

Faculty of Information & Communication Technologies

Technical Report SUTICT-TR2006.05

# Solution Representation for Job Shop Scheduling Problems in Ant Colony Optimisation

James Montgomery, Carole Fayad<sup>1</sup> and Sanja Petrovic<sup>1</sup>  
15 May 2006



SWIN  
BUR  
NE

SWINBURNE  
UNIVERSITY OF  
TECHNOLOGY

<sup>1</sup>School of Computer Science & IT,  
University of Nottingham

# Solution Representation for Job Shop Scheduling Problems in Ant Colony Optimisation

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Job Shop Scheduling and Solution Construction</b>	<b>3</b>
<b>3</b>	<b>A Real-World JSP</b>	<b>4</b>
<b>4</b>	<b>ACO for a Fuzzy JSP</b>	<b>6</b>
<b>5</b>	<b>Computational Results</b>	<b>8</b>
5.1	Solution quality compared within alternative measures . . . . .	8
5.2	Solution quality compared between values of $\lambda$ . . . . .	9
5.3	CPU time . . . . .	11
<b>6</b>	<b>Conclusions</b>	<b>11</b>

# Solution Representation for Job Shop Scheduling Problems in Ant Colony Optimisation

## Abstract

Ant colony optimisation, a constructive metaheuristic inspired by the foraging behaviour of ants, has been applied to a wide range of problems since its inception. Many of these are production scheduling problems such as the job shop, in which a collection of operations (grouped into jobs) must be scheduled for processing on different machines. In typical ACO applications, solutions are generated by constructing a permutation of the operations, from which a deterministic algorithm can generate the actual schedule. This paper considers an alternative approach in which each machine is assigned one of a number of alternative dispatching rules, which heuristically determines the processing order for that machine. This representation creates a substantially smaller search space that likely contains good solutions. The performance of both approaches is compared on a real-world job shop scheduling problem in which processing times and job due dates are modelled with fuzzy sets. Results indicate that the new approach produces better solutions more quickly than the traditional approach.

*Keywords:* Ant colony optimisation, fuzzy job shop scheduling, solution representation.

## 1 Introduction

Ant colony optimisation (ACO) is a constructive metaheuristic, inspired by the foraging behaviour of ant colonies, that produces a number of solutions over successive iterations of solution construction. During each iteration, a number of artificial ants build solutions by probabilistically selecting from problem-specific solution components, influenced by a parameterised model of solutions (called a pheromone model in reference to ant trail pheromones). The parameters of this model are updated at the end of each iteration using the solutions produced so that, over time, the algorithm learns which solution components should be combined to produce the best solutions. When adapting ACO to suit a problem an algorithm designer must first decide how solutions are to be represented and built (i.e., what base *components* are to be combined to form solutions) and then what characteristics of the chosen representation are to be modelled.

Production scheduling problems consist of a number of jobs, made up of a set of operations, each of which must be scheduled for processing on one of a number of machines. Precedence constraints are imposed on the operations of each job. The majority of ACO

algorithms for these problems represent solutions as permutations of the operations to be scheduled (operations are the base components of solutions), which determines the relative order of operations that require the same machine (see, e.g., [1, 3, 4, 14]). A deterministic algorithm can then produce the best possible schedule given the precedence constraints established by the permutation. This approach is more generally referred to as the *list scheduler algorithm* [3]. An alternative approach is to assign different heuristics to each machine which determine the relative processing order of operations, thereby searching the reduced space of schedules that can be produced by different combinations of the heuristics [5]. Building solutions in this manner may offer an advantage by concentrating the search on heuristically good solutions. This paper compares these two solution representations by using a real-world job shop scheduling problem (JSP).

A formal description of the JSP is given in Section 2, including further details of the two solution construction approaches. Section 3 describes the real-world JSP instance to which both approaches are applied, in which processing times and due dates are modelled by fuzzy sets to reflect the uncertain nature of these in industrial settings. Details of the ACO algorithms developed for the fuzzy JSP are given in Section 4 followed by analyses of their empirical performance in Section 5. Section 6 describes the implications of the results for the future application of ACO to such problems.

