otherwise} \end{matrix}$$

We can then rewrite the aforementioned constraints as follows;

$$c_{ik} - t_{ik} + L(1 - x_{ihk}) \geq c_{ih}, \qquad i = 1,2,...,n \quad h,k = 1,2,...,m \quad h \neq k \tag{1}$$

where, *L* is a large positive number. Consider two jobs, *i* and *j*, that are to be processed on machine *k*. If job *i* comes before job *j*, we need the following constraint:

$$c_{jk} - c_{ik} \geq t_{jk} \tag{2.a}$$

Otherwise, if job *j* comes first, the constraint becomes

$$c_{ik} - c_{jk} \geq t_{ik} \tag{2.b}$$

Therefore, we also need to define another indicator variable y_{ijk} as follows:

$$y_{ijk} = \begin{cases} 1, \text{if job } i \text{ precedes job } j \text{ on machine } k, \\ 0, \text{otherwise} \end{cases}$$

We can then rewrite the aforementioned constraints as follows;

$$c_{jk} - c_{ik} + L(1 - y_{ijk}) \geq t_{jk,} \quad i, j = 1,2,...,n, \quad k = 1,2,...,m \quad i \neq j \tag{2}$$

The JSS problem with a makespan objective can be formulated as follows:

$$\min \max_{1 \leq k \leq m} \left[\max_{1 \leq i \leq n} [c_{ik}] \right]$$

$$s.t.$$

$$c_{ik} - t_{ik} + L(1 - x_{ihk}) \geq c_{ih}$$

$$c_{jk} - c_{ik} + L(1 - y_{ijk}) \geq t_{jk}$$

$$c_{ik} \geq 0$$

$$x_{ihk} = 0 \; or \; 1$$

$$y_{ijk} = 0 \; or \; 1$$

$$h, k = 1,2,...,m \quad i, j = 1,2,...,n \quad h \neq k \quad i \neq j$$

The model mentioned above is a general case for JSS problems. It can be specified by changing the objective function and adding specific constraints for desired purposes.

2 Solution Methodologies for JSS Problem

The JSS is amongst the hardest combinatorial optimization problems. One of the earliest works on scheduling theory is Johnson's (1954). Up to time many researchers tried to generate optimum or near optimum schedules to the JSS problem by various techniques. An *n x m* size JSS has an upper bound of $(n!)^m$ possible solutions, thus a 15 × 15 problem may have at most 5.591×10^{181} different solutions (with minimized idle times). Due to this factorial explosion only small instances can be solved with exact methods in acceptable time periods and problems of dimensions greater than 15 × 15 are usually considered to be beyond the reach of the exact methods.

2.1 Main Methodologies Used to Solve JSS Problem

There exists a rapid and extensively growing body of literature for JSS. Basically we can cluster the techniques used to solve JSS into two groups; optimization algorithms and approximation algorithms. Various techniques applied to solve JSS are given below and a comprehensive survey of JSS techniques can be found in Jain and Meeran (1999).

- **Optimization algorithms**
  - Efficient methods – solvable in polynomial time
  - Enumerative methods
    - Mathematical Formulations
      - *Integer Linear Programming*
      - *Mixed Integer Linear Programming*
      - *Langrangian Relaxation*
      - *Surrogate (constraint) Duality*
      - *Decomposition Techniques*
    - Branch and Bound
- **Approximation Algorithms**
  - Constructive Methods
    - *Priority Dispatch Rules*
    - *Insertion Algorithms*
    - *Bottleneck Based Heuristics*
  - Iterative Methods (General Algorithms)
    - Artificial Intelligence (AI)
      - *Constraint Satisfaction*
      - *Neural Networks*
      - *Expert Systems*
      - *Ant Optimization*
  - Local Search
  - Problem Space Methods
    - *GRASP*
    - *Genetic Algorithms*
    - *Reinsertion Algorithms*
    - *Threshold Algorithms*
      - *Simulated Annealing*
      - *Threshold Accepting*
      - *Iterative Improvement*
  - Tabu Search
  - Large Step Optimization

Most of the recent research approaches for JSS are dispatching rules, decomposition methods, and metaheuristic search techniques. Simple heuristic rules are dominantly used in JSS research area. Thus, there are a considerable number of dispatching rules that have been developed across many industries (Zoghby, 2004). With simple dispatch heuristics we schedule a job when it arrives at an empty machine and when the machine is available; we simply schedule the highest priority job currently available at the machine.

2.2 Metaheuristic Methods for JSS Problem

Metaheuristics are semi-stochastic approaches for solving a variety of hard optimization problems. The main differences between traditional mathematical methods and metaheuristics can be determined based on solution optimality and CPU time. The beauty of traditional mathematical methods is that they provide an optimal solution to the stated problem. However, when large size instances are treated, the CPU time required to achieve an optimal solution becomes an issue. In addition, traditional approaches most often require simplifications of the original problem formulation. Some times due to oversimplifications, computed solution does not solve the original problem. Furthermore, one of the fundamental advantages of the metaheuristics approach over traditional mathematical methods is its capability to adapt to the considered problem (Back *et al.*, 1997). All metaheuristics should not be considered as ready to use algorithms but rather as a general concept that can be applied to most of the real world applications. In many cases metaheuristics are augmented with additional approaches to enhance performance for specific problems. Genetic algorithms, Tabu Search, Simulated Annealing and Ant Colony Optimization are among the most successful ones. The success of these methods depends on their ability to find good (not necessarily optimum) solutions to the hard combinatorial optimization problems in short time periods. They are generally easy to implement and they can handle specific constraints in practical applications easily. In addition, developing models for solving difficult combinatorial optimization problems characterized with uncertainty is a very important and challenging research task. Metaheuristic techniques need to be combined with fuzzy sets theory techniques for solving complex problems characterized with uncertainty.

