Solution Representation for Job Shop Scheduling Problems in Ant Colony Optimisation

James Montgomery*[1], Carole Fayad[2], and Sanja Petrovic[2]

[1] Faculty of Information & Communication Technologies, Swinburne University of Technology, Melbourne, Australia
jmontgomery@ict.swin.edu.au
[2] School of Computer Science & IT, University of Nottingham, Nottingham, UK
{cxf,sxp}@cs.nott.ac.uk

Abstract. Production scheduling problems such as the job shop consist of a collection of operations (grouped into jobs) that must be scheduled for processing on different machines. Typical ant colony optimisation applications for these problems generate solutions 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 a dispatching rule, which heuristically determines the order of operations on 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 in which, during successive iterations of solution construction, 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.

* Corresponding author

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, 2]). 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* [1]. 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 [3]. 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 analysis of their empirical performance in Section 5. Section 6 describes the implications of the results for the future application of ACO to such problems. An extended version of this paper, including more extensive empirical analyses, is presented in [9].

2 Job Shop Scheduling and Solution Construction

The JSP examined in this study consists of a set of n jobs $J_1, \ldots, J_n$, with associated release dates $r_1, \ldots, r_n$ and due dates $d_1, \ldots, d_n$. 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 $M_1, \ldots, M_m$. Only one operation of 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 criteria have to be minimised simultaneously, the average tardiness of jobs C_{AT} and the number of tardy jobs C_{NT}, calculated as follows:

$$C_{AT} = \frac{1}{n} \sum_{j=1}^{n} T_j \tag{1}$$

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

$$C_{NT} = \sum_{j=1}^{n} u_j \tag{2}$$

where $u_j = 1$ if $T_j > 0$, 0 otherwise.

It is common in ACO applications for the JSP and related scheduling problems to generate a permutation of the operations, which implicitly determines the relative processing order of operations on each machine. 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*. A deterministic algorithm transforms the relative processing order into an actual schedule.

Different solution construction approaches 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 [10]. 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 [3]. 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 has been provided by a printing company, Sherwood Press, in Nottingham, United Kingdom [5]. 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.

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 these uncertain values. A triangular membership function $\mu_{\tilde{p}_{ij}}(t) = (p^1_{ij}, p^2_{ij}, p^3_{ij})$ is used to model the fuzzy processing time $\tilde{p}_{ij}$ of job J_j on machine M_i, $i = 1, \ldots, m$, $j = 1, \ldots, n$, where p^1_{ij} and p^3_{ij} are lower and upper bounds of the processing time, while p^2_{ij} is the so-called modal point [7]. An example of fuzzy processing time is shown in Fig. 1(a). A trapezoidal fuzzy set (d^1_j, d^2_j) is used to model the due date $\tilde{d}_j$ of each job, where d^1_j is the crisp due date and the upper bound d^2_j of the trapezoid exceeds d^1_j by 10%, following the policy of the company. An example of a fuzzy due date is given in Fig. 1(b).

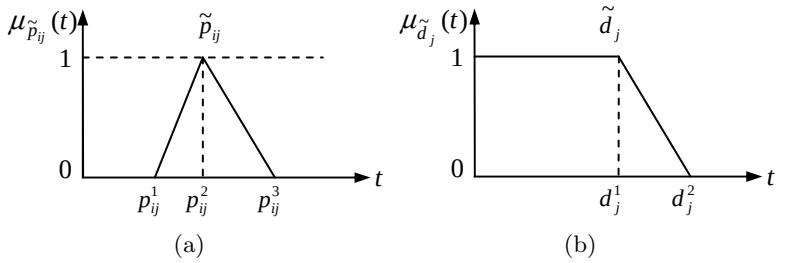

Fig. 1. Fuzzy (a) processing time and (b) due date

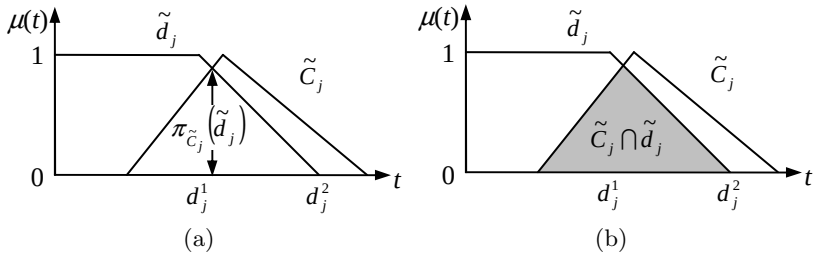

Fig. 2. Satisfaction grade of tardiness using (a) possibility measure and (b) area of intersection