## 2 Job Shop Scheduling and Solution Construction

The JSP examined in this study consists of a set of  $n$  jobs  $J_1, \dots, J_n$ , with associated release dates  $r_1, \dots, r_n$  and due dates  $d_1, \dots, d_n$ . Each job consists of a sequence of operations (determined by the processing requirements of the job) that must each be scheduled for processing on one of  $m$  machines  $M_1, \dots, M_m$ . Only one operation from a job may be processed at any given time, only one operation may use a machine at any given time and operations may not be preempted. Two objectives are minimised simultaneously: the average tardiness of jobs  $C_{AT}$  and the number of tardy jobs  $C_{NT}$ .

$$C_{AT} = \frac{1}{n} \sum_{j=1}^n T_j \quad (1)$$

where  $T_j = \max\{0, C_j - d_j\}$  is the tardiness of job  $J_j$  and  $C_j$  is the completion time of job  $J_j$ .

$$C_{NT} = \sum_{j=1}^n u_j \quad (2)$$

where  $u_j = 1$  if  $T_j > 0$ , 0 otherwise. In many benchmark JSP instances, all jobs are released at time zero and do not have due dates, so the objective becomes to minimise the time required to complete all jobs, called the makespan. Many applications of ACO to the JSP have been to this variant (e.g., [3, 4, 12, 16]).

Regardless of whether a JSP instance has release dates, due dates or neither, to generate a solution it is sufficient to determine the relative processing order of operations that

require the same machine. A deterministic algorithm can then produce the best possible schedule given those constraints. Indeed, it is common in ACO applications for the JSP and other related scheduling problems to generate a permutation of the operations, which implicitly determines this relative order (e.g., [1, 3, 4, 14, 16]). These algorithms are restricted to creating permutations that respect the required processing order of operations within each job, which can consequently be called *feasible permutations*.

Different approaches to constructing solutions produce different search spaces. The space of feasible permutations of operations for a JSP is very large (a weak upper bound is  $O(k!)$ , where  $k$  is the number of operations) and is certainly much larger than the space of actual solutions. This space also has a slight bias towards good solutions, which can be exploited by some pheromone models and proves disastrous for others [12]. Another notable feature of this search space is that while all solutions can be reached, solutions (schedules) are represented by differing numbers of permutations.

An alternative approach to building solutions is to assign different *dispatching rules* (i.e., ordering heuristics) to each machine, which subsequently build the actual schedule [5]. The search space then becomes the space of all possible combinations of rules assigned to machines, which is  $O(|D|^m)$  where  $D$  is the set of rules and  $m$  the number of machines. Given a small number of dispatching rules (this study uses four, described in Section 4) it is highly probable that this search space is a subset of the space of all feasible schedules. However, assuming the dispatching rules are individually likely to perform well it is expected that this reduced space largely consists of good quality schedules.

The performance of these two approaches is compared on a real-world JSP instance, described in the next section.

### 3 A Real-World JSP

The data set used for validation of this research has been provided by a printing company, Sherwood Press, in Nottingham, United Kingdom [7]. There are 18 machines in the shop floor, grouped within seven work centres: printing, cutting, folding, card-inserting, embossing and debossing, gathering, stitching and trimming, and packaging.

Jobs follow a pre-determined order. A ‘job bag’ is assigned to record the order of machines a job is processed on, how long it is ‘expected’ to be processed on that machine and what is the job’s ‘promised delivery date’. Due to both machine and human factors, processing times of jobs are uncertain and due dates are not fixed but promised instead. Therefore, fuzzy sets are used to model uncertain processing times of jobs and due dates as well as the decision maker’s preference to the tardiness of each job.

A triangular membership function  $\mu_{\tilde{p}_{ij}}(t) = (p_{ij}^1, p_{ij}^2, p_{ij}^3)$  is used to model the fuzzy processing time  $\tilde{p}_{ij}$  of job  $J_j$  on machine  $M_i$ ,  $i = 1, \dots, m$ ,  $j = 1, \dots, n$ , where  $p_{ij}^1$  and  $p_{ij}^3$  are lower and upper bounds of the processing time, while  $p_{ij}^2$  is the so-called modal point [9]. An example of fuzzy processing time is shown in Fig. 1(a). A trapezoidal fuzzy set  $(d_j^1, d_j^2)$  is used to model the due date  $\tilde{d}_j$  of each job, where  $d_j^1$  is the crisp due date and the upper bound  $d_j^2$  of the trapezoid exceeds  $d_j^1$  by 10%, following the policy of the