2.2.1 Genetic Algorithms (GA)

Genetic Algorithms (GAs) are adaptive heuristic search algorithm premised on the evolutionary ideas of natural selection and genetic. The basic concept of GAs is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. They represent an intelligent exploitation of a random search within a defined search space to solve a problem. First pioneered by John Holland (1962), Genetic Algorithms have been widely studied, experimented and applied in many fields in engineering worlds. One of the first attempts to approach a simple JSS problem through the application of GAs can be seen in the research of Davis (1985). Since then, a significant number of GAs to JSS problems have been appearing. The first applications of GAs for fuzzy JSS problem were made by Ishii (1995), Tsujimura (1995), and Sakawa (1995). In these works, the values of processing times and duedates were considered as fuzzy numbers due to man-made factors.

2.2.2 Tabu Search (TS)

The basic concept of Tabu Search as described by Glover (1986) is “a meta-heuristic superimposed on another heuristic”. The overall approach is to avoid entrainment in cycles by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited (hence "tabu"). The Tabu search begins by marching to a local minima. To avoid retracing the steps used, the method records recent moves in one or more Tabu lists. The original intent of the list was not to prevent a previous move from being repeated, but rather to insure it was not reversed. At initialization the goal is make a coarse examination of the solution space, known as 'diversification', but as candidate locations are identified the search is more focused to produce local optimal solutions in a process of 'intensification'. The Tabu search has traditionally been used on combinatorial optimization problems. Many of the applications in the literature involve integer programming problems, scheduling, routing, traveling salesman and related problems. Dell’Amico and Trubian (1991) have solved the makespan job shop problem to obtain higher accuracy results at high computation times than earlier tabu search results.

2.2.3 Simulated Annealing (SA)

As its name implies, the Simulated Annealing (SA) exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system. The algorithm is based upon that of Metropolis *et al.* (1953), which was originally proposed as a means of finding the equilibrium configuration of a collection of atoms at a given temperature. The connection between this algorithm and mathematical minimization was first noted by Pincus (1970), but it was Kirkpatrick *et al.* (1983) who proposed that it form the basis of an optimization technique for combinatorial (and other) problems. SA's major advantage over other methods is an ability to avoid becoming trapped at local minima. The algorithm employs a random search which not only accepts changes that decrease objective function, but also some changes that increase it. Van Laarhoven *et al.* (1992) used SA for finding the minimum makespan in JSS problem. They found that SA is not effective and efficient as some other heuristics they used.

2.2.4 Ant Colony Optimization (ACO)

Real ants leave on the ground a deposit called *pheromone* as they move about, and they use pheromone trail to communicate with each other for the purpose of finding shortest path between food resource and their nest. Ants tend to follow the way in which there is more pheromone trail. When an obstacle is placed on their existing path, which is between food and nest, some ants will prefer to go around it by the left side and the others by the right side. Those that have chosen the shortest path will rejoin the previous path more quickly, and this will result a more pheromone trail on the shorter path, and more ant will be attracted to the shorter path. In ACO algorithms, artificial ants with the above described characteristics and the additional features stated below (Dorigo, 1999) collectively search for good quality solutions to the optimization problem.

- Artificial ants live in a discrete world and their moves consist of transitions from discrete states to discrete states.
- Artificial ants have an internal state. This private state contains the records of its past actions.
- Artificial ants deposit an amount of pheromone, which is a function of the quality of the solution found.
- Artificial ants timing in pheromone laying is problem dependent and often does not reflect real ants behavior. For example, in many cases artificial ants update pheromone trails only after having generated a solution.
- To improve overall system efficiency, ACO algorithms can be enriched with extra capabilities like look ahead, local optimization, backtracking, and so on, that can not be found in real ants.

Based on the generic framework of ACO, numerous ant algorithms have been developed. They are named according to the different procedures used within them. Ant System (AS) (Dorigo *et al.*, 1991, Dorigo, 1992) is the first algorithm to fall into the framework of ACO and it was introduced using the Traveling Salesman Problem (TSP) as an example application. Although it was found to be inferior to state-of-the-art algorithms for TSP, its importance mainly lies in the inspiration it provided for a number of extensions that significantly improved performance and are currently among the most successful ACO algorithms (Dorigo and Stützle, 2004).

In this section we present the AS to be a basis for the proposed algorithm in Sect. 4. Artificial ants construct solutions for Travelling Salesman Problem (TSP) by moving on the related graph from one city to another according to the AS rules. In each iteration, each ant moves from one city to another. If the problem is consisting of N cities, after N iterations every ant will make a complete tour. During each iteration, each ant applies a probabilistic decision rule to select the next city that will be added to the ants' tour (3)

$$P_{ij}^{r}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\gamma} \times [\eta_{ij}]^{\theta}}{\sum\limits_{j \notin tabu_k} [\tau_{ij}(t)]^{\gamma} \times [\eta_{ij}]^{\theta}} & if\ j \notin tabu_k \\ 0 & if\ j \in tabu_k \end{cases} \tag{3}$$

$P_{ij}^{r}(t)$: the transition probability from town i to town j for the r_{th} ant at time t.

$\tau_{ij}(t)$: the intensity of the trail on edge (i,j) at time t.

η_{ij} = 1/d_{ij} : visibility of town j from town i

d_{ij} : distance between town i and town j

$tabu_r$: dynamically growing set which contains the tabu list of the r_{th} ant

γ , θ : parameters that control the relative importance of trail versus visibility

Tabu list saves the towns visited for each ant in order to forbid the ant to visit them again. When a tour is completed, it shows the solution generated by each ant. At time t, each ant chooses the next town, where it will be at time $t + 1$. If there are m ants, m moves will be carried out in the interval $(t,t + 1)$, and when the iteration number is N, a cycle will be completed. When a cycle is completed, every ant will have generated a complete tour and the trail intensities at the edges will be updated according to the equation (4)

$$\tau_{ij}(t + N) = \rho.\tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{4}$$

ρ : the coefficient of evaporation ($0 < \rho < 1$)

$$\Delta\tau_{ij} = \sum_{r=1}^{R} \Delta\tau_{ij}^{r}$$

$\Delta\tau_{ij}^{r}$: the quantity of trail substance laid on edge (*i,j*) by the r_{th} ant at the end of the each cycle.

