Universiteit Leiden
Opleiding Informatica

Insect Division of Labour

Applied to Online Scheduling

Name: Koen van der Blom

Date: 11/11/2014

1st supervisor: Prof. Dr. Thomas H.W. Bäck
2nd supervisor: Dr. Michael Emmerich

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Insect Division of Labour Applied to Online Scheduling

Koen van der Blom

November 11, 2014

Abstract

Scheduling in the painting facilities of automotive manufacturing plants is a complex process. The paint loss caused by machines continuously switching colours leads to significant costs. Optimising these schedules while assigning new jobs in real time and dealing with machine break downs is therefore an important issue in the truck painting problem.

Previous successes in solving this problem with algorithms inspired by colony insects made evident that this is an area worthy of further investigation. Both improving on existing methods and looking from other angles through previously unconsidered models of division of labour inspired by insects may lead to solutions closer to the the optimum.

Improving the understanding of the strengths and weaknesses of various approaches to scheduling for painting trucks is done by comparing them on a singular problem set. Along with the previously used methods and yet unconsidered models is a proposed algorithm building on those existing approaches. To measure the performance of all the used approaches the total setup time, throughput and flow time were kept track of. At the same time queue and storage usage were measured to avoid solutions with unrealistic requirements.

As newly considered model, foraging for work showed a phenomenal performance in terms of the total setup time. This was primarily the case in experiments with uniform colour distributions and those with heavy workloads, which the model was able to handle best across all measured statistics. For a realistic colour distribution it was also competitive on the front of total setup time. While the proposed algorithm could not compete with the performance by foraging for work on the total setup time, it showed improvement over the methods it used as inspiration in most situations. With a realistic colour distribution the proposed algorithm was also competitive with foraging for work on total setup time and outperformed it on most other measures.

Foraging for work is presented as a new and important competitor in optimising for the total setup time. Further, the importance of evaluating the performance of new approaches which may excel in different areas deserves emphasis. The proposed algorithm serves as another step forward in decreasing setup times while maintaining a high throughput.

Keywords: online scheduling, optimisation, division of labour, colony insects, swarm intelligence

Contents

1 Introduction

Both insect colonies and online scheduling aim for maximal performance, while acting on limited information. In colony insects, this leads to a division of labour over the many individual insects. For online scheduling, this division is done over operational units, be that personnel, factory machines or computer servers. Due to the similarities between these two, it is not much of a stretch to apply concepts of one to improve the other.

Morley et al. [22] [20] [21] applied chaos theory to truck painting while working for General Motors. Using chaos theory, they moved job assignment responsibilities from an overseeing control unit to the painting machines themselves. Seeing the work by Morley et al. [22] [20] [21], both Campos et al. [6] and Cicirello and Smith [8] realised that the chaos theory approach corresponds to what can be observed in insect colonies, where the individuals select their tasks without any global supervision. Using the reinforced threshold model described by Théraulaz et al. [31], Campos et al. [6] compared an insect based task allocation algorithm to the market based approach Morley et al. [22] [20] [21] used. This was done with the aim of showing the viability of insect inspired division of labour in application. Around the same time, Cicirello and Smith [8] were also creating an algorithm based on the work by Théraulaz et al. [31]. Their focus was on describing a competitive scheduling algorithm. Following these initial two teams, several others worked on analysing and improving their approaches in a number of areas. This led to more effective algorithms from, among others, Nouyan et al. [23] [24] and, most recently, Meyyappan et al. [18].

Despite there being a range of models describing colony insect division of labour in biology, so far all approaches considered for application originate from the reinforced threshold model Théraulaz et al. [31] described. How the approaches so far compare in performance is also unclear. This problem originates in different authors all using other variations of the problem and, sometimes slightly, different test cases.

This work contributes to the field by empirically comparing, a selection of both existing work and adaptations of not previously applied models of division of labour in colony insects, on a singular set of problems. In addition, a proposed algorithm combining aspects from two similar approaches is showcased.

This document is structured as follows. Section 2 introduces the problem with a description of a wider known problem to show how it differs, followed by a formal notation of the problem. Related work is included in Section 3. Then, Section 4 details how insect behaviour in nature relates to the approaches to solving the problem which are employed in this work. After that, Section 5 starts by explaining algorithms that were previously applied to the problem and variations on those. The same section continues with descriptions of the insect division of labour models included in this work, which were not previously used to solve the problem. Section 6 provides details on the proposed algorithm. This is followed by the experimental setup, described in Section 7. Naturally, Section 8 then follows with the results and their evaluation. I-n conclusion, the most important results are summarised, their significance is discussed and further work is proposed in Section 9.

2 Problem

This section introduces the problem domain of this work. By doing so, the reader will be provided with a solid understanding of the problem and its relation to a selection of common scheduling problems. As a starting point, the widely known job shop scheduling problem is described. Following that, the flow shop and parallel machine problems lead into the truck painting problem that is the subject of this research. The problem description concludes with the formal notation of this problem.

2.1 Job shops, flow shops and parallel machines

One of the most general and widely known scheduling problems is the job shop scheduling problem. This concerns the scheduling of n jobs over m machines, distributed over k stages. Jobs may consist of multiple operations, but no more than k. The sequence in which the operations pass through the stages may differ per job (though the operation order within a job is fixed). For example, in Figure 1 the layout of a job shop for $m = k = 3$ is shown. A job $J_1(S_1, S_2, S_3)$ may go through the stages in the order shown in Figure 1, while job $J_2(S_3, S_1, S_2)$ follows an alternate ordering and job $J_n(S_3, S_2, S_1)$ yet another. Each operation has a processing time p, which is the time a job requires on a machine for the corresponding stage.