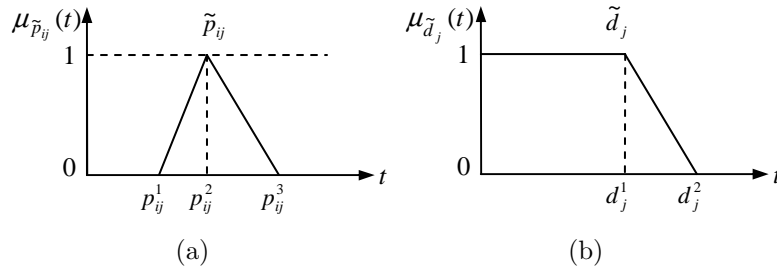


Figure 1: Fuzzy (a) processing time and (b) due date

company. An example of a fuzzy due date is given in Fig. 1(b).

The objective function takes into account both the average tardiness of jobs and the number of tardy jobs. Clearly, the two objectives are measured in different units but have to be used simultaneously to assess the quality of schedules. Values of objectives are mapped onto satisfaction grades, which take values in  $[0, 1]$  and can be combined in an overall satisfaction grade. Each satisfaction grade is calculated taking into consideration the completion times of the jobs, which is fuzzy as a result of dealing with fuzzy processing times.

Two approaches used to measure tardiness in [7] (i.e., to compare fuzzy completion times with fuzzy due dates) are investigated: (1) based on the possibility measure introduced by Dubois and Prade [6], used by Itoh and Ishii [8] to handle tardy jobs in a JSP; and (2) based on the area of intersection measure introduced by Sakawa and Kubota [13].

1. The possibility measure  $\pi_{\tilde{C}_j}(\tilde{d}_j)$  measures the satisfaction grade of a fuzzy completion time  $SG_T(\tilde{C}_j)$  of job  $J_j$  by evaluating the possibility of a fuzzy event  $\tilde{C}_j$  occurring within the fuzzy set  $\tilde{d}_j$  [8] (illustrated in Fig. 2(a)):

$$SG_T(\tilde{C}_j) = \pi_{\tilde{C}_j}(\tilde{d}_j) = \sup \min\{\mu_{\tilde{C}_j}(t), \mu_{\tilde{d}_j}(t)\} \quad j = 1, \dots, n \quad (3)$$

where  $\mu_{\tilde{C}_j}(t)$  and  $\mu_{\tilde{d}_j}(t)$  are the membership functions of fuzzy sets  $\tilde{C}_j$  and  $\tilde{d}_j$  respectively. This measure is referred to as *poss* hereafter.

2. The area of intersection (denoted *area* hereafter) measures the portion of  $\tilde{C}_j$  that is completed by the due date  $\tilde{d}_j$  (illustrated in Fig. 2(b)):

$$SG_T(\tilde{C}_j) = (\text{area } \tilde{C}_j \cap \tilde{d}_j) / (\text{area } \tilde{C}_j) \quad (4)$$

The satisfaction grades of tardiness defined in (3) and (4) are used in two objectives:

1. To maximise the satisfaction grade of *average tardiness*  $S_{AT}$ :

$$S_{AT} = \frac{1}{n} \sum_{j=1}^n SG_T(\tilde{C}_j) \quad (5)$$

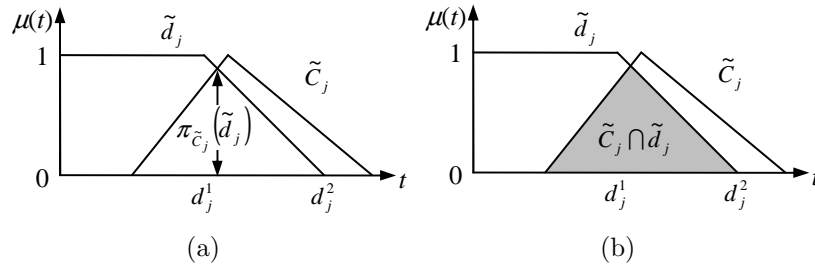


Figure 2: Satisfaction grade of tardiness using (a) possibility measure and (b) area of intersection