$$\Delta\tau_{ij}^{r} = \begin{cases} \frac{Q}{L_r}, & \text{If } r_{th} \text{ ant used edge } (i, j) \text{ in its tour} \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

Q is a constant and L_r is the tour length of the r_{th} ant.
Formally the AS algorithm is (Dorigo *et al.*, 1996):

```
Step-1. Initialize:
        Set t: = 0                (t is the time counter)
        Set NC: = 0        (NC is the cycle counter)
        For every edge (i,j) set an initial value $\tau_{ij}(t)$ = c for trail intensity and
        $\Delta\tau_{ij} = 0$
Step-2. Set s: = 1          (s is the tabu list index)
        For r: = 1 to R do
                Place the starting town of the $r_{th}$ ant in tabu$_r$(s)
Step-3. Repeat until tabu list is full
        Set s: = s+1
        For r: = 1 to R do
                Choose the town j to move to, with probability $P_{ij}^{r}(t)$ given by
                equation (3)
                Move the $r_{th}$ ant to the town j
                Insert town j in tabu$_r$(s)
Step-4. For r: = 1 to R do
                Move the $r_{th}$ ant from tabu$_r$(N) to tabu$_r$(1)
                Compute the length $L_r$ of the tour described by r-th ant
                Update and save the shortest tour found
        For r: = 1 to R do
                Compute $\Delta\tau_{ij}^{r}$ by equation (5)
                $\Delta\tau_{ij} = \Delta\tau_{ij} + \Delta\tau_{ij}^{r}$
Step-5. For every edge (i,j) compute $\tau_{ij}(t+N)$ according to equation (4)
        Set t: = t+N
        Set NC : = NC+1
        For every edge (i,j) set $\Delta\tau_{ij} = 0$
Step-6. If (NC < NC$_{max}$) and there is not stagnation behaviour
then
        Empty all tabu lists
        Goto Step-2
else
        Print the shortest tour
        Stop
```

Numerous successful implementations of the ACO metaheuristics have been applied to many different combinatorial optimization problems like routing, assignment, scheduling, subset and network routing (Dorigo and Stützle, 2004). The applications for scheduling problems are summarized in Table 1.

Table 1. Applications of ACO for scheduling (Dorigo and Stützle, 2004)

Problem Name	Main References
Job Shop	Colorni, Dorigo, Maniezzo, Trubian (1994)
Open Shop	Pfahringer (1996)
Flow Shop	Stützle (1998)
Total Tardiness	Bauer, Bullnheimer, Hartl, Strauss (2000)
Total Weighted Tardiness	den Besten, Stützle, Dorigo (2000)
	Merkle, Middendorf (2000, 2003)
	Gagne, Price and Gravel (2002)
Project Scheduling	Merkle, Middendorf and Schemeck (2000, 2002)
Group Shop	Blum (2002, 2003)

Many researchers used ACO algorithms to solve various types of the scheduling problems and it can be seen that ACO is among the best-performing approaches for some types of scheduling problems such as single-machine total weighted tardiness problem, the open shop problem, and the resource constrained project scheduling problem. However, ACO research results are far behind the state of the art for classic scheduling problems, like the permutation flow shop problem and the job shop problem, which will be in our interest in this work.

3 Fuzzy Job Shop Scheduling Problem

3.1 Fuzzy Sets and Fuzzy Logic

Fuzzy sets were introduced by Zadeh in 1965 to represent/manipulate data and information possessing nonstatistical uncertainties. Fuzzy logic provides an inference morphology that enables approximate human reasoning capabilities to be applied to knowledge-based systems. The theory of fuzzy logic provides a mathematical strength to capture the uncertainties associated with human cognitive processes, such as thinking and reasoning.

There are two main characteristics of fuzzy systems that give them better performance for specific applications;

- Fuzzy systems are suitable for uncertain or approximate reasoning, especially for the system with a mathematical model that is difficult to derive.
- Fuzzy logic allows decision making with estimated values under incomplete or uncertain information.

3.1.1 Operations with Fuzzy Numbers

If we denote a triangular fuzzy number $\tilde{A}$ by a triplet (a^1, a^2, a^3) then addition of two positive triangular fuzzy numbers $\tilde{A}$ and $\tilde{B}$ is calculated by the following formula:

$$\tilde{A} + \tilde{B} = (a_1, a_2, a_3) + (b_1, b_2, b_3) = (a_1 + b_1, a_2 + b_2, a_3 + b_3)$$

The addition of two triangular fuzzy numbers will also be a triangular fuzzy number.

According to the extension principle of Zadeh, the membership function $\mu_{\tilde{A}\vee\tilde{B}}(z)$ or $\tilde{A} \vee \tilde{B}$ through the $\vee$ (max) operation becomes as follows:

$$\mu_{\tilde{A}\vee\tilde{B}}(z) = \sup_{z=x\vee y} \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(y))$$

The result of the $\vee$ (max) operation does not always become a triangular number. For simplicity, we approximate the $\vee$ (max) operation with the following formula as in Sakawa and Kubota (2000).

$$\tilde{A} \vee \tilde{B} = (a_1, a_2, a_3) \vee (b_1, b_2, b_3) \approx (a_1 \vee b_1, a_2 \vee b_2, a_3 \vee b_3) \tag{6}$$

The ranking method used in this work involves three ordered criteria (Kaufmann and Gupta, 1988);

Criterion 1. The greatest associate ordinary number, $C_1(\tilde{A}) = \dfrac{a^1 + 2a^2 + a^3}{4}$, is used as a first criterion for ranking.

Criterion 2. If C_1 does not rank the fuzzy numbers, those which have the best maximal presumption $C_2(\tilde{A}) = a^2$ will be chosen as a second criterion.

Criterion 3. If C_1 and C_2 do not rank the fuzzy numbers, the difference of the spreads $C_3(\tilde{A}) = a^3 - a^1$ will be used as a third criterion.

