Alternative Solution Representations for the Job Shop Scheduling Problem in Ant Colony Optimisation

James Montgomery

Complex Intelligent Systems Laboratory
Centre for Information Technology Research
Faculty of Information & Communication Technologies
Swinburne University of Technology
Melbourne, Australia
jmontgomery@ict.swin.edu.au

Abstract. Ant colony optimisation (ACO), a constructive metaheuristic inspired by the foraging behaviour of ants, has frequently been applied to shop 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. An alternative approach is to assign each machine one of a number of alternative dispatching rules to determine its individual processing order. This representation creates a substantially smaller search space biased towards good solutions. A previous study compared the two alternatives applied to a complex real-world instance and found that the new approach produced better solutions more quickly than the original. This paper considers its application to a wider set of standard benchmark job shop instances. More detailed analysis of the resultant search space reveals that, while it focuses on a smaller region of good solutions, it also excludes the optimal solution. Nevertheless, comparison of the performance of ACO algorithms using the different solution representations shows that, using this solution space, ACO can find better solutions than with the typical representation. Hence, it may offer a promising alternative for quickly generating good solutions to seed a local search procedure which can take those solutions to optimality.

Keywords: Ant colony optimisation, 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.

Shop 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., [1,2,3,4]). 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* [2].

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. Building solutions in this manner may offer an advantage by concentrating the search on heuristically good solutions. A previous study compared these two solution representations in ACO algorithms for a real-world job shop scheduling problem (JSP) with staggered release and due dates modelled using fuzzy sets [5]. Applied to that single real-world instance the alternative approach performed extremely well, finding better solutions than the list scheduler ACO in considerably less time. An open question was whether the same relative performance would be observed on other, benchmark JSP instances.

This paper examines, in greater detail than in [5], the search space produced by the alternative solution representation when applied to a number of commonly used benchmark JSP instances (Section 4). An empirical comparison is subsequently made of ACO algorithms using the typical and alternative solution construction approaches (Sections 5–6). Section 7 describes the implications of the results for the future application of ACO to such problems. A formal description of the JSP and further details of the typical solution construction approach are given first.

2 Job Shop Scheduling

The JSP examined in this study is of the $n \times m$ form, with a set of n jobs $J_1, \ldots, J_n$ and m machines $M_1, \ldots, M_m$. Each job consists of a predetermined sequence of m operations, each of which requires one of the m machines. 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 pre-empted.

Table 1. JSP instances used in this study

Instance	Best known	n	m
abz5	1234	10	10
abz6	943	10	10
abz7	656	20	15
abz8	669	20	15
abz9	679	20	15
ft10	930	10	10
ft20	1165	20	5
la21	1046	15	10
la24	935	15	10
la25	977	15	10
la27	1235	20	10
la29	1152	20	10
la38	1196	15	15
la40	1222	15	15
orb08	899	10	10
orb09	934	10	10

The objective is to schedule operations for processing on machines such that the total time to complete all jobs, the *makespan*, is minimised. The makespan of a solution s is denoted $C_{max}(s)$.

Table 1 describes the instances used in this study to compare the alternative solution representations. They are commonly used benchmarks in the ACO and wider operations research literature and are all available from the OR-Library [6].

3 Typical Solution Construction for the JSP

To generate a solution to the JSP 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., [1,2,3,4,7]). 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 $k = n \cdot m$ is the number of operations) and is certainly much larger than the space of feasible schedules [8]. This space also has a slight bias towards good solutions, which can be exploited by some pheromone models and proves disastrous for others. Another notable feature of this search space is that while all solutions can be reached, solutions (schedules) are repre-

sented by differing numbers of permutations. These issues are discussed in some detail by Montgomery, Randall and Hendtlass [8,9].

4 Search Space Created by Dispatching Rules

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. 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 search space will correspond to a subset of the space of all feasible schedules. Further, given that dispatching rules are chosen with the aim of minimising the makespan or number of tardy jobs, this is probably the case even for large sets of rules. However, if the dispatching rules individually perform well it is expected that this reduced space largely consists of good quality schedules.

Clearly, such an approach is inappropriate for single machine scheduling problems or problems in which too few criteria are available to heuristically determine the processing order of competing operations, as in either situation the search space is reduced by too great an amount. It is, however, entirely appropriate for problems with multiple machines and various criteria upon which to judge competing operations. This study examines its application to a number of common benchmark JSPs using four dispatching rules. The remainder of this section examines whether, for these instances using these four rules, the approach is appropriate.