2. To maximise the satisfaction grade of *number of tardy jobs*  $S_{NT}$ : A parameter  $\lambda$  is introduced such that a job  $J_j$ ,  $j = 1, \dots, n$ , is considered to be tardy if  $SG_T(\tilde{C}_j) \leq \lambda$ ,  $\lambda \in [0, 1]$ . After calculating the number of tardy jobs  $nTardy$ , the satisfaction grade  $S_{NT}$  is evaluated as:

$$S_{NT} = \begin{cases} 1 & \text{if } nTardy = 0 \\ (n'' - nTardy)/n'' & \text{if } 0 < nTardy < n'' \\ 0 & \text{if } nTardy > n'' \end{cases} \quad (6)$$

where  $n'' = 15\%$  of  $n$ , where  $n$  is the number of jobs.

Two different aggregation operators, which combine the satisfaction grades of the objectives, were investigated:

1. Average of the satisfaction grades:  $F_1 = \frac{1}{2}(S_{AT} + S_{NT})$
2. Minimum of the satisfaction grades:  $F_2 = \min\{S_{AT}, S_{NT}\}$

These two aggregation operators are referred to as *average* and *min* respectively hereafter.

## 4 ACO for a Fuzzy JSP

Two ACO algorithms were developed based on *MAX-MIN* Ant System (*MMAS*), which has been found to work well in practice [15]. The first, denoted *MMAS<sub>perm</sub>*, constructs solutions as permutations of the operations, while the second, *MMAS<sub>rules</sub>*, assigns dispatching rules to machines. The set of dispatching rules  $D$  consists of the following four rules: Early Due Date First, Shortest Processing Time First, Longest Processing Time First and Longest Remaining Processing Time First. Note that the rules are not followed blindly: the earliest available operation is always chosen except when there are two or more such operations, in which case the rule determines which is given preference.

The two solution representations require different pheromone models. The models chosen have been found to produce the best performance for their respective solution

representations [10]. For  $\mathcal{MMAS}_{perm}$ , a pheromone value, denoted  $\tau(o_i, o_j)$ ,<sup>1</sup> exists for each directed pair of operations that use the same machine, and represents the learned utility of operation  $o_i$  preceding operation  $o_j$  [2]. There may be several such precedence relations affected by the selection of a single operation. During solution construction, the set of unscheduled operations that require the same machine as a candidate operation  $o$  is denoted by  $O_o^{rel}$ . Blum and Sampels [2] recommend taking the minimum of the relevant pheromone values. Thus, at each step of solution construction, the probability of selecting an operation  $o$  to add to the partial permutation  $p$  is given by

$$P(o, p) = \begin{cases} \frac{\min_{o_r \in O_o^{rel}} \tau(o, o_r)}{\sum_{o' \notin p} \min_{o_r \in O_{o'}^{rel}} \tau(o', o_r)} & \text{if } o \notin p \text{ and } |O_o^{rel}| > 0 \\ 1 & \text{if } o \notin p \text{ and } |O_o^{rel}| = 0 \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

Note that the second branch is required so that the last operation on each machine is scheduled immediately, as there is no meaningful pheromone value that can be used.

For  $\mathcal{MMAS}_{rules}$ , a pheromone value  $\tau(M_k, d)$  is associated with each combination of machine and dispatching rule  $(M_k, d) \in M \times D$ , where  $M$  is the set of machines. At each step of solution construction, a machine is assigned a dispatching rule. Although the order in which assignments are made is significant in problems where certain items may only be assigned a limited number of times (e.g., in the generalised assignment problem [11]), here there is no limit to the number of times a rule can be used, so the assignment order is immaterial. This was confirmed during initial testing. The probability of assigning a dispatching rule  $d \in D$  to machine  $M_k$  is given by

$$P(M_k, d) = \frac{\tau(M_k, d)}{\sum_{d' \in D \setminus \{d\}} \tau(M_k, d')}. \quad (8)$$

Pheromone values are updated the same way in both algorithms, with each value  $\tau$  (corresponding to some value from either model) updated according to

$$\tau \leftarrow (\rho - 1)\tau + \rho \cdot \Delta\tau \quad (9)$$

where  $\rho$  is the pheromone evaporation rate and  $\Delta\tau$  is the amount of reinforcement given to a particular pheromone value determined by

$$\Delta\tau = \begin{cases} F(s) & \text{if } \tau \text{ is part of iteration best solution} \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

where  $F(s)$  is the objective value of the solution  $s$ . Pheromone values are bounded by  $[\tau_{min}, \tau_{max}]$ , the values of which are given in the next section.