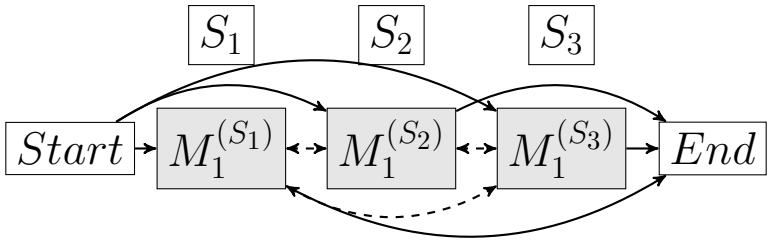


Figure 1: A simple job shop with three machines and stages. Dashed edges indicate bi-directionality.

The flow shop problem is a variant of the job shop where the operations of every job J must have the same sequence through the stages. As such, every job J_1 through J_n for the flow shop in Figure 2 will have the sequence (S_1, S_2, S_3). Both flow and job shops may have multiple machines in one or more of their stages, this is referred to as a flexible, or hybrid, shop. This is also shown in Figure 2, where the second stage has two machines.

Single-stage scheduling problems where $m > 1$, are referred to as the parallel machine problem. Scheduling jobs in all of these environments is done with some objective in mind, like minimising the makespan. The makespan refers to the amount of time in the schedule between the start of the first job and the end time of the job that finishes last. In other words, given the processing time of each individual job, the task is to optimally distribute them over the machines and schedule them such that the entire collection of jobs is completed in the shortest possible amount of time. With the availability of all job information such a

problem can, at least theoretically, be optimally solved.

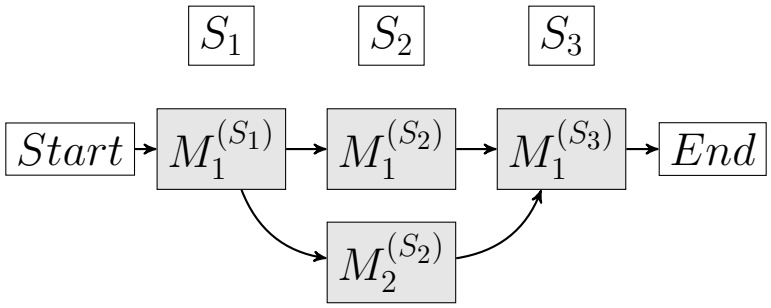


Figure 2: A flexible (or hybrid) flow shop, at least one stage has multiple identical machines.

When the problem however concerns an online problem, as in Figure 3, this is a different matter. Online scheduling refers to the problem class where a decision on the allocation of jobs has to be made based on limited information. Generally, only the assignment of jobs so far and information (e.g. processing time) about the next job to be scheduled are available. This job must then be allocated to one of the machines, before information about the next job after that becomes available. In fact, usually it is unknown whether or not there even will be a next job.

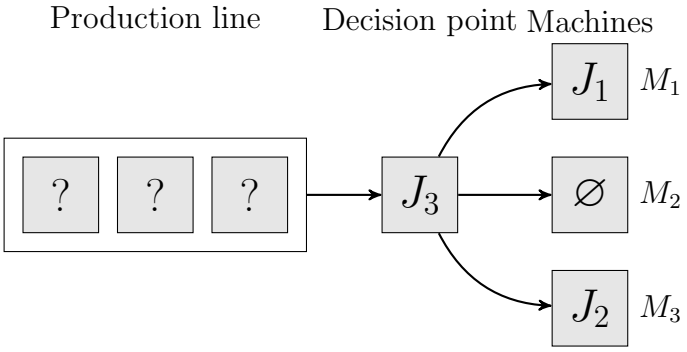


Figure 3: An m machines version of the online parallel machines problem.

2.2 Painting trucks

Working for General Motors (GM), Morley et al. [22] [20] [21] had to deal with the truck painting problem. Within the painting facilities of GM, colour changes of the painting machines resulted in a significant loss of paint. In fact, almost as many colour changes were being used as trucks were painted. From personal communication with a Ford researcher, Cicirello and Smith [8] noted that paint switches tend to cost somewhere between a sixth and an eight of the paint needed to paint a single truck. Through optimising the painting schedules of these machines, Morley et al. [22] [20] [21] aimed to reduce the paint loss.

The situation was one in which trucks come off a production line at some rate. Once a

truck comes off this production line it has to be painted. There are multiple painting booths, which can each paint one colour at a time. The colour of a truck is only known once it comes off the production line, so the schedule cannot be made in advance and has to be generated dynamically. Clearly this concerns an m machines online parallel machine problem.

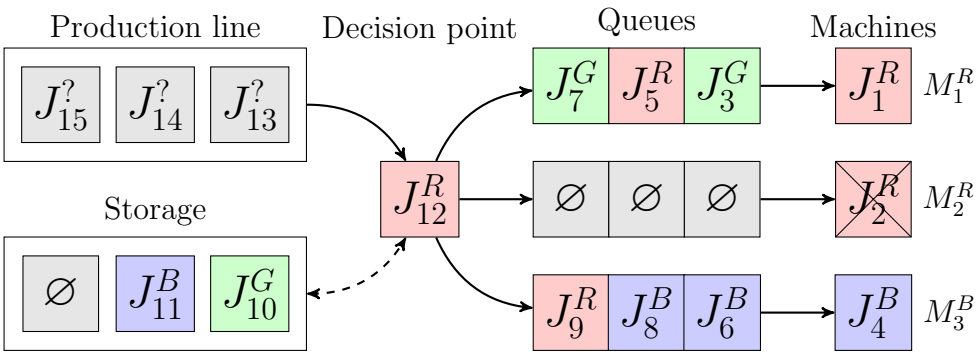


Figure 4: The truck painting problem, an m machines version of the online parallel machines problem.