The four rules used in this study are Earliest Starting Time (EST), Shortest Processing Time (SPT), Longest Processing Time (LPT) and Longest Remaining Processing Time (LRPT). SPT and LPT relate to an individual operation's processing time while LRPT refers to the remaining processing time of a candidate operation's containing job. EST is perhaps the simplest heuristic, choosing the operation that can start the soonest, with ties broken randomly. Note that the three other 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.

For small instances and a set of four rules it is possible to completely enumerate the set of assignment solutions.[1] This was performed for the test instances with up to 200 operations to discover the distribution of the cost of schedules described. The distributions for the larger instances were estimated by sampling 4×10^6 randomly generated solutions. Note that as the EST rule breaks ties randomly, there is some degree of error in the lower and upper bounds presented, although it is likely the distributions described here are good approximations of the true distributions. Fig. 1 presents box-plots of the distributions discovered,

[1] Although complete enumeration of the search space obviates the need for a metaheuristic, on any moderate-sized instance or as the number of rules grows it quickly becomes impractical.

expressed in terms of the relative percentage deviation (RPD) from the best known cost, defined as

$$RPD = \frac{C_{max}(s) - C_{max}(s^*)}{C_{max}(s^*)} \cdot 100 \quad (1)$$

where s is a solution and s^* is the best known solution.

The most striking feature of the distributions is that they *do not include the optimum*. Additionally, tests with a smaller number of rules found that many unique assignment solutions generate the same schedule, as was anticipated.[2] Nevertheless, it is still possible that the assignment approach does focus on a good region of the space of schedules, and thus may present a good starting point for the subsequent application of a local search algorithm. As the worst cost is not known for these instances it cannot be proved that these distributions are biased towards good solutions. However, examination of the cost distribution of schedules produced by randomly generated feasible *permutations* lends some support to that conjecture. Fig. 2 presents box-plots for the cost distributions for 4×10^6 randomly generated feasible permutations. Notably, the minima of those distributions are in most cases above the median of those for assignment solutions while the body of those distributions typically lies above the maximum of that for assignment solutions. Of course, sample distributions for the permutation approach do not represent the full space of solutions that can be represented by permutations and indeed an ACO algorithm constructing permutations can improve on the minima of those randomly generated samples (see Section 6 for such results).

Table 2 summarises the characteristics of the search spaces created by the alternative construction approaches. With respect to search space size, the space of assignments of rules to machines (for four rules) for the instances studied is hundreds of orders of magnitude smaller than the upper bound on the space of feasible permutations.

Clearly, the two alternative approaches offer a mixture of advantages and disadvantages to any heuristic that uses them. The likelihood that, across a wider range of instances, the dispatching rules approach excludes the optimal certainly impacts on its utility. However, a previous comparative study of ACO algorithms using both approaches applied to a large, complex JSP instance found that the approach outperformed an ACO algorithm that constructs permutations in terms of both solution quality and computation time [5].[3]

Nevertheless, in a practical application of the approach, a local search component is required if the schedules described by dispatching rules are to be fully optimised. Furthermore, the local search cannot operate on the assignments directly, as that space does not contain the optimum. The next section compares ACO algorithms using both solution construction approaches. To avoid the con-

[2] Determining the number of *distinct* solutions was impractical with four rules.

[3] The number of construction steps per solution in ACO for the JSP is $n \cdot m$ when constructing permutations but only m when assigning dispatching rules.

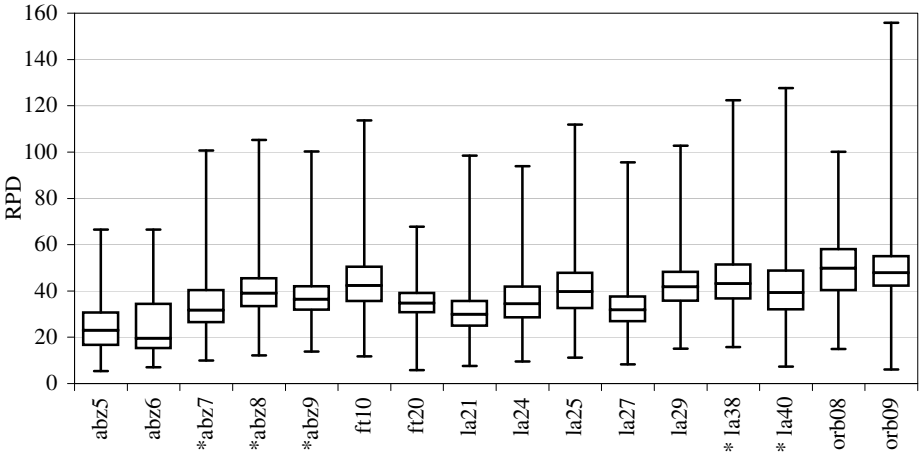

Fig. 1. Cost distributions (expressed as relative percentage deviation (RPD) from the best known) for solutions obtainable using EST, SPT, LPT and LRPT rules. Distributions marked with * are approximations based on 4×10^6 sampled solutions