<sup>1</sup> $\tau$  is historically used in ACO due to the pheromone model's inspiration in ant *trail* pheromones.

Table 1:  $\mathcal{MMAS}$  parameters

Parameter	Value
number of ants	10
iterations	3000
$\rho$	0.1
$\tau_{max}$	1
$\tau_{min}$ in $\mathcal{MMAS}_{rules}$	$1 \times 10^{-3}$
$\tau_{min}$ in $\mathcal{MMAS}_{perm}$	$1 \times 10^{-4}$

## 5 Computational Results

The performance of the algorithms was compared on one month's data collected from Sherwood Press (the March set used by Fayad and Petrovic [7]). The resulting JSP instance consists of 549 operations partitioned into 159 jobs.

The algorithms were implemented in the C language and executed under Linux on a 2.6GHz Pentium 4 with 512Mb of RAM. The  $\mathcal{MMAS}$  control parameters are summarised in Table 1. Note that the values of  $\tau_{min}$  and  $\tau_{max}$  were chosen to approximate those suggested by Stützle and Hoos [15] based on the size of the solution representation and pheromone update.

Both algorithms were executed with each combination of parameter values for evaluating solutions: *poss* and *area* tardiness measures,  $\lambda \in \{0.3, 0.7\}$ , and *average* and *min* aggregation operators. Each combination was run across 10 random seeds.

### 5.1 Solution quality compared within alternative measures

The results revealed that when using the *min* aggregation operator,  $\mathcal{MMAS}_{perm}$  is unable to find a solution with a non-zero objective value. This is because the algorithm, in the absence of any heuristic bias, searches randomly until a subset of pheromone values is updated. Further testing confirmed that a random search of permutations is unlikely to produce solutions with  $S_{NT} > 0$ . Consequently, a second version of the algorithm, named  $\mathcal{MMAS}_{perm}^{min}$ , was developed in which the pheromone update was modified such that, if all solutions in an iteration have an objective value of zero, the best solution in terms of  $S_{AT}$  is used to update pheromone values using the *average* aggregation operator. A similar modification was not necessary for  $\mathcal{MMAS}_{rules}$  as random assignments of dispatching rules to machines typically produced solutions with  $S_{NT} > 0$ .

Tables 2 and 3 summarise the results for  $\lambda = 0.3$  and  $\lambda = 0.7$  respectively, split according to the algorithm, tardiness measure and aggregation operator used. Included in the table are results for the genetic algorithm (GA) developed by Fayad and Petrovic [7], denoted  $GA_{rules}$ , which represents solutions in the same manner as  $\mathcal{MMAS}_{rules}$ .

It is evident in both tables that  $\mathcal{MMAS}_{perm}^{min}$  is much more successful than its original

Table 2: Algorithm performance across solution evaluation measures (with  $\lambda = 0.3$ ). The best result for each measure is given with the mean value in parentheses. Bold items are best within each solution quality measure