The objective function takes into account both the average tardiness of jobs and the number of tardy jobs. As these are measured in different units they are mapped onto satisfaction grades in the range $[0, 1]$, which are then combined in an overall satisfaction grade. Two approaches used to measure tardiness in [5] are investigated:

1. The possibility measure $\pi_{\tilde{C}_j}(\tilde{d}_j)$, used by Itoh and Ishii [6] to handle tardy jobs in a JSP, 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 $\tilde{d}_j$ [6] (illustrated in Fig. 2(a)):

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

   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 measure (denoted *area* hereafter), introduced by Sakawa and Kubota [11], measures the proportion of $\tilde{C}_j$ that is completed by the due date $\tilde{d}_j$ (illustrated in Fig. 2(b)):

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

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}:

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

2. To maximise the satisfaction grade of *number of tardy jobs* S_{NT}: A parameter λ is introduced such that a job J_j, $j = 1, \ldots, n$, 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 calculated as:

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

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

Two different aggregation operators, average and minimum (denoted *average* and *min* hereafter), were investigated for combining the satisfaction grades of the objectives.

4 ACO for a Fuzzy JSP

Two ACO algorithms were developed based on the $\mathcal{MAX} - \mathcal{MIN}$ Ant System ($\mathcal{MM}$AS), which has been found to work well in practice [12]. The first of these, denoted $\mathcal{MM}\text{AS}_{perm}$, constructs solutions as permutations of the operations, while the second, denoted $\mathcal{MM}\text{AS}_{rules}$, 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.

The two solution representations require different pheromone models. The models chosen have been found to produce the best performance for their respective solution representations [8]. For $\mathcal{MM}\text{AS}_{perm}$, a pheromone value, denoted $\tau(o_i, o_j)$, 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 [1]. At each step of 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 [1] take the minimum of the relevant pheromone values. Thus, the probability of selecting an available operation o to add to the partial permutation p is given by

$$P(o, p) = \frac{\min_{o_r \in O_o^{rel}} \tau(o, o_r)}{\sum_{o' \notin p} \min_{o_r \in O_{o'}^{rel}} \tau(o', o_r)}. \tag{7}$$

The last operation on each machine is scheduled as soon as it becomes available.

For $\mathcal{MM}\text{AS}_{rules}$, 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. The probability of assigning a dispatching rule $d \in D$ to machine M_k is given by

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

In both algorithms, after each iteration all pheromone values are reduced in proportion to $(\rho - 1)$ while those for the iteration best solution s^{ib} are increased by $\rho \cdot F(s^{ib})$, where ρ is the pheromone evaporation rate and F is the overall satisfaction grade of s^{ib} (given by either the *average* or *min* aggregation operator).

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 [5]). 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 $\mathcal{MM}$AS control parameters used were: 10 ants per iteration; 3000 iterations; $\rho = 0.1$; $\tau_{max} = 1$; $\tau_{min} = 1 \times 10^{-3}$ in $\mathcal{MM}\text{AS}_{rules}$ and $\tau_{min} = 1 \times 10^{-4}$ in $\mathcal{MM}\text{AS}_{perm}$. The values of τ_{min} and τ_{max} were chosen to approximate those suggested by Stützle and Hoos [12] based on the size of the solution representation and pheromone update.

Both algorithms were executed with different combinations of parameter values for solution evaluation: *poss* and *area* tardiness measures, and *average* and *min* aggregation operators. The value of λ was fixed at 0.7. Each combination was run with 10 different random seeds.

5.1 Solution quality

The results revealed that when using the *min* aggregation operator, $\mathcal{MM}\text{AS}_{perm}$ is unable to find a solution with a non-zero objective value. This is because the algorithm, facing a large number of solutions with $S_{NT} = 0$, 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 $S_{NT} > 0$. A second version of the algorithm, named $\mathcal{MM}\text{AS}^{min}_{perm}$, 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. Such a modification was not necessary for $\mathcal{MM}\text{AS}_{rules}$ as random assignments of dispatching rules to machines typically produced solutions with $S_{NT} > 0$.