As becomes evident by looking at Figure 4, the problem has a number of other characteristics. Note that the job situation considered in the figure is not a realistic one, as the system should rarely (if ever) get clogged like in the figure, let alone after such a small number of jobs. This does however provide the ability to show the various quirks of this problem.

Painting booths can switch between colours (R, G, B), which costs time and, as previously mentioned, paint. Due to this, it is advantageous to let machines specialise on a single colour as much as possible. Additionally, painting booths can break down with some probability, such as machine M_2 in Figure 4, indicated by it being crossed out. This causes them to stop processing for a random amount of time (within some range). Every painting booth has a queue of equal length, limiting the number of trucks that can be waiting for a single painting booth.

In case a truck is not assigned to any painting booth for some reason (e.g. all queues are already full), it will be placed in storage. At every time step both the truck(s) coming off the production line and any that may be in storage will go through a process, depending on the applied algorithm, in an attempt to assign them to a booth. In a real situation this storage space is obviously finite, but the size is generally undefined and can therefore, within simulations, be considered infinite. Even so, its use should be monitored since having more than a few trucks in storage can be considered as a failure of the system.

Finally, there is the concept of priority. Although this will not be taken into account in this work, some solutions to the problem did consider this situation, therefore awareness of this is important. A priority job is, as is evident from the name, a job that is somehow important and needs to be finished quickly. Now, the reason this is not taken into account here, is because it is unclear what the impact of priority jobs is. Acting on priority jobs may in fact have a negative impact on the objective function due to the lack of a penalty when high priority jobs are ignored. The ability to simply ignore this problem characteristic,

to improve results, makes that its inclusion would result in an unfair comparison towards algorithms that do include priorities in their solution.

2.3 Formal notation

Formal notation of scheduling problems is done in triplets of $\alpha|\beta|\gamma$, as proposed by Graham et al. [15]. In such a triplet, α represents the shop configuration, β shows which constraints and assumptions are used and finally, γ is the objective function to optimise. Since the original description by Graham et al. [15], there have been a fair number of extensions to this notation, the ones used here have been listed in the review work by Chen et al. [7], Allahverdi et al. [1] [2], and Ruiz and Vázquez-Rodríguez [28].

Notation	Meaning
α	Shop configuration
β	Constraints
γ	Objective function
P_m	Parallel machines problem for some m machines
$online$	An online problem is considered
r_j	All jobs j have a release date
S_{sd}	Setup times are sequence dependent
$block$	Jobs are blocked from entering the next stage when all queues are full
$brkdwn$	Machines may break down
$p_j = p$	All jobs j have an equal processing time p
TST	Total setup time
TSC	Total setup cost
F	Flow time
$\sum U_j$	Throughput

Table 1: Summary of used notation.

In the previous subsection it has been determined that truck painting concerns the parallel machines problem, therefore $\alpha = P_m$, which refers to the parallel machines problem with identical machines P, for some m machines.

A number of constraints are part of the problem, because of this the β field is denoted as follows: $\beta = online, r_j, S_{sd}, block, brkdwn, p_j = p$. First, $online$ refers to the fact that this is an online problem. This means that job information will only be available for jobs whose release date r_j has passed. With online problems the allocation of a job is irrevocable, so assignments cannot be changed after the arrival of a new job. As mentioned r_j denotes the release date, every job j has such a release date. For truck painting, this is the time at which a truck comes off the production line. Next, S_{sd} indicates the sequence dependence sd of setup times S. If the same colour is sequenced, no setup time is required; if different colours are sequenced, there is a setup time. The fourth restriction, $block$, indicates limited buffer sizes between stages, forcing jobs to wait in a previous stage when the buffer is full. This is used to denote the restriction on queue sizes of machines and the use of the storage

space when necessary. As fifth component, the *brkdwn* constraint indicates the possibility of machine break downs. Finally, $p_j = p$ defines the processing time of every job j as equal to p, this makes evident that the painting time is the same for every truck.

Three objectives are considered for optimisation. First off, there is the total setup time (which is directly related to the number of flushes) $\gamma = TST$. The total setup cost ($\gamma = TSC$) is not considered since only an estimate of the cost is available, which, due to the direct relation, can be calculated provided the total setup time, if so desired. As second objective, the average flow time (also referred to as cycle time) $\gamma = F$ is optimised. This refers to the time from the moment a job enters the system, until it is finished processing. The third and final objective is the throughput $\gamma = \sum U_j$. When a job j is completed $U_j = 1$, otherwise $U_j = 0$. As such, this is simply the sum of completed jobs. These three objectives may be considered individually, or in some combination. Ideally, optimisation would happen for all of these objectives, but the total setup time is the most significant for the goal of reducing production costs. Therefore, it is prioritised as long as the other objectives do not deteriorate to excessively poor results.

The makespan ($\gamma = C_{max}$) is, unlike in Campos et al. [6], not considered. This is the total amount of time it takes for all jobs to complete. It is not considered, because it is not a very interesting objective for online problems. Average flow time is already a good indicator for when the last job will be finished (release time of the last job + average flow time). More significantly, there is barely any variation in the makespan. Due to the release times of the jobs, there is a very clear limit to the minimal possible makespan (the release time of the last job + processing time). As such, the makespan will only significantly deviate when an algorithm produces schedules which are unable to keep up with the production rate. All this is of course under the assumption that the processing capacity is sufficient to process at least as fast as the production rate.

The truck painting problem can then formally be written as in Equation 1 below. Clearly, for this problem the number of constraints are an important source of complexity.

$$P_m|online, r_j, S_{sd}, block, brkdwn, p_j = p|TST, F, \sum U_j \tag{1}$$

3 Related work

Some research related to the problem considered in this work is discussed in this section. This includes painting problems as well as more general automotive factory problems. A summary of scheduling work in automotive factories is also referred to, such as to provide a starting point for further work.