Algorithm	$F$	$S_{AT}$	$S_{NT}$	$C_{NT}$
Using <i>poss</i> and <i>average</i>				
$\mathcal{MMAS}_{perm}$	0.81 (0.74)	0.91 (0.90)	0.71 (0.58)	7 (10)
$\mathcal{MMAS}_{rules}$	<b>0.76 (0.76)</b>	<b>0.93 (0.93)</b>	0.58 (0.58)	10 ( <b>10</b> )
$\text{GA}_{rules}$	<b>0.77 (0.76)</b>	<b>0.93 (0.92)</b>	<b>0.62 (0.59)</b>	<b>9 (10)</b>
Using <i>poss</i> and <i>min</i>				
$\mathcal{MMAS}_{perm}$	0	0.72 (0.71)	0	35 (38.9)
$\mathcal{MMAS}_{perm}^{min}$	0.67 (0.57)	0.86 (0.83)	0.67 (0.57)	8 (10.2)
$\mathcal{MMAS}_{rules}$	0.58 ( <b>0.58</b> )	<b>0.93 (0.92)</b>	0.58 ( <b>0.58</b> )	10 ( <b>10</b> )
$\text{GA}_{rules}$	<b>0.62 (0.57)</b>	0.92 ( <b>0.92</b> )	<b>0.62 (0.57)</b>	<b>9 (10)</b>
Using <i>area</i> and <i>average</i>				
$\mathcal{MMAS}_{perm}$	0.74 (0.68)	0.90 (0.89)	0.58 (0.47)	10 (12.7)
$\mathcal{MMAS}_{rules}$	<b>0.75 (0.75)</b>	<b>0.93 (0.93)</b>	<b>0.58 (0.58)</b>	<b>10 (10)</b>
$\text{GA}_{rules}$	0.73 (0.69)	0.92 (0.9)	0.54 (0.49)	11 (12)
Using <i>area</i> and <i>min</i>				
$\mathcal{MMAS}_{perm}$	0	0.70 (0.69)	0	42 (43.9)
$\mathcal{MMAS}_{perm}^{min}$	0.54 (0.50)	0.85 (0.84)	0.54 (0.50)	11 (12)
$\mathcal{MMAS}_{rules}$	<b>0.58 (0.58)</b>	<b>0.93 (0.92)</b>	<b>0.58 (0.58)</b>	<b>10 (10)</b>
$\text{GA}_{rules}$	0.54 (0.47)	0.91 (0.9)	0.54 (0.47)	11 (13)

form when using the *min* aggregation operator. Further investigation revealed that it required the use of the *average* aggregation operator in 6–33% of iterations, with the lower value being with  $\lambda = 0.3$  and *poss*, and the upper value being with  $\lambda = 0.7$  and *area*.

When using  $\lambda = 0.3$  and the *poss* aggregation operator  $\mathcal{MMAS}_{perm}$  performs better than  $\mathcal{MMAS}_{rules}$  in terms of the number of tardy jobs. However, when using the typically more pessimistic *area* measure, its performance is slightly worse. Both algorithms compare favourably with  $\text{GA}_{rules}$ . When  $\lambda = 0.7$ ,  $\mathcal{MMAS}_{rules}$  consistently outperforms  $\mathcal{MMAS}_{perm}$ .  $\mathcal{MMAS}_{rules}$  also performs better than  $\text{GA}_{rules}$ .

## 5.2 Solution quality compared between values of $\lambda$

Examination of the satisfaction grades of solutions produced by the algorithms using higher values of  $\lambda$  (e.g., Table 3, with  $\lambda = 0.7$ ) may appear to indicate that they are of poorer quality than those obtained at lower values of  $\lambda$  (e.g., Table 2, with  $\lambda = 0.3$ ). To investigate this further, solutions produced at the two values of  $\lambda$  were compared by re-evaluating them using the same tardiness measure and aggregation operator as used when they were produced, but with the other value of  $\lambda$ . This revealed that, for  $\mathcal{MMAS}_{perm}$ ,

Table 3: Algorithm performance across solution evaluation measures (with  $\lambda = 0.7$ ). The best result for each measure is given with the mean value in parentheses. Bold items are best within each solution quality measure

Algorithm	$F$	$S_{AT}$	$S_{NT}$	$C_{NT}$
Using <i>poss</i> and <i>average</i>				
$MMAS_{perm}$	0.69 (0.62)	0.91 (0.91)	0.46 (0.34)	13 (15.9)
$MMAS_{rules}$	<b>0.73 (0.73)</b>	<b>0.93 (0.93)</b>	<b>0.54 (0.53)</b>	<b>11 (11.2)</b>
$GA_{rules}$	0.69 (0.66)	0.92 (0.91)	0.46 (0.41)	12 (13)
Using <i>poss</i> and <i>min</i>				
$MMAS_{perm}$	0	0.72 (0.71)	0	48 (51.2)
$MMAS_{perm}^{min}$	0.42 (0.35)	0.89 (0.88)	0.42 (0.35)	14 (15.5)
$MMAS_{rules}$	<b>0.54 (0.53)</b>	<b>0.93 (0.93)</b>	<b>0.54 (0.53)</b>	<b>11 (11.3)</b>
$GA_{rules}$	0.46 (0.34)	0.91 (0.90)	0.46 (0.34)	12 (15)
Using <i>area</i> and <i>average</i>				
$MMAS_{perm}$	0.62 (0.59)	0.90 (0.90)	0.33 (0.28)	16 (17.3)
$MMAS_{rules}$	<b>0.71 (0.70)</b>	<b>0.93 (0.93)</b>	<b>0.50 (0.48)</b>	<b>12 (12.5)</b>
$GA_{rules}$	0.64 (0.62)	0.91 (0.91)	0.37 (0.33)	14 (15)
Using <i>area</i> and <i>min</i>				
$MMAS_{perm}$	0	0.70 (0.69)	0	49 (52.1)
$MMAS_{perm}^{min}$	0.42 (0.32)	0.88 (0.87)	0.42 (0.32)	14 (16.4)
$MMAS_{rules}$	<b>0.50 (0.48)</b>	<b>0.93 (0.92)</b>	<b>0.50 (0.48)</b>	<b>12 (12.5)</b>
$GA_{rules}$	0.37 (0.26)	0.90 (0.89)	0.37 (0.26)	14 (17)