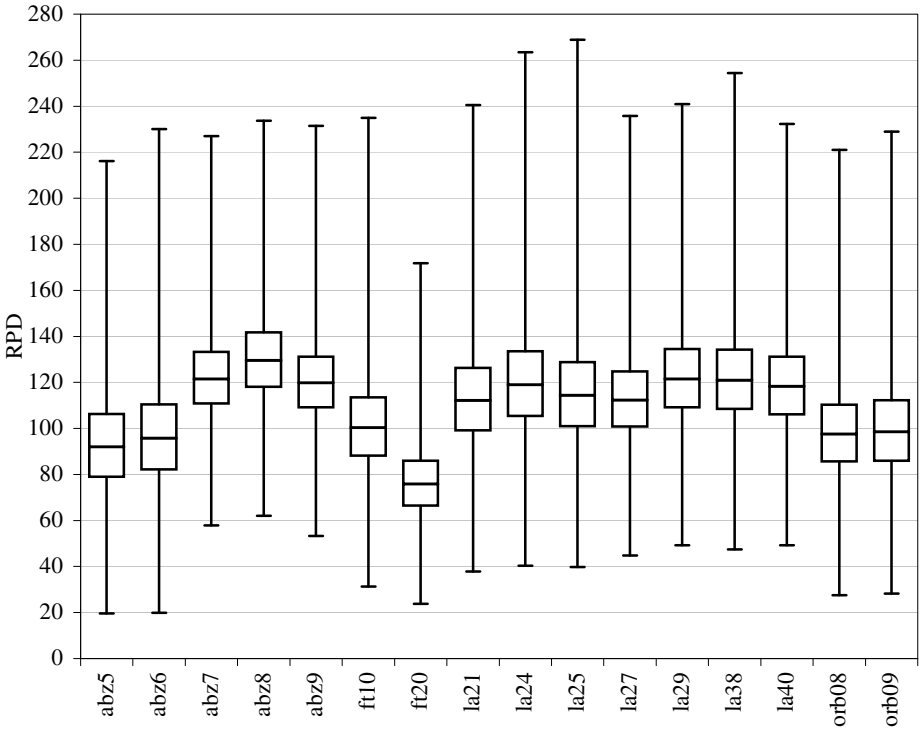

Fig. 2. Cost distributions (expressed as relative percentage deviation (RPD) from the best known) for 4×10^6 randomly generated permutation solutions. While the search space includes the optimum, it is unlikely to be found using random search

Table 2. Comparison of permutation and dispatching rules search spaces. k is number of operations, D is set of dispatching rules, m is number of machines. Typically $|D| < m < k$. Notes: $^{\#}$ this result is true for the rules and instances used in this study

	Solution approach	
Search space feature	permutation	dispatching rules
Size	$\ll O(k!)$	$O(\|D\|^m)$
Includes optimal solution	yes	no$^{\#}$
Solution representation bias	yes	yes
Biased towards good solutions	yes, but not practically so [8,9]	yes

founding effects of an integrated local search procedure, local search has not been included in the algorithms compared in this paper.

5 Comparison ACO Algorithm Details

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 [10]. The first of these, denoted $\mathcal{MM}$AS-P, constructs solutions as permutations of the operations, while the second, denoted $\mathcal{MM}$AS-R, assigns dispatching rules to machines. The set of dispatching rules D consists of the four rules described in Section 4. Although local search is considered an integral part of state-of-the-art ACO applications [11,12], in order to observe the differences between the two approaches, local search is not incorporated into either.