Table 1 summarises the satisfaction grades of tardiness measures according to the aggregation operator used for each algorithm. It is evident that $\mathcal{MM}\text{AS}^{min}_{perm}$ is much more successful than its original form when using the *min* aggregation

Table 1. Performance of the algorithms. 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 *poss* and *average*				
$\mathcal{MM}\text{AS}_{perm}$	0.69 (0.62)	0.91 (0.91)	0.46 (0.34)	13 (15.9)
$\mathcal{MM}\text{AS}_{rules}$	**0.73 (0.73)**	**0.93 (0.93)**	**0.54 (0.53)**	**11 (11.2)**
Using *poss* and *min*				
$\mathcal{MM}\text{AS}_{perm}$	0	0.72 (0.71)	0	48 (51.2)
$\mathcal{MM}\text{AS}^{min}_{perm}$	0.42 (0.35)	0.89 (0.88)	0.42 (0.35)	14 (15.5)
$\mathcal{MM}\text{AS}_{rules}$	**0.54 (0.53)**	**0.93 (0.93)**	**0.54 (0.53)**	**11 (11.3)**
Using *area* and *average*				
$\mathcal{MM}\text{AS}_{perm}$	0.62 (0.59)	0.90 (0.90)	0.33 (0.28)	16 (17.3)
$\mathcal{MM}\text{AS}_{rules}$	**0.71 (0.70)**	**0.93 (0.93)**	**0.50 (0.48)**	**12 (12.5)**
Using *area* and *min*				
$\mathcal{MM}\text{AS}_{perm}$	0	0.70 (0.69)	0	49 (52.1)
$\mathcal{MM}\text{AS}^{min}_{perm}$	0.42 (0.32)	0.88 (0.87)	0.42 (0.32)	14 (16.4)
$\mathcal{MM}\text{AS}_{rules}$	**0.50 (0.48)**	**0.93 (0.92)**	**0.50 (0.48)**	**12 (12.5)**

operator. Further investigation revealed that it required the use of the *average* aggregation operator in up to 33% of iterations. Across solution evaluation measures, $\mathcal{MM}\text{AS}_{rules}$ clearly outperforms $\mathcal{MM}\text{AS}_{perm}$.

5.2 CPU time

An order of magnitude difference was observed between the CPU time of the two algorithms, with $\mathcal{MM}\text{AS}_{perm}$ taking more than 1400 seconds compared to approximately 100 seconds for $\mathcal{MM}\text{AS}_{rules}$. This is to be expected given the respective number of components each must consider at each constructive step; $\mathcal{MM}\text{AS}_{perm}$ considers approximately 40 operations on average, while $\mathcal{MM}\text{AS}_{rules}$ considers only four. Moreover, $\mathcal{MM}\text{AS}_{rules}$ finds its best solutions very early in each run (often within 1 second) while $\mathcal{MM}\text{AS}_{perm}$ does not converge until quite late.

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] C. Blum and M. Sampels. An ant colony optimization algorithm for shop scheduling problems. *J. Math. Model. Algorithms*, 3(3):285–308, 2004.

[2] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant system for job-shop scheduling. *JORBEL*, 34(1):39–53, 1994.

[3] U. Dorndorf and E. Pesch. Evolution based learning in a job shop scheduling environment. *Comput. Oper. Res.*, 22:25–44, 1995.

[4] D. Dubois and P. H. *Possibility theory: An approach to computerized processing of uncertainty*. Kluwer Academic, New York, 1988.

[5] C. Fayad and S. Petrovic. A fuzzy genetic algorithm for real-world job shop scheduling. In M. Ali and F. Esposito, editors, *Innovations in Applied Artificial Intelligence*, volume 3533 of *LNAI*, pages 524–533, 2005. Springer-Verlag.

[6] T. Itoh and H. Ishii. Fuzzy due-date scheduling problem with fuzzy processing time. *Int. Trans. Oper. Res.*, 6:639–647, 1999.

[7] G. Klir and T. Folger. *Fuzzy sets, uncertainty and information*. Prentice Hall, New Jersey, 1988.

[8] E. J. Montgomery. *Solution biases and pheromone representation selection in ant colony optimisation*. PhD thesis, Bond University, 2005.

[9] J. Montgomery, C. Fayad and S. Petrovic. *Representation of solutions to job shop scheduling problems in ant colony optimisation*. Technical Report SUTICT-TR2006.05, Faculty of Information & Communication Technologies, Swinburne University of Technology, 2006.

[10] 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, *Innovations in Applied Artificial Intelligence*, volume 3533 of *LNAI*, pages 218–228, 2005. Springer-Verlag.

[11] M. Sakawa and R. Kubota. Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms. *Eur. J. Oper. Res.*, 120(2):393–407, 2000.

[12] T. Stützle and H. Hoos. $\mathcal{MAX} - \mathcal{MIN}$ ant system. *Future Gen. Comp. Sys.*, 16:889–914, 2000.