Yu and Ram [34] focus on a similar problem to the one considered in this work, which does not include break downs. It does however deal with multi-stage jobs, optimising the queue order, due dates for jobs, machines that can only process a certain selection of job types and machines with different processing times. They too draw inspiration form the popular threshold model, described by Théraulaz et al. [31], for their approach. In the comparison they consider their own approach, an auction based model and a naïve model.

In a case study of a Korean factory, Moon et al. [19] discuss a situation with some very factory specific considerations. The focus seems to be on optimising the scheduling of the storage area before the painting stage. Similarities are in the consideration of multiple paint colours, different colour distributions and the need for setups when changing to a different colour. They don't compare to different algorithms, just a number of different alternatives of their own. One interesting point they mention in regards to paint costs from setups is an approximate paint loss of 0.46 litre when changing colour. Although this seems to be for regular cars, as opposed to trucks, it gives an indication of the magnitude of the problem.

The more general production scheduling process in a car factory is considered by Inman and Schmeling [16]. They mention how the order in which cars go through production changes due to the need to rework parts at various stages during the process. This is (one of) the reasons cars arrive at the paintshop in a non-optimal order and thus scheduling is required there.

In the work by Ding and Sun [9] there is also a focus on the more general production process of cars, though painting is also included. An interesting point here is that different departments (body, painting and assembly) in the factory have different scheduling considerations, another reason scheduling is needed before painting. They consider a sorting area before the painting process to optimise the painting schedule, after painting they then place the cars back in the original order they arrived in at the painting area.

Boysen et al. [5] provide a summary of work on scheduling in car manufacturing plants, including painting problems. This includes the articles discussed above, except the one by Yu and Ram [34]. Moreover the articles by Campos et al. [6], Cicirello and Smith [8], Kittithreerapronchai and Anderson [17], Morley et al. [22] [20] [21] and Nouyan et al. [23] [24] are also discussed.

4 Division of labour in social insect colonies

This section provides a description of social insect behaviour leading to a division of labour in their colonies. Following that, an explanation is provided as to why algorithms based on this behaviour form a viable approach to solving the truck painting problem.

4.1 Behaviour

A variation of factors are believed to influence division of labour in colony insects. Behavioural differences can be observed based on age, size or shape, genetics, the colony life cycle, experience, environmental stimuli, developmental variations and social interaction. Below follows a concise description of how all of these factor into division of labour. More details and references may be found in the works of Beshers and Fewell [3], and Duarte, Weissing Pen and Keller [10].

Different age groups tend to perform different tasks, younger insects will generally perform tasks in the nest, while older individuals go outside to forage and defend the nest. As work outside the nest is more dangerous, it makes sense to send out the least valuable (those with the shortest remaining lifespan) individuals. Dividing labour based on shape happens because some individuals are simply more suitable for certain tasks based on their morphology. For example, an ant with larger mandibles may be more fit to defend the colony. Genetic differences appear due to the selection of different mates in the colony. Groups with different ancestries tend to prefer different task sets, although it remains unclear whether this has an effect on the colonies efficiency. During the colony life cycle the needs of the colony change. When the colony is in a growth phase it requires different worker distributions than during the reproduction phase. As such, insects may be incited to change tasks with phase changes. Moreover, different phases also tend to lead to another distribution, generally in terms of morphology, of the newly produced larvae. Previous task experience influences the division of labour as insects that successfully completed a task of a certain type are more likely to take on another task of the same type. There is evidence supporting that experienced individuals are not just more likely to perform a task, but do so more efficiently. Stimuli from the environment also play a role. Individuals that come across a certain task type regularly, e.g. they run into garbage in the nest a lot, may be incited to switch from their current task as the environment indicates a shortage elsewhere. The developmental environment is believed to play a role in division of labour as well. One example is how the temperature during larval development later influences at which temperatures an individual responds by moving brood to keep it in an optimal environment. Social interaction can be observed in insects recruiting other workers for a task type. Conversely, workers may also be inhibited from performing certain tasks, by transferring inhibiting substances between each other.

4.2 Why insect algorithms?

Not all of these factors translate equally well to application in the truck painting problem. The vast majority is however applicable to a more general production environment. Here

follows a quick look at how these factors could be related to such an environment. This will indicate the great overlap in these different environments which makes it sensible to take a look at colony insect inspired algorithms applied to manufacturing problems, such as truck painting.

Age may matter in that older machines are more likely to break down and therefore assigning (priority) jobs to them instead of machines that are less likely to break should be avoided where possible. Machines of different sizes and shapes may be more effective in performing certain tasks than others, think of newer versions with further optimised designs working alongside older versions. Genetic difference can be related to a manufacturing environment as machines from different manufacturers working alongside each other. This may not be a common occurrence, but it could happen that a factory changes to machines of another manufacturer, while keeping older machines (that still work) around while they last. Similarly uncommon may be a restructuring of operations within a factory, while keeping parts that aren't (currently) being changed operational. This could be seen as a phase change like in the colony life cycle. Experience is easily linked to setup operations, like colour changes in the truck painting problem. Here, 'experienced' machines require less time for another task of the same type compared to machines that require a setup. Environmental stimuli could be explained as what a machine observes in terms of job availability, or the required movement distance (either by the job or the machine) in order to process a job. This may then play a role in the decision about whether or not to take a certain job. Developmental variations don't seem to have an obvious counterpart in manufacturing, at least not different from size and shape or genetic variations. Social interaction would be the sharing of information between machines about their status. This can include things such as their queue size and whether they require repair, be that planned or unexpected.