The two solution representations require different pheromone models. The models chosen have been found to produce the best performance for their respective solution representations [9]. For $\mathcal{MM}$AS-P, a pheromone value, denoted $\tau(o_i, o_j)$,[4] 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 [13]. 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 [13] recommend taking the minimum of the relevant pheromone values. This approach, like many ACO algorithms, benefits from the incorporation of heuristic information in the construction decision, by convention denoted η. While any dispatching rule could conceivably be used for this purpose, Blum and Sampels [2] have found that the EST rule works well on a range of instances. Accordingly,

$$\eta(o) = \frac{1}{t_{es}(o, s^p)} \tag{2}$$

[4] τ is historically used in ACO due to the pheromone model's inspiration in ant *trail* pheromones.

where $t_{es}(o, s^p)$ is the earliest time operation o could start given the current partial solution s^p. Combining this measure with the pheromone information, at each step of solution construction, the probability of selecting an operation o to add to the partial permutation p is given by

$$P(o,p) = \begin{cases} \dfrac{\left(\min_{o_r \in O_o^{rel}} \tau(o, o_r)\right) \cdot \eta(o)}{\sum_{o' \notin p} \left(\min_{o_r \in O_{o'}^{rel}} \tau(o', o_r)\right) \cdot \eta(o')} & \text{if } o \notin p \text{ and } |O_o^{rel}| > 0 \\ 1 & \text{if } o \notin p \text{ and } |O_o^{rel}| = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

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 $\mathcal{MM}$AS-R, 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 [14]), here there is no limit to the number of times a rule can be used, so the assignment order is immaterial [5]. The probability of assigning a dispatching rule $d \in D$ to machine M_k is given by

$$P(M_k, d) = \frac{\tau(M_k, d)}{\sum_{d' \in D \setminus \{d\}} \tau(M_k, d')}. \tag{4}$$

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

$$\tau \leftarrow (\rho - 1)\tau + \rho \cdot \Delta\tau \tag{5}$$

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

$$\Delta\tau = \begin{cases} \dfrac{1}{C_{max}(s)} & \text{if } \tau \text{ is part of iteration best solution} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $C_{max}(s)$ is the makespan of the solution s. Pheromone values are bounded by $[\tau_{min}, \tau_{max}]$, the values of which are controlled using the value of the current best solution and size of the pheromone update in accordance with the rules defined by Stützle and Hoos [10].

6 Computational Results

The performance of the algorithms was compared on the benchmark instances described in Table 1. The algorithms were implemented in the C language and

Table 3. Minimum, median, maximum and interquartile range (IQR) of solution cost (in RPD) for $\mathcal{MM}$AS-P and $\mathcal{MM}$AS-R. The last column shows the *estimated* best possible RPD in the space of dispatching rules used in this study. Bold values indicate the smaller value for that measure and that instance between $\mathcal{MM}$AS-P and $\mathcal{MM}$AS-R. *M-W test* indicates the direction of the difference between the distributions of RPD scores if the difference is statistically significant for $\alpha \leq 0.05$

	$\mathcal{MM}$AS-P				M-W	$\mathcal{MM}$AS-R				lower
Instance	min	med	max	IQR	test	min	med	max	IQR	bound
abz5	**2.6**	**4.3**	6.3	1.2	<	5.3	5.3	**5.3**	**0.0**	5.3
abz6	**0.4**	**2.4**	**4.0**	1.8	<	7.1	7.1	7.3	**0.0**	7.1
ft10	**8.6**	**13.5**	**14.8**	2.5	<	11.7	15.6	15.6	**0.4**	11.7
ft20	12.8	17.5	24.8	8.3	>	**5.9**	**7.1**	**8.2**	**0.6**	5.8
orb08	**9.7**	19.6	21.9	6.5		14.9	**18.0**	**18.4**	**0.3**	14.9
orb09	**1.5**	**6.3**	**9.3**	3.6	<	6.1	9.2	12.0	**3.5**	6.1
la21	**7.0**	**9.2**	11.6	2.3		7.8	9.3	**10.7**	**1.6**	7.6
la24	**7.6**	10.0	12.7	2.8		9.5	**9.5**	**9.5**	**0.0**	9.5
la25	**8.2**	**12.3**	13.8	4.5		12.5	13.1	**13.3**	**0.4**	11.2
la27	11.3	14.0	18.0	3.4	>	**8.3**	**10.1**	**10.9**	**1.5**	8.3
la29	**15.5**	16.8	20.0	1.0	>	15.6	**16.1**	**16.2**	**0.4**	15.1
la38	**12.7**	**14.7**	**17.1**	**2.3**	<	16.6	18.4	19.7	2.6	15.7
la40	**6.5**	**8.1**	**10.1**	**1.7**	<	7.4	9.0	10.4	2.0	7.4
abz7	12.2	14.1	19.1	2.7	>	**10.1**	**10.9**	**11.7**	**0.9**	9.9
abz8	14.1	16.2	19.1	3.4	>	**12.1**	**12.6**	**15.2**	**1.4**	12.1
abz9	18.3	20.3	27.5	2.0	>	**13.8**	**15.5**	**16.9**	**1.9**	13.8