solutions produced using  $\lambda = 0.3$  have twice as many or more tardy jobs when re-evaluated using  $\lambda = 0.7$ , while for  $MMAS_{rules}$ , solutions produced using  $\lambda = 0.3$  typically have 1–5 more tardy jobs when re-evaluated. The same applies for results generated using  $GA_{rules}$ .

Thus, the differences observed between the original satisfaction grades of solutions produced using different values of  $\lambda$  are not an indication of better or poorer results; the setting of  $\lambda$  is merely an indication of the level of tolerance the decision maker is willing to show. The results of re-evaluating solutions show that each algorithm will find those solutions which are “good enough” according to the chosen value of  $\lambda$ , and without the pressure of a higher value will not search for solutions that would meet stricter requirements. Therefore, in the case of a scheduling problem of sufficient complexity, the decision maker might choose a lower value of  $\lambda$  to help increase the region of the search space that contains “good” solutions. On the other hand, if a problem is not considered complex enough by the scheduler, he/she might decide to increase the value of  $\lambda$ , thereby increasing the difficulty of finding better solutions.

Table 4: Mean CPU time in seconds used in total and until best solution found. Results are divided based on the tardiness measure (*poss* or *area*) and aggregation operator (*average* and *min*) used. Note that  $\mathcal{MMAS}_{perm}^{min}$  behaves identically to  $\mathcal{MMAS}_{perm}$  when using *average*

Algorithm	total				best solution			
	<i>poss</i>		<i>area</i>		<i>poss</i>		<i>area</i>	
	<i>average</i>	<i>min</i>	<i>average</i>	<i>min</i>	<i>average</i>	<i>min</i>	<i>average</i>	<i>min</i>
$\mathcal{MMAS}_{perm}^{min}$	1466.3	1428.6	1509.5	1504.8	1303.3	1025.8	1432.8	396.1
$\mathcal{MMAS}_{rules}$	91.0	91.1	108.8	108.3	0.7	0.5	1.2	0.3

### 5.3 CPU time

Table 4 summarises the mean computation time required to complete 3000 iterations and until the best solution was found, broken down according to the solution evaluation measure used. The most notable feature of these results is the order of magnitude difference between the two ACO algorithms. This result is to be expected given the respective number of components each must consider at each constructive step;  $\mathcal{MMAS}_{perm}$  considers approximately 40 operations on average, while  $\mathcal{MMAS}_{rules}$  considers only four. Moreover,  $\mathcal{MMAS}_{rules}$  finds its best solutions very early in each run (often within 1 second) while  $\mathcal{MMAS}_{perm}$  does not converge until quite late.

A less significant difference was observed between the *poss* and *area* measures, with the latter requiring longer computation time. This is most noticeable in  $\mathcal{MMAS}_{rules}$ , where solution evaluation represents a larger proportion of computation time than it does in  $\mathcal{MMAS}_{perm}$ .