3.2 Fuzzy Job Shop Scheduling Problem

Although the JSS problem has often been investigated, very little of this research is concerned with the uncertainty characterized by the impression in problem variables. In most of the work about JSS it is assumed that the problem data are known precisely at the advance or the prevalent approach to the treatment of the uncertainties in the JSS problem is use of probabilistic models. It is obvious that, if there exists a human interaction in the system there will be uncertainties in processing times. However, the evaluation and optimization of probabilistic models is computationally expensive and the use of probabilistic models is realistic only when description of the uncertain parameters is available from the historical data (Balasubramanian, 2003).

The main application area of fuzzy set theory to scheduling is in the systematic framework it provides for the representation, application, propagation and relaxation of imprecise constraints, and the evaluation of schedules with respect to vaguely defined goals. The desirability of a particular schedule is given by the degree to which it simultaneously satisfies all the goals and constraints, which may be interpreted as the schedule's degree of membership of the intersection of fuzzy constraint/goal sets (Slany, 1994).

McCahon and Lee (1992) were the first to illustrate the application of fuzzy set theory as a means of analyzing performance characteristics for a flow shop problem. Ishii *et al.* (1992) investigated two machine open shop problems with maximum lateness criteria and an identical machine scheduling problem with fuzzy due dates. Fortemps (1997) used simulated annealing in order to minimize the makespan of JSS with fuzzy durations. That was the first significant application which considers the uncertainty in time parameters, six-point fuzzy numbers were used to represent fuzzy durations. Flexible solutions which can cope with all possible durations were generated for benchmark fuzzified problems. Balasubramanian and Grossmann (2003) present a mixed integer linear programming (MILP) for flow shop scheduling with fuzzy task durations and compute optimistic and pessimistic values of makespan. The model was computationally tractable only for reasonably sized problems. Sakawa and Kubota (2000) introduced fuzzy job shop scheduling problems (FJJS) by incorporating the fuzzy processing time and fuzzy duedate. On the basis of the agreement index of fuzzy duedate and fuzzy completion time, multi objective FJJS problems have been formulated as three-objective ones which not only maximizes the minimum agreement index but also maximize the average agreement index and minimize the maximum fuzzy completion time. They proposed a genetic algorithm for generating solutions for multi objective FJSS problem. Lin (2002) considered a FJSS problem with imprecise processing time. He used fuzzy numbers and level $(\lambda,1)$ interval valued fuzzy numbers for the representation of vague processing times. He used Johnson's constructive algorithm for JSS problems. Ghrayeb (2003) presented a bi-criteria genetic algorithm approach to solve FJSS problem, in which the integral value and the uncertainty of the fuzzy makespan, which are conflicting objectives, are minimized. Fuzzified benchmark problems were used to show the effectiveness of the proposed approach.

Using fuzzy numbers to represent the uncertainty in processing times is very plausible for real world applications. If a decision maker estimates the processing time (t_{jk}) of the job j on machine k as an interval rather than a crisp value then the interval can be represented as a fuzzy number. The use of interval $\left[t_{jk} - \Delta_{jk1}, t_{jk} + \Delta_{jk2}\right]$ is more appropriate than the crisp t_{jk} value. The decision maker should carefully determine the parameters Δ_{jk1} and Δ_{jk2}, which satisfy $0 < \Delta t_{jk1} < t_{jk}$ and $0 < \Delta t_{jk2}$. As shown in Fig. 1 the fuzzy processing time is represented by a triangular fuzzy number.

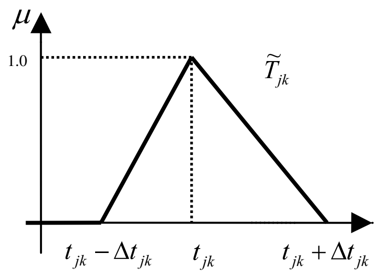

Fig. 1. Fuzzy processing time

In addition, as in many real-world situations, if a certain amount of delay (Δd_j) may be tolerated we can also use fuzzy numbers for due dates, Fig. 2. The fuzzy due date of job j is represented by the degree of satisfaction with respect to the job completion time.

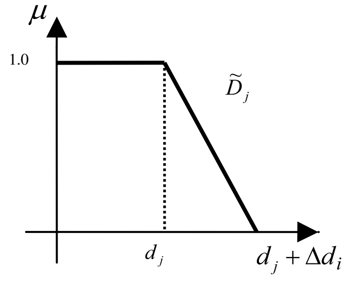

Fig. 2. Fuzzy due date

As we use fuzzy processing times for jobs, the completion time of the each job will also be a triangular fuzzy number (TFN). The fuzzy completion time for each job is denoted by $\tilde{C}_i$. Now we can model the FJSS similar to the crisp JSS model but with fuzzy processing times and fuzzy due dates;

$$\tilde{C}_{ik} - \tilde{T}_{ik} + L(1 - x_{ihk}) \geq \tilde{C}_{ih} \qquad i = 1,2,...,n \qquad h,k = 1,2,...,m \tag{8}$$

$$\tilde{C}_{jk} - \tilde{T}_{jk} + L(1 - y_{ijk}) \geq \tilde{C}_{ik} \qquad i,j = 1,2,...,n \qquad k = 1,2,...,m \tag{9}$$

$$\begin{array}{lll} \tilde{C}_{ik} \geq 0 & i = 1,2,...,n & k = 1,2,...,m \\ x_{ihk} = 0 \; or \; 1 & i = 1,2,...,n & h,k = 1,2,...,m \\ y_{ijk} = 0 \; or \; 1 & i,j = 1,2,...,n & k = 1,2,...,m \end{array}$$

It can be shown that if x_{jhk} = 1 and y_{ijk} = 1, equations (7) and (8) implies
$$\tilde{C}_{jk} \geq (\tilde{C}_{jh} \vee \tilde{C}_{ik}) + \tilde{T}_{jk} \qquad i,j = 1,2,...,n \quad h,k = 1,2,...,n$$
according to the extension principle.