The considered variation of the online parallel machines problem deals with multiple operation types (colours), with a cost for switching between them (paint and time loss). Having multiple operation types makes this a problem fit for insect division of labour models, which rely on their ability to learn a preference for one or more operation types to divide labour. Moreover, the online aspect of the problem means there is another similarity to the situation faced by social insects in nature. Decisions in social insect colonies are made based on a very limited view of the whole. Assuming this is properly modelled, this should, at least in appropriate situations, lead to a similarly effective division of labour as observed in nature.

5 Algorithms

In this section, various algorithms that were used in the past, or will be considered in this work, are described. Starting with the market-based approach, described by Morley et al. [22] [20] [21], which can serve as a baseline. Following that, the algorithms described by Campos et al. [6], and Cicirello and Smith [8] are explained. Some of the variations on those two, that have been introduced since, have also been included. Finally, insect inspired models that have not yet seen application to this problem are introduced.

5.1 Market-based approach (Morley et al.)

Morley et al. [22] [20] [21] approached the truck painting problem, not in the traditional fashion of trying to describe to the system what should happen in every possible situation, but by describing how to achieve the desired result, the trucks being painted. Meaning that, instead of having a global controller telling everyone what to do, everyone knows for themselves what they need to do to achieve the desired result (at least for their part). No longer is there a need to understand what is happening in the complete system, an understanding of the local situation is sufficient.

In 1993, the system was described through the three rules below [22].

1. Assign the truck to the booth with the shortest non-full queue whose last truck is of the same colour as this next truck (if such a booth exists).
2. Assign the truck to a booth with an empty queue if such a booth exists.
3. Assign the truck to a booth with the shortest non-full queue if such a booth exists.

Oddly enough, these rules describe the perspective of a global controller, contrary to the perspective of the considered solution. Cicirello and Smith [8] noted the redundancy of the second rule, given the third. In 1996 [20] and 1998 [21], the four rules below were described, which take the machine perspective. Note that this description includes important (priority) jobs, which as described in Section 2.2 will not be taken into account in this work.

1. Try to take another truck the same colour as the current colour.
2. Take particularly important jobs.
3. Take any job to stay busy.
4. Do not take another job if the paint booth is down or the queue is full.

While both provide a decent representation of the system, neither of these two rule sets are completely accurate descriptions of the system. The actual system (as described in [21]) works in a market-like fashion, where everyone bids and the highest bidder wins. This bidding competition is not at all represented by either rule system. The height of the bid

does however depend on the concepts presented in the rules: colour similarity, queue length and job importance. As such, a bid can be placed as a result of a few simple rules. Based on those bids, a scheduler then assigns jobs to the highest bidder. Together, this leads to a market-based solution where assignments are a result of a bidding process.

The solution provided by Morley et al. [22] [20] [21] resulted in a 10% reduction in the required number of colour changes. Consequently, this led to a saving of a million dollar, compared to the previously used system, in the first nine months after being implemented, in a single painting facility.

Beyond the given rule sets and the use of a bidding mechanism, no implementation details were provided. For their comparison, Campos et al. [6] described these rules as the bid function B_k in Equation 2, for every machine k. Every truck i has a priority w_i and a colour c_i, in the equation, $c_{i,k}$ indicates one when the last truck in queue of machine k has the same colour as truck i, or zero when the colours differ. Then, ΔT_k is the time until machine k would start to paint truck i. Each of these components has a weight, respectively P, C and L. When the w_i term is ignored, and therefore P, this becomes a function consisting solely of queue length and colour similarity. This doesn't actually change the outcome of the algorithm, as both w_i and P are the same for every bidder. For this reason, the implementation used later in this work removes $P \cdot w_i$ from the equation, leaving only $1 + C \cdot c_{i,k}$ in the nominator of this equation.

$$B_k = \frac{P \cdot w_i \cdot (1 + C \cdot c_{i,k})}{\Delta T_k^L} \tag{2}$$

In Equation 3, the components of ΔT are detailed. It consists of the number of trucks in queue q_k, for machine k, multiplied by the paint time t_p, plus the number of times the colour of the next truck in queue is different from before n_k^f, multiplied by the flushing time t_f, plus the remaining paint time on the truck currently being painted t_k^r. All of these are measured in seconds for this work, Campos et al. [6] did not specify the unit they used.

$$\Delta T_k = q_k \cdot t_p + n_k^f \cdot t_f + t_k^r \tag{3}$$

The highest bidder resulting from the equation is assigned the truck. When a tie occurs, the bidder that does not need to change colour wins. If there are multiple bidders not needing a colour change the lowest numbered one would be given the job. When there is a group of bidders that all have a colour different from what the truck requires, one is chosen at random.

Queue length does not play a role in breaking ties, since it is not possible for bidders with the same colour to have the same bid, without their queue length being equal. This is the result of two properties of this algorithm. First off, the priority is the same for every bidder. Secondly, after applying the colour tie breaking rule, the remaining bidders all have the same value for the colour factor of the function. This leaves only ΔT_k^L as variable factor in the function. Therefore, this variable must be the same for all remaining bidders in order for them to have bid the same.

5.2 Ant-based (Campos et al.)

Inspired by the fixed threshold model described by Bonabeau et al. [4] and the later response threshold model by Théraulaz et al. [31], Campos et al. [6] proposed an insect inspired algorithm. The idea behind both the threshold model and the insect inspired algorithm is that individual insects have a, possibly variable, threshold θ, for every task type. When the stimulus for a task surpasses this threshold the insect will perform tasks of that type. The stimulus is the perceived need for a task through interaction with the environment and other colony members. E.g., often running into trash may incite nest cleaning, while only occasionally observing trash likely results in sticking with the current task. This threshold is believed to be influenced by a number of factors, such as genetics and experience, as discussed before in Section 4.