## 6 Conclusions

Typical ACO algorithms for production scheduling problems such as the JSP build solutions as permutations of the operations to be scheduled, from which actual schedules are generated deterministically. An alternative approach when the problem in question has multiple machines and various criteria upon which to judge the urgency of competing operations is to assign different dispatching rules to each machine. The chosen dispatching rules are then responsible for determining the relative processing order of operations on each machine. This paper compared both approaches on a multi-objective real-world JSP, modelled with fuzzy operation processing times and job due dates. The results show that assigning dispatching rules to machines produces higher quality solutions in far less time than building a permutation of the operations. This supports the claim that the assignment of dispatching rules restricts the search space to an area of good quality solutions.

As this study focused on a single, real-world JSP instance (albeit using a variety of solution quality measures) future work is required to determine if these results hold

for other production scheduling instances. Additionally, it is now common practice in most ACO algorithms to use a local search procedure to improve the solutions produced, something not done in this study so that differences between the two solution construction approaches could be observed. While the addition of local search to a permutation-based ACO algorithm for these problems may allow it to perform better, it is potentially more useful in the new approach, where it can explore solutions that combinations of dispatching rules would otherwise never produce.

## References

- [1] A. Bauer, B. Bullnheimer, R. F. Hartl, and C. Strauss. Minimizing total tardiness on a single machine using ant colony optimization. *Cent. Eur. J. Oper. Res.*, 8(2):125–141, 2000.
- [2] C. Blum and M. Sampels. Ant colony optimization for FOP shop scheduling: A case study on different pheromone representations. In *2002 Congress on Evolutionary Computation*, pages 1558–1563, 2002.
- [3] C. Blum and M. Sampels. An ant colony optimization algorithm for shop scheduling problems. *J. Math. Model. Algorithms*, 3(3):285–308, 2004.
- [4] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant system for job-shop scheduling. *JORBEL*, 34(1):39–53, 1994.
- [5] U. Dorndorf and E. Pesch. Evolution based learning in a job shop scheduling environment. *Comput. Oper. Res.*, 22:25–44, 1995.
- [6] D. Dubois and P. H. *Possibility Theory: An Approach to Computerized Processing of Uncertainty*. Kluwer Academic, New York, 1988.
- [7] C. Fayad and S. Petrovic. A fuzzy genetic algorithm for real-world job shop scheduling. In M. Ali and F. Esposito, editors, *18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2005)*, volume 3533 of *Lecture Notes in Artificial Intelligence*, pages 524–533, Bari, Italy, 2005. Springer-Verlag.
- [8] T. Itoh and H. Ishii. Fuzzy due-date scheduling problem with fuzzy processing time. *Int. Trans. Oper. Res.*, 6:639–647, 1999.
- [9] G. Klir and T. Folger. *Fuzzy Sets, Uncertainty and Information*. Prentice Hall, New Jersey, 1988.
- [10] E. J. Montgomery. *Solution Biases and Pheromone Representation Selection in Ant Colony Optimisation*. PhD thesis, Bond University, 2005.

- [11] J. Montgomery, M. Randall, and T. Hendtlass. Search bias in constructive metaheuristics and implications for ant colony optimisation. In M. Dorigo et al., editors, *4th Int'l Workshop on Ant Colony Optimization and Swarm Intelligence, ANTS 2004*, volume 3172 of *Lecture Notes in Computer Science*, pages 390–397, Brussels, Belgium, 2004. Springer-Verlag.
- [12] J. Montgomery, M. Randall, and T. Hendtlass. Structural advantages for ant colony optimisation inherent in permutation scheduling problems. In M. Ali and F. Esposito, editors, *18th Int'l Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2005)*, volume 3533 of *Lecture Notes in Artificial Intelligence*, pages 218–228, Bari, Italy, 2005. Springer-Verlag.
- [13] M. Sakawa and R. Kubota. Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy due date through genetic algorithms. *Eur. J. Oper. Res.*, 120(2):393–407, 2000.
- [14] T. Stützle. An ant approach to the flow shop problem. In *6th European Congress on Intelligent Techniques & Soft Computing (EUFIT '98)*, pages 1560–1564, Aachen, Germany, 1998. Verlag Mainz.
- [15] T. Stützle and H. Hoos.  $\mathcal{MAX} - \mathcal{MIN}$  ant system. *Future Gen. Comp. Sys.*, 16:889–914, 2000.
- [16] S. van der Zwaan and C. Marques. Ant colony optimisation for job shop scheduling. In *3rd Workshop on Genetic Algorithms and Artificial Life (GAAL 99)*, 1999.