As an example assume that we are generating a schedule and Job-1 must be processed on machine-4 before machine-5 and Job-2 is the last job scheduled on machine 5 up to now with completion time (8,10,14). The completion time of the Job-1 on machine-4 is (7,12,13). If we want to assign Job-1 on machine-5, the minimum completion time of Job-1 on machine-5 can be calculated as;
$$\tilde{C}_{15} = (\tilde{C}_{14} \vee \tilde{C}_{25}) + \tilde{T}_{15}$$
$$\tilde{C}_{15} = \left[(7,12,13) \vee (8,10,14)\right] + \tilde{T}_{15} = (8,12,14) + \tilde{T}_{15}$$

For an objective function which minimizes the makespan of the schedule we can use the following one;

Minimize $z = \max \tilde{C}_i \qquad i = 1,...,n$

However the aim of the schedule might be generating a schedule in order to minimize the lateness of maximum late job from its due date. If this is the case, we can use the agreement indices (Sakawa and Kubota, 2000). The agreement index (AI) expresses how much the $\tilde{C}_i$ is inside the boundaries of the fuzzy due date ($\tilde{D}_i$). As shown in Fig. 3.;

AI = [area ($\tilde{C}_i \cap \tilde{D}_i$)]/[area ($\tilde{C}_i$)]

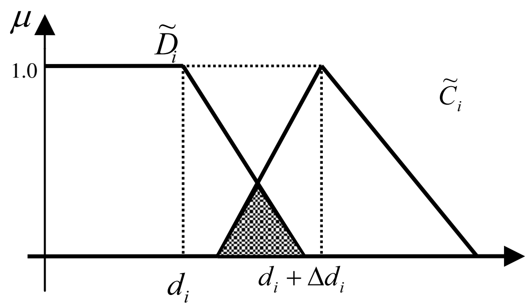

Fig. 3. Agreement index

And, the objective function will be;

$$\text{Maximize } z = \min AI_i \qquad i = 1,\ldots,n \tag{9}$$

4 Fuzzy Ant Colony Optimization Algorithm for Fuzzy JSS Problem

In this section we propose an implementation of fuzzy logic in ACO framework for JSS problems. ACO algorithms show good performance in solving problems that are combinatorial in nature. However, some of the real-life problems are characterized both by uncertainty and by combinatorial in nature. There may be four main types of imprecision in the JSS problems. They are fuzzy processing time, fuzzy due date, fuzzy preferences and fuzzy objective functions with fuzzy criteria (Lin, 2002). We tried to handle the fuzziness in JSS problem in Sect. 3 for fuzzy processing times and fuzzy due dates with a objective function which maximizes the minimum agreement index between fuzzy due dates and fuzzy completion times.

Although some work has been done on combining fuzzy rules with ant-based algorithms for optimization problems, to our knowledge until now fuzzy rules have not yet been used to control the behavior of the artificial ants in a scheduling algorithm. In addition there exists only a little work which combines fuzzy logic with ant-based algorithms. Schockaert *et al.* (2004) proposed a clustering algorithm with fuzzy ants. Lucic (2002) proposed a fuzzy ant system (FAS) for transportation problems.

Human operators use subjective knowledge or linguistic information on a daily basis when making decisions. The environment in which a human expert (human controller) makes decisions is often complex, making it difficult to formulate a suitable mathematical model. Thus, the development of fuzzy logic systems seems justified in such situations. The control strategies of artificial ants can be easily formulated in terms of numerous descriptive rules (Lucic, 2002).

Through many publications, the methodology of how ants make the choice where to go in the next iteration has stayed practically unmodified for an entire decade. We make modifications to the classical Ant System algorithm in order to use fuzzy processing time and fuzzy due date. But the basic modification was in adaptation of the fuzzy rules instead of transition probability function in the classical Ant System algorithm. Therefore the proposed algorithm is not only an ant algorithm which can operate with fuzzy variables but also an algorithm which uses fuzzy rules for generating solutions.

In principle, there is an infinite number of feasible schedules for any JSS problem, because arbitrary amounts of idle time can be inserted. However, this is not useful for a given known sequence and regular objectives. By definition a local shift occurs in a feasible schedule if one operation can be moved left to start earlier and keep feasibility. A schedule is semi-active if no local left shifts are available. It is possible to further improve semi-active schedules in many cases. Although an activity may not be moved immediately to the left, it may be possible to "jump" over obstructions to the left and obtain a better schedule. A global left shift occurs if one operation can be shifted into a hole earlier in the schedule and preserve feasibility. The set of all schedules in which no global left shift can be made is called the set of active schedules. The number of active schedules is still much too large for all but the smallest problems when using search. Thus attention often focuses on an even smaller subset called dispatch schedules or nondelay schedules. A dispatch or nondelay schedule is one in which no machine is kept idle at a time when it could begin processing some operation. The set of all feasible schedules may be classified as follows. A set of all nondelay schedules is part of the set of all active schedules, which is part of the set of all schedules (Fig. 4). There is always an active schedule that is optimal for a regular objective. The best nondelay schedule may not be optimal (Morton and Pentico, 1993).

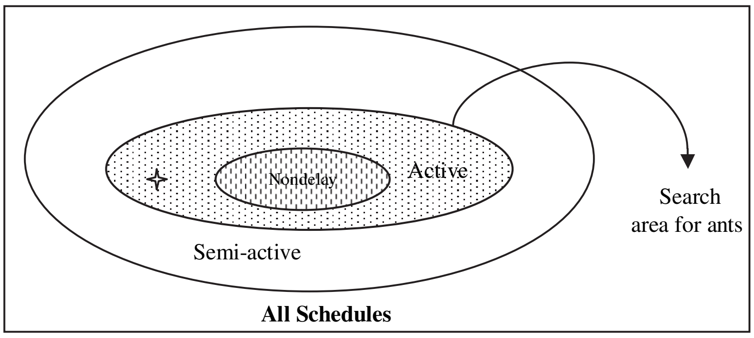

Fig. 4. Venn diagram of schedule relationships

Schedule generation procedures treat operations in an order that is consistent with the precedence relations of the problem. In other words, no operation is considered until all of its predecessors have been scheduled. Once we schedule all the predecessors of an operation, that operation becomes schedulable. Generation procedures operate with a set of schedulable operations at each stage. At each stage, the operations that have already been assigned starting time make up a partial schedule. Let

PS(t) : a partial schedule containing t scheduled operations.