executed under Linux on a 3.2GHz Xeon processor. Each run used 100 ants and executed 500 iterations of solution construction. The $\mathcal{MM}$AS pheromone decay control parameter $\rho = 0.1$. These settings were found to produce the best performance in both algorithms. Each algorithm and instance combination was executed across 10 random seeds.

6.1 Makespan

Table 3 describes, for each instance, the distributions of best solution cost (expressed in RPD) for $\mathcal{MM}$AS-P and $\mathcal{MM}$AS-R found across multiple runs of each algorithm. The instances appear in non-decreasing order of number of operations. Bold values indicate the smaller result within that instance and measure (min, median, max or interquartile range (IQR)) between the alternative algorithms. Although smaller values for IQR are not necessarily an indicator of better performance, they do indicate more consistent performance. To give an indication of the performance of $\mathcal{MM}$AS-R in exploring the space of assignments of dispatching rules, the last column gives the estimated lower bound on solution cost for each instance. Mann-Whitney tests were used to compare the distributions within each instance. Where those tests indicated a statistically

Table 4. Median CPU time in seconds used to complete 500 iterations and until best solution found, and iteration when best solution found, for $\mathcal{MM}$AS-P and $\mathcal{MM}$AS-R

Instance	Mean CPU time (s)				Iteration when best found	
	total		best solution			
	$\mathcal{MM}$AS-P	$\mathcal{MM}$AS-R	$\mathcal{MM}$AS-P	$\mathcal{MM}$AS-R	$\mathcal{MM}$AS-P	$\mathcal{MM}$AS-R
abz5	23.7	3.1	2.7	0.1	58	11
abz6	23.7	2.9	2.5	0.7	52	124
ft10	23.5	2.9	5.3	0.1	113	17
ft20	46.4	3.7	17.5	1.7	189	238
orb08	22.9	2.9	4.6	0.6	100	102
orb09	23.6	3.1	4.7	2.0	100	324
la21	63.6	5.3	28.6	0.6	225	53
la24	63.3	5.2	19.6	0.3	155	31
la25	63.6	5.0	19.0	0.3	150	25
la27	130.6	8.1	58.9	1.9	226	114
la29	130.8	7.7	54.0	0.5	206	30
la38	118.3	8.4	32.2	0.6	136	35
la40	117.9	8.8	40.0	0.9	170	49
abz7	247.2	12.8	71.1	1.5	144	59
abz8	247.6	12.9	71.1	2.0	144	78
abz9	246.5	12.8	118.1	1.2	240	49

significant result (at or below the 5% level), the central column indicates the direction of the difference (i.e., $<$ means $\mathcal{MM}$AS-P outperformed $\mathcal{MM}$AS-R while $>$ indicates the opposite).

Based on these results, neither algorithm is clearly better than the other across all instances studied. The apparently aberrant statistical result for the `la29` instance is because, even though $\mathcal{MM}$AS-P found a better solution on one of its runs, $\mathcal{MM}$AS-R produced solutions of similar cost more consistently. Considering just those instances where statistically significant differences were found there is an apparent trend showing better performance from $\mathcal{MM}$AS-R on larger instances, although this may be an effect of the actual instances used. In several cases $\mathcal{MM}$AS-R was able to locate assignment solutions at the estimated (for large instances) lower bound for the space it searches. Notably, it appears that, in the absence of a local search procedure, the traditional construction approach is unable to find the optimal solution even though it exists in the space of solutions it searches. Thus both algorithms require local search in order to find optimal solutions.

6.2 CPU time

Table 4 summarises the median computation time required to complete 500 iterations and until the best solution was found, as well as the iteration in which the best solution was found. As predicted, $\mathcal{MM}$AS-R is significantly faster than

$\mathcal{MM}$AS-P due to the difference in the number of required construction steps each iteration—as the number of operations grows the ratio between $\mathcal{MM}$AS-P's and $\mathcal{MM}$AS-R's runtimes approaches the number of jobs n. $\mathcal{MM}$AS-R also frequently locates its best solution after fewer iterations than $\mathcal{MM}$AS-P. The faster execution of $\mathcal{MM}$AS-R commends it as a good alternative for integration with a potentially computationally intensive local search, and would also allow for a greater number of separate runs of the algorithm to be performed than $\mathcal{MM}$AS-P given the same amount of time.