Applying this threshold model to the truck painting problem led to the following approach. A colour demand D_j for every colour j was included, as shown in Equation 4. This demand consists of the sum of the priorities w_i, for every truck i that has not yet been assigned to a machine. The Dirac delta function δ, results in only the trucks requiring a colour $c_i = j$ being included in the demand of a colour j. Despite priorities, and as such the described demand, not being included in the problem definition used in this work, it is included here because it plays a role in the bid function Campos et al. [6] described.

$$D_j = \sum_i w_i \cdot \delta(c_i - j) \tag{4}$$

With the goal in mind of showing the similarities between the market-based and ant-based approaches, Campos et al. [6] described a bid function, although bidding is probably not a very ant-like process. The bid of a machine is described in Equation 5. D_{c_i} refers to the just introduced demand for a colour c of truck i. Then, θ_{k,c_i} is the threshold of machine k for colour c. In the model, the thresholds are bound between θ_{max} and θ_{min}. Here, ΔT_k is the same as in the market-based approach and as such also computed the same as in Equation 3. The terms α and β are weights for the influence of θ and ΔT_k respectively.

$$P_k = \frac{D_{c_i}^2}{D_{c_i}^2 + \alpha \cdot \theta_{k,c_i}^2 + \Delta T_k^{2 \cdot \beta}} \tag{5}$$

Cicirello and Smith [8] simplified the bid function P_k through removing D_j from the function. This was done by changing the function to the one shown in Equation 6. Since the calculation of D_j does not incorporate any machine specific aspects, and is therefore the same for every machine, this is possible. Therefore, P_k would change in the same way for every machine k, resulting in the same machine winning the bid for the truck. As such, D_j can be replaced by a static value. Alternatively, D_j and the division could be left out entirely (leaving: $\alpha \cdot \theta_{k,c_i}^2 + \Delta T_k^{2 \cdot \beta}$). The lowest bidder would then be the winner, leading once more to the same result. From this observation it becomes clear that the colour priorities and demands do not actually play a role in the allocation process Campos et al. [6] proposed. As such, these concepts can be left out. This also immediately prevents any potential issues with the exclusion of priorities from the problem.

$$P_k = \frac{1}{1 + \alpha \cdot \theta_{k,c_i}^2 + \Delta T_k^{2 \cdot \beta}} \tag{6}$$

The ant-based algorithm handles ties in the same way as previously described for the market-based approach, except for one part. When there are multiple machines that have the same colour as required, the one with the shortest queue will be selected, before the rule that takes the lowest numbered machine is considered. This is because, with the bid function of Campos et al. [6], it is not clear that queue length is the only remaining factor.

Threshold values of each machine are updated when a truck is assigned to a machine. The machine k that won the bid, has its threshold for that colour decreased by ξ, with the rule in Equation 7. Equation 8 then increases the threshold of all other machines m, by ϕ.

$$\theta_{k,c_i} \leftarrow \theta_{k,c_i} - \xi \tag{7}$$

$$\theta_{m,c_i} \leftarrow \theta_{m,c_i} + \phi \tag{8}$$

From the comparison between the market-based approach as described by Campos et al. [6] and their own ant-based approach, a couple of results are noteworthy here. There was an improvement (reduction) of the number of flushes required when painting, using the ant-based approach. Although this improvement was not in all situations significant. Deviations from the makespan in both directions, varying between the considered situations, were produced, although these were primarily insignificant.

5.3 R-Wasps (Cicirello and Smith)

Cicirello and Smith [8] proposed an other insect inspired algorithm, called R-Wasps, based on the same response threshold model by Bonabeau et al. [4] and Théraulaz et al. [31]. Differences between the model come from the manner in which the concept of thresholds is translated into an algorithm. Where Campos et al. [6] used the threshold to determine the height of bids made by machines, Cicirello and Smith [8] use the threshold in a more insect-like manner to determine whether or not a machine will compete for a certain task at all. In situations with multiple bidders, this is followed by a wasp-inspired dominance contest, based on work by Théraulaz et al. [30] [29], to determine which machine acquires the task.

They used Equation 9 to determine for every machine w whether or not to bid, based on the resulting probability P. This probability is a result of a combination of the threshold $\theta_{w,j}$ for a job type j and the stimulus S_j, emitted by a job of the same type. Like in the model described by Campos et al. [6], the thresholds are bound between θ_{max} and θ_{min}. The stimulus of a job is based on how long it has been waiting for assignment. So, the longer the job remains unassigned, the stronger the stimulus becomes.

$$P(bid|\theta_{w,j}, S_j) = \frac{S_j^2}{S_j^2 + \theta_{w,j}^2} \tag{9}$$

Since the exact definition of the the stimulus is not clear from Cicirello and Smith [8], for this work it is defined as follows. The stimulus of a job is one, plus the time in seconds this job was been awaiting assignment. It starts at one at the moment of creation in order to avoid a possible division by zero in Equation 9, which may occur when both the stimulus and threshold are zero. Setting the minimum of the stimulus to one, rather than that of the threshold (Cicirello and Smith [8] tuned θ_{min} to one), has to do with the use of this same equation for calculating the probability to bid by a variation on this algorithm introduced later in this section which considers a θ_{min} value of zero.