SO(t) : the set of schedulable operations at stage t, corresponding to a given PS(t).

s_j : the earliest time at which operation $j \in SO(t)$ could be started.

f_j : the earliest time at which operation $j \in SO(t)$ could be finished.

For a given active partial schedule, the potential start time for schedulable operation j, denoted s_j, is determined by the completion time of the direct predecessor of operation j and the latest completion time on the machine required by operation j. The larger of these two quantities is s_j. The potential finish time f_j is simply s_j+t_j, where t_j is the processing time of operation j. A systematic approach to generating active schedules works as follows (Baker, 1997).

Active Schedule Generation Algorithm

Step 1. Let t = 0 and begin with PS(t) as the null partial schedule. Initially, SO(t) includes all operations with no predecessors.

Step 2. Determine $f^* = \min_{j \in SO(t)}\{f_j\}$ and the machine k^* on which f^* could be realized.

Step 3. For each operation $j \in SO(t)$ that requires machine k^* and for which $s_j < f^*$, calculate a priority index according to a specific priority rule.

Find the operation with the smallest index and add this operation to PS(t) as early as possible, thus creating partial schedule, PS(t + 1).

Step 4. Update the data set as follows:

(a) Remove operation j from SO(t).
(b) Form SO(t + 1) by adding the direct successor of job *j* to SO(t).
(c) Increment t by one.

Step 5. Return to Step 2 and continue in this manner until an active schedule has been generated.

Proposed Model

The proposed model in this paper searches the best active schedule with artificial ants. Basically, the main steps of the proposed algorithm is as follows,

```
Step 1: Set parameters, initialize the pheromone trails.
Step 2: While (termination condition is not met)
         do the following:
             construct solutions;
             improve the solutions by local search (optional);
             update the pheromone trail or trail intensities;
Step 3: Return the best solution found
        Check the termination criterion:
             Go to Step 2 or Stop.
```

Step 1. Setting parameters and initializing the pheromone trails

Values of the parameters are determined in accordance with our experiences gained in the numerical experimentations and the recommendations based on the previous studies. Number of ants (m) constructing complete solutions at each iteration is equal to the number of jobs (J) to be scheduled. The initial trail intensities are chosen as $\tau_{jn}^{k}=1$, where τ_{jn}^{k} denotes the trail intensity for the j_{th} job for the n_{th} position on machine k. The persistence of trail (ρ) is set to 0.95, where $(1-\rho)$ denotes the evaporation rate.

Step 2. Construction of schedules

An ant starts constructing a schedule by determining f^* and k^* in order to narrow the search space. For each candidate job, $j \in SO(t)$, that requires machine k^* and for which $s_j < f^*$, the ant will have greater or lesser perceived utility towards it depending on the total time of operations remaining to be scheduled for job j, and its trail intensity for the position. Remaining times for jobs and trail intensities may be represented by fuzzy values (Fig. 5 and Fig. 6).

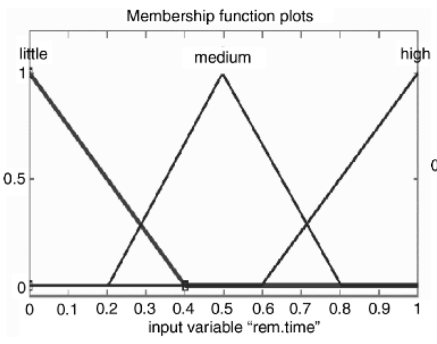

Fig. 5. Fuzzy remaining time

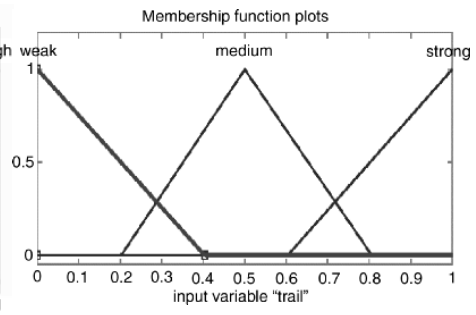

Fig. 6. Fuzzy trail intensity

The approximate reasoning algorithm for calculating the ant's perceived utility of choosing an operation consists of the following 9 rules in Table 2:

Table 2. Fuzzy rule base for artificial ants

Rule	Trail	Remaining Time	Utility
1	WEAK	LITTLE	VERY LOW
2	WEAK	MEDIUM	LOW
3	WEAK	HIGH	MEDIUM
4	MEDIUM	LITTLE	MEDIUM
5	MEDIUM	MEDIUM	HIGH
6	MEDIUM	HIGH	VERY HIGH
7	HIGH	LITTLE	VERY HIGH
8	HIGH	MEDIUM	VERY HIGH
9	HIGH	HIGH	VERY VERY HIGH

As an example, Rule-6 is:

If trail intensity is **MEDIUM** and remaining time is **HIGH**
Then utility is **VERY HIGH**

Graphical representation of the surface of the fuzzy rule base is shown in Fig. 7. It gives the utility value of selecting of a job for a position, according to the rules in Table. 2. This is the main differences between crisp ACO and the fuzzy ACO algorithm. In Sect. 2.2.4 the utility values for candidates were calculated by equation (1). We use the fuzzy rule base instead as represented in Table 2.