7 Conclusions

Typical ACO algorithms for shop 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 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 examined the solution space produced by the space of dispatching rule assignments on a number of commonly studied benchmark JSP instances. Crucially, when using the four dispatching rules examined in this paper, that space does not contain the optimal solution. Given that dispatching rules are themselves simple heuristics, it is plausible that even with a vastly expanded range of rules the optimal solution may still be out of reach. Consequently, any real-world application employing this solution representation not only requires a local search component, but that local search must work directly on the schedules described by the dispatching rules and not the pattern of assignments.

Despite this severe drawback to the alternative solution representation, it does appear to concentrate the search on promising areas of the solution space and, in a constructive algorithm such as ACO, leads to a dramatic reduction in required computation. A comparison of ACO algorithms employing both the traditional solution representation and the alternative show a mixture of results, with neither algorithm clearly outperforming the other across the test instances. However, a slight trend for better performance from the new approach on the larger instances, coupled with its reduced computation times, suggest that it is a good candidate for seeding a local search procedure. As there is an unavoidable interaction between ACO and the local search procedure it uses (as the locally optimised solutions are used to update pheromone information), future work could examine the relative performance of the two approaches when local search is incorporated.

References

1. Bauer, A., Bullnheimer, B., Hartl, R.F., Strauss, C.: Minimizing total tardiness on a single machine using ant colony optimization. Cent. Eur. J. Oper. Res. **8** (2000) 125–141

2. Blum, C., Sampels, M.: An ant colony optimization algorithm for shop scheduling problems. J. Math. Model. Algorithms **3** (2004) 285–308
3. Colorni, A., Dorigo, M., Maniezzo, V., Trubian, M.: Ant system for job-shop scheduling. JORBEL **34** (1994) 39–53
4. Stützle, T.: An ant approach to the flow shop problem. In: 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT '98), Aachen, Germany. Verlag Mainz (1998) 1560–1564
5. Montgomery, J., Fayad, C., Petrovic, S.: Solution representation for job shop scheduling problems in ant colony optimisation. In Dorigo, M., et al., eds.: 5th International Workshop on Ant Colony Optimization and Swarm Intelligence, ANTS 2006, Brussels, Belgium. Lecture Notes in Computer Science, Vol. 4150. Springer Verlag (2006) 484–491
6. Beasley, J.E.: OR-Library, http://people.brunel.ac.uk/~mastjjb/jeb/info.html (2005)
7. van der Zwaan, S., Marques, C.: Ant colony optimisation for job shop scheduling. In: 3rd Workshop on Genetic Algorithms and Artificial Life (GAAL 99). (1999)
8. Montgomery, J., Randall, M., Hendtlass, T.: Structural advantages for ant colony optimisation inherent in permutation scheduling problems. In Ali, M., Esposito, F., eds.: 18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2005), Bari, Italy. Lecture Notes in Artificial Intelligence, Vol. 3533. Springer-Verlag (2005) 218–228
9. Montgomery, J., Randall, M., Hendtlass, T.: Solution bias in ant colony optimisation: Lessons for selecting pheromone models. Computers & Operations Research (in press) Available online: doi:10.1016/j.cor.2006.12.014
10. Stützle, T., Hoos, H.: $\mathcal{MAX} - \mathcal{MIN}$ ant system. Future Gen. Comp. Sys. **16** (2000) 889–914
11. Dorigo, M., Stützle, T.: The ant colony optimisation metaheuristic: Algorithms, applications and advances. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. International Series in Operations Research and Management Science, Vol. 57. Kluwer Academic Publishers, Boston, MA (2002) 251–285
12. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press (2004)
13. Blum, C., Sampels, M.: Ant colony optimization for FOP shop scheduling: A case study on different pheromone representations. In: 2002 Congress on Evolutionary Computation. (2002) 1558–1563
14. Montgomery, J., Randall, M., Hendtlass, T.: Search bias in constructive metaheuristics and implications for ant colony optimisation. In Dorigo, M., et al., eds.: 4th Int'l Workshop on Ant Colony Optimization and Swarm Intelligence, ANTS 2004, Brussels, Belgium. Lecture Notes in Computer Science, Vol. 3172. Springer-Verlag (2004) 390–397