In their algorithm, threshold updates happen at every time step, as opposed to on job assignment, like Campos et al. [6] did with their ant-based approach. Cicirello and Smith [8] mention that these updates happen every twelve seconds. It is however unclear whether everything happens in twelve seconds steps. This work assumes only threshold updates are done every twelve seconds, job production and assignment remains in 60 second steps. Every machine has its thresholds for the active colour (if any) decreased by δ_1, as shown in Equation 10. Where active colour refers to the colour the machine is currently painting, or the colour it is doing a setup for.

$$\theta_{w,j} = \theta_{w,j} - \delta_1 \tag{10}$$

All other colours of active machines, i.e. machines that are painting or doing a setup, then have their thresholds increased by δ_2, this is shown in Equation 11.

$$\theta_{w,j} = \theta_{w,j} + \delta_2 \tag{11}$$

Finally, idle machines, i.e. not painting or doing a setup, decrease all their thresholds by δ_3^t, as in Equation 12. Where t is the amount of time the machine has been idle. Cicirello and Smith [8] do not specify the unit of measurement for t, this work uses seconds.

$$\theta_{w,j} = \theta_{w,j} - \delta_3^t \tag{12}$$

When there are multiple bidders, Cicirello and Smith [8] decide which machine acquires the job based on the previously mentioned dominance contest. The dominance of each machine w, is based on their 'force' F_w, shown in Equation 13. The force consists of the processing time $T_{w,p}$ and the setup time $T_{w,s}$, of all the jobs in the machine's queue. Again, seconds are used, although this was unspecified in the original description. To avoid a possible division by zero in the dominance contest explained in Equation 14, the force must be at least one. Note that, contrary to intuition, a lower force value is better.

$$F_w = 1.0 + T_{w,p} + T_{w,s} \tag{13}$$

Equation 14 describes the calculation of the probability P of the first machine, with force F_1, winning the contest against another machine with force F_2. When there are more than two competitors, a single elimination tournament of dominance contests will be held. Seeding is done based on the force variables of the competitors. Cicirello and Smith [8] did not further specify the seeding process. For the experiments later in this work it is assumed

the competitors are re-seeded at every round. The force variables are used in the seeding process such that the highest and lowest compete, the second-highest and second-lowest, etcetera. To avoid odd numbers of competitors C appearing at any stage of the tournament, the $2^{\lceil log_2(C) \rceil} - C$ machines with the largest F proceed to the second round without contest. It doesn't seem intuitive to let the machines with the largest F proceed as these are the least fit for the job, but this is how Cicirello and Smith [8] described it. The experimental implementation used for this work does the reverse, where the lowest, most fit, machines skip the first round.

$$P(\text{Wasp 1 wins}|F_1, F_2) = \frac{F_2^2}{F_1^2 + F_2^2} \tag{14}$$

In the comparisons made by Cicirello and Smith [8], their R-Wasps algorithm consistently significantly outperformed both the approaches of Morley et al. [22] [20] [21] and of Campos et al. [6], in terms of the required number of flushes. Flow times on the other hand are significantly higher for R-Wasps, except for one problem where long setup times when switching colours caused the other algorithms to entirely fill up their queues, forcing them into longer flow times. The flow time refers to the length of time between the release of a job from the production line to its completion. This indicates that R-Wasps, on average, produces longer queues, or makes more use of the storage space, or a combination of those two.

It should be noted here that Campos et al. [6] avoided using the storage space as they considered its use a system failure, while Cicirello and Smith [8] embraced its use. Using the storage space increases the pool of jobs that machines can select from, leading to a situation where it is easier to sequence jobs of the same colour. To be fair, the option for a machine not to bid in R-Wasps likely also has a positive effect on the number of flushes used, regardless of the use of storage space, since this allows machines specialising in frequently occurring colours to avoid having to switch. Knowing this, it can be stated that these results do not mean that either algorithm is better than the other in a general sense. R-Wasps however clearly performs better in the problem variations considered by Cicirello and Smith [8] for the objective of minimising the number of flushes used.

5.4 Variations

A number of variations on the algorithms by Campos et al. [6] and Cicirello and Smith [8] will be discussed in this subsection.

5.4.1 Ant-Task-Allocation (Nouyan et al.)

Nouyan et al. [23] [24] considered five modifications to the R-Wasps algorithm proposed by Cicirello and Smith [8]. The more recent article by Nouyan et al. [24], does not include the fourth change discussed (no bid for a created job), but it is included here and considered as part of the approach in experimentation.

1. TUR (Threshold Update Rules)

   One obvious improvement is that Nouyan et al. [23] [24] update the thresholds based on the last job in queue, as opposed to based on the currently processing job. This avoids problems with jobs being added to the end of the queue of a different type than the current last in queue, but of the same colour as the current job.

2. CFV (Calculation of the Force Variable)

   They also include a related improvement, the potential requirement for a setup between the last job in queue and the currently offered job. This is incorporated into the force calculation with the addition of the job setup $T_{w,j}$ into Equation 15. This value will be the setup time when a setup is needed, or zero when no setup is required. Again, the unit was not specified and seconds are used in the experiments.

$$F_w = 1.0 + T_{w,p} + T_{w,s} + T_{w,j} \tag{15}$$

3. DC (Dominance contest)

   The difference in probability to win becomes smaller as the number of competitors grows. Nouyan et al. [24] resolve this by changing the dominance contest to the formula on the left in Equation 16. This formula can, and often will, however result in probabilities greater than one, because of the exclusion of competitor k from the summation. Therefore, experimentation with this algorithm will employ the earlier version of the equation from Nouyan [23], on the right in Equation 16.

$$P_k(F_1, ..., F_n) = \frac{\frac{1}{F_k^2}}{\sum_{i \neq k}^{n} \frac{1}{F_i^2}} \qquad \text{or} \qquad P_k(F_1, ..., F_n) = \frac{\frac{1}{F_k^2}}{\sum_{i=1}^{n} \frac{1}{F_i^2}} \tag{16}$$