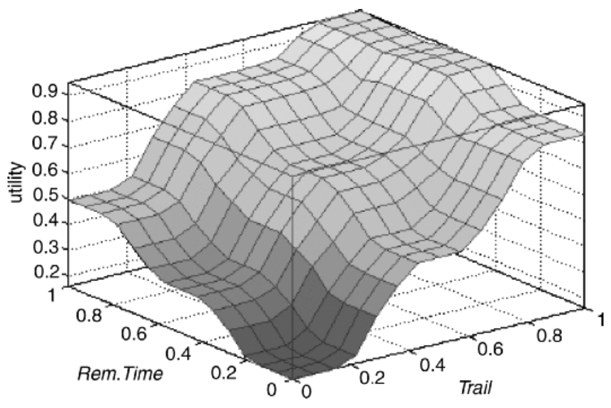

Fig. 7. Utility value of selecting of a job for a position, according to the fuzzy rules

After obtaining utilities for each candidate operation with fuzzy rule base, it is possible to calculate probabilities associated with each operation. The probability of each operation will represent its probability of being selected by the ant.

u_{jn}^{k} : utility of job *j* being scheduled for the n_{th} position of machine *k*.

$$p_{jn}^{k} = \frac{u_{jn}^{k}}{\sum_{j \in m^*} u_{jn}^{k}} \tag{10}$$

Final choice would be obtained in a proportional manner as in crisp ACO algorithm known as the roulette wheel selection. The selected job for machine m* will be added on the Gantt chart of ant r. In this way one ant can generate a complete schedule. Artificial ants live in discrete time, time index (t) is zero at the beginning of the algorithm and it is increased one by one as the jobs are added to the Gantt chart and when t reaches the value of total number of operations a complete schedule is generated. Since there are R ants, total of R schedules will have been generated in a cycle. After finishing a cycle trail intensities will be updated.

The original equation (1) used in the crisp Ant System is replaced by approximate reasoning. In this way, it will also be possible to calculate transition probabilities even if some of the input data were only approximately known.

Updating Trail Intensities

Trail intensities made the jobs more desirable for the specific positions if they had been frequently chosen and the objective function value of the schedule for these positions had been greater at the earlier cycles. Furthermore, evaporation of pheromone lessen the desirability of the jobs for the specific positions if they had been seldom chosen and the objective function value of the schedule for these positions had been smaller at the earlier cycles. In this work we wanted to minimize the lateness of maximum late job from its due date so the agreement indices (AI), defined in section 3.2, are used for determining the quantity of the trails to be left by ants;

Let, $\Delta\tau_{jn}^{kr}$ is the quantity of trail substance to be left, on position n for job j on the k-th machine by ant r, at the end of a cycle. Each ant left trail substance for each job for its position in the schedule as much as the minimum AI value of the jobs of the schedule generated by the considered ant, the quantity of the trail to be left is denoted by;

$$\Delta\tau_{jn}^{kr} = \min \mathrm{AI}_j \qquad j = 1,2,\ldots,n \tag{11}$$

After each cycle, trail intensities are updated by the equation below;
R: total number of ants (r = 1,2,…,R)

$$\tau_{jn}^{k,new} = \rho \times \tau_{jn}^{k,old} + \sum_{r=1}^{R} \Delta\tau_{jn}^{k,r} \tag{12}$$

Step 3.
We should set some termination conditions to stop the search. The algorithm may be stopped if it finds a solution which is known to be a optimal or a near optimal solution. It can also be stopped after it makes a predefined number of iterations. In addition, we can also end the search if it is in a stagnation behavior. If all ants start to construct same solutions because of the high trail intensification on some positions for specific jobs we do not need to wait for algorithm to escape from this behavior because it is almost impossible.

5 An Application for JSS Problem Using Fuzzy ACO

Sakawa and Mori (1999) applied Genetic Algorithm (GA) and Simulated Annealing (SA) to the FJSS problem as shown in Table 3;

Table 3. Fuzzy JSS problem (6 jobs, 6 machines)

	Processing Machines (Fuzzy Processing Time)					
Job 1	1(5,6,13)	5(3,4,5)	2(1,2,3)	6(3,4,5)	4(2,3,4)	3(2,3,4)
Job 2	1(3,4,5)	2(2,4,5)	3(1,3,5)	6(4,5,6)	4(5,6,7)	5(6,7,8)
Job 3	3(1,2,3)	6(5,6,7)	5(4,5,6)	4(3,4,5)	2(1,2,3)	1(1,2,3)
Job 4	6(2,3,4)	5(1,2,3)	4(2,3,4)	2(2,3,5)	1(3,4,6)	3(3,4,5)
Job 5	6(3,4,5)	5(2,3,4)	4(1,2,3)	3(2,3,4)	2(4,5,6)	1(2,3,4)
Job 6	5(6,7,8)	6(4,5,6)	1(2,3,4)	2(3,4,5)	3(2,3,4)	4(1,2,3)
	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6
Fuzzy due date	30,40	35,40	20,28	32,40	30,35	40,45

We also tested the proposed fuzzy ACO for FJSS problem on the same problem and made 10 trails as they did. Results by proposed fuzzy ACO are represented on the Table 4 with the Sakawa and Mori (1999)'s solutions. Results represent the minimum agreement indices for the solutions obtained from each trail. The second term in parenthesis in ACO column represents the first cycle in which the max (min AI_i) is captured.

Table 4. Results for the FJSS problem

Trail	GA	SA	ACO	Trail	GA	SA	ACO
1	0,69	0,59	0,69 (84)	6	0,69	0,38	0,42 (220)
2	0,69	0,39	0,69 (99)	7	0,69	0,38	0,69 (242)
3	0,69	0,36	0,69 (188)	8	0,69	0,38	0,69 (120)
4	0,69	0,36	0,39 (124)	9	0,69	0,36	0,69 (87)
5	0,69	0,39	0,69 (285)	10	0,69	0,36	0,69 (162)

As a termination criteria they use 50 generations with population size 30 for GA (almost 30*50 = 1500 iterations) and 3000 iterations for SA. We run the proposed algorithm for 250 cycles and as there exist 6 ants each generating a solution in each cycle, it makes about 1500 iterations.

It should be also noted that we did not use a local search algorithm to improve solutions generated by artificial ants in order to show the explicit force of the Ant algorithm. It is possible to reach good solutions in shorter time periods with local search algorithms.

6 Conclusion

The JSS problem is NP-hard in its simplest case and we generally need to add new constraints when we want to solve a JSS in any practical application area. Therefore, as its complexity increases we need algorithms that can solve the problem in a reasonable time period and can be modified easily for new constraints. As it is mentioned, exact algorithms can be used in small sized problems due to very long solution times and it can be very hard to modify an exact solution algorithm only for a new constraint. However, the proposed Ant algorithm can solve JSS problems in reasonable time and it is very easy to modify the artificial ants for new constraints. In addition, it is very easy to modify artificial ants for multi-objective cases.

Determining parameter values (γ, θ, ρ, m) for an Ant algorithm is a very difficult and time consuming task for researchers. Generally, they are found after many trails or same values are used from previous works. Parameter values have great effect on solution time and it is also possible to reach optimal solution for specific values for them but after a slight differences in these values it can be impossible to reach near-optimal solutions. In the proposed fuzzy Ant algorithm model, γ and θ parameters, which control the relative importance of trail versus visibility, are not used as crisp values. We stand in fuzzy rule base instead of equation (1) in order to generate transition probabilities. And it became more easier to find appreciate rules than determining crisp parameter values.

JSS problems are generally handled with precise and complete data in theoretical studies. There are various exact and approximate solution techniques for these problems. However, it is a usual case to work with insufficient data in practical applications. As the uncertainty and the problem size increases, such techniques became inadequate to generate applicable solutions. Nevertheless, specialized human experts often use their knowledge and experience when they have to deal with such kind of uncertainties. Therefore, they can usually able to generate near optimum and applicable solutions. Actually, the urgent attribute of the Fuzzy Ant System is its ability to represent specialized human experts' knowledge by using a fuzzy rule base. Accordingly, it can deal with uncertain and incomplete data and generate near optimum solutions in shorter time periods.

References

Adams, J., Balas, E. Zawack, D. (1988) , The Shifting Bottleneck Algorithm for Job-Shop Scheduling. Management Science 34, 391–401.

Back, T., Hammel, U., Schwefel, H.P. (1997), Evolutionary Computation: Comments on the History and Current State, IEEE Transactions on Evolutionary Computation 1,1, 3–17.

Balasubramanian, J., Grossmann, I.E. (2003), Scheduling optimization under uncertainty – an alternative approach. Computers and Chemical Engineering 27, 469–490.

Baker, K.R. (1997), Elements of sequencing and scheduling. Kenneth R. Baker.

Conway, R.W., Maxwell, W.L., and Miller, L.W., (1967), Theory of Scheduling. Addison Wesley.

Dorigo, M., Stützle, T. (2004), Ant Colony Optimization. MIT.

Fortemps, P. (1997). Jobshop scheduling with imprecise durations: a fuzzy approach. IEEE Transactions on Fuzzy Systems 5 (4), 557.

Gen, M., Cheng, R. (1997), Genetic Algorithms & Engineering Design. New York:Wiley.

Ghrayeb, O.A. (2003), A bi-criteria optimization: minimizing the integral value and spread of the fuzzy makespan of job shop scheduling problems. Applied Soft Computing 2/3F, 197–210.

Glover, F. (1986). Future paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research 5, 533–549.

Holland, J.H. (1962), Outline for a logic theory of adaptive systems. Journal of the ACM 3, 297.

Johnson, S.M. (1954), Optimal two-and three-stage production schedules with setup times included. Naval Research Logistics Quartely 1, 61–68.

Jain, A.S., Meeran, S. (1999). Deterministic job-shop scheduling: Past, present and future. European Journal of Operational Research 113, 390–434.

Kaufmann, A., Gupta, M. (1988), Fuzzy Mathematical Models in Engineering and Management Science, North-Holland, Amsterdam.

Kirkpatrick, S., Gerlatt, C. D. Jr., Vecchi, M.P. (1983), Optimization by Simulated Annealing. Science 220, 671–680.

Lin, F. (2002), Fuzzy Job Shop Scheduling Based on Ranking Level (λ,1) Interval Valued Fuzzy Numbers. IEEE Transactions on Fuzzy Systems Vol. 10, No.4.

Lawler, E.L., Lenstra, J.K. Rinnooy Kan, H.G. (1982), "Recent developments in deterministic sequencing and scheduling: A survey ," in Deterministic and Stochastic Scheduling, Dempster, M., Lenstra, J., and Rinnooy Kan, H. Eds. Dordrecht, The Netherlands: Reidel.

Lucic, P. (2002), Modeling Transportation Systems using Concepts of Swarm Intelligence and Soft Computing. PhD thesis, Virginia Tech.

McCahon, C.S., Lee, E.S. (1992), Fuzzy job sequencing for a flow shop. European Journal of Operational Research 62, 294.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M. N., Teller, A.H. Teller, E., (1953). Equations of State Calculations by Fast Computing Machines, J. Chem. Phys. 21, 1087–1092.

Morton, T.E., Pentico, D.W. (1993), Heuristic scheduling systems with applications to production systems and project management, Wiley Series in Engineering and Technology, John Wiley and Sons, Inc.

Pincus, M. (1970), A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems, Operations Research 18, 1225-1228.

Roy, B., Sussmann, B. (1964), Les problémes d'ordonnancement avec contraintes disjonctives, SEMA, Paris, Note DS 9 bis.

Sakawa, M., Kubota, R. (2000), Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms. European Journal of Operational Research 120, 393-407.

Van Laarhoven, P.J.M., Aarts, E. Lenstra, J.K. (1992), Job shop scheduling by simulated annealing. Operations Research 40, 113-125.

Zadeh, L.A. (1965), Fuzzy Sets, Information and Control 8, 338-353.

Zoghby, J., Batnes, J.W., Hasenbein J.J. (2004), Modeling the reentrant job shop scheduling problem with setups for metaheuristic searches. European Journal of Operational Research.