4. BCJ (No Bid for a Created Job)

   Jobs with a colour that rarely occurs may end up in situations where no machine bids to acquire them. This may happen as a result of only the thresholds of job types that get processed being actively decreased; leading to a situation where there are no machines specialising in the infrequently occurring job type. In R-Wasps, the only manner in which these jobs may eventually get assigned is by their stimulus increasing with time. Since this can be a relatively slow process, these jobs may remain unassigned for quite long periods. Nouyan [23] increased the chances of jobs with no bidders, by lowering the threshold $\theta_{w,j}$ of every machine w for the corresponding job type j, as in Equation 17.

$$\theta_{w,j} = \theta_{w,j} - \gamma_1 \tag{17}$$

5. IMC (Idle Machine does not Compete)

This is an extra update rule for machines that are idle and don't bid when a job is offered. This rule, in Equation 18, decreases the threshold for the job type that was offered, but not bid on, by γ_2.

$$\theta_{w,j} = \theta_{w,j} - \gamma_2 \tag{18}$$

Nouyan et al. [23] showed how their modifications to the R-Wasps algorithm outperformed the original in terms of idle time and throughput (the number of jobs completed during the runtime), in all the situations they considered. This was done for the BCJ and IMC modifications individually, the combination of BCJ, IMC and CFV, and the combination of all considered modifications together. Moreover, their changes almost always showed improvement in the number of setups, except for one of the six considered situations, where R-Wasps did better than IMC individually, and the BCJ, IMC and CFV combination.

Later, Nouyan et al. [24] measured the makespan by creating an environment with a production rate equal to the optimal processing rate, which could realistically never be achieved due to the inclusion of break downs. In this situation they showed all fifteen possible combinations of the TUR, CFV, DC and IMC modifications (from each individually to all combined) outperformed R-Wasps. Most notably, any combination that included TUR significantly outperformed those without TUR.

It should be noted that in both of the articles, Nouyan et al. [23] [24] considered a situation where there were more machines than colours, as opposed to the reverse considered in the work presented here.

5.4.2 Kittithreerapronchai and Anderson

Kittithreerapronchai and Anderson [17] investigate various parts of the system proposed by Campos et al. [6], without explicitly proposing an improved algorithm. They evaluated parameter optimisation, global or local threshold updates and the rules used for tie breaking.

The differences between local and global update rules as they described them are as follows. First off, they apply updates as in Campos et al. [6], when a machine acquires a job. Local update rules update only the thresholds of that specific machine, both for the type of job that was acquired and all other job types. For global update rules the thresholds of the particular job type are updated for all machines, thresholds of other job types are not changed. This means that the difference is two fold, for which job types and for which machines thresholds are updated.

1. Parameter optimisation

   For single objective functions Kittithreerapronchai and Anderson [17] suggested that parameter optimisation may not require complex optimisation methods. The α parameter does not seem to have much impact as long as it is greater than zero. It should be noted though that the considered range of β is much smaller. Despite β being used in an exponent, a smaller step size for α seems like it could be more interesting. The impact of both parameters may in fact be more similar than their plot suggests.

They provided a gradation in the significance of the parameters, ranging from insignificant to moderate to sensitive. Respectively, these refer to parameters that account for 20%, 20-60% and more than 60% of system performance across parameter space. In the insignificant group were θ_{min} and ξ, moderate were α, ϕ and θ_{max}, finally β was the only parameter classified as sensitive. Note that they did not provide data or graphs for all of these, only a comparison between α and β is provided. It makes sense that not all parameters have an equally significant impact, but a possible improvement of 20% by tuning more parameters is certainly not something to ignore. Additionally, they suggest that it should be relatively easy to achieve a good balance between flow time and the number of colour changes solely through tuning β.

2. Global versus local update rules

   All graphs in their paper seem to support local updates, in their abstract they however state that global updates are better. It is not immediately clear what they base this on. This seems to be due to global updates reaching equilibrium faster than local updates, which they consider important, despite local updates using both fewer colour changes and a shorter flow time.

   Their comparison on flow time between global and local updates shows that the global update rule reaches equilibrium faster. They also show that the local update rule generally produces better results. Combining these two findings it can be said that the use of global or local update rules should be situation dependent. Changing colour distributions, or at least those that change rapidly, may benefit from global updates to quickly reach the new equilibrium. Relatively stable distributions would however benefit from the higher quality results of local updates. In a factory setting it would likely be best to use local updates, since day to day there probably would not be that many changes, at least in the truck painting case. The final colour settings from one day could therefore be carried over to the next day to avoid having to start form nothing in terms of reaching the equilibrium. An other option might be switching between the two at some point in the process. It is however difficult to say how effective this would be due the equilibria being at different points. Following the switch, the new rule set still needs to find its own equilibrium, which may or may not be worth the effort. One remaining question is whether the other possible update combinations, i.e. update all local thresholds and those of the same type globally, or simply all thresholds, might be worth considering.

3. Breaking tie rules

   Kittithreerapronchai and Anderson [17] consider the effectiveness of some tie breaking rules. First, they consider the regular case, as described by Campos et al. [6], where first the colour of the last truck in queue is considered, then the waiting time and finally a random decision. Next, they look at only using the colour followed by a random decision, the waiting time with a random decision and lastly, random by itself.

   They show that using all three rules produces the same results as using the colour and

random. Waiting time combined with random and random by itself also produce the same results, though this is a worse result than the first two approaches. In short, this shows that the waiting time has no impact on the tie breaking in the system Campos et al. [6] designed. Either the colour decides, or a random selection makes the decision. It is however not immediately clear why this would be the case.

5.4.3 Meyyappan et al.

Meyyappan et al. [18] propose improvements to the system described by Cicirello and Smith [8]. They described their changes as three separate models, where each of them includes more changes than the one before.

1. M1

   Model 1 describes two changes, which happen to be the same as two of the changes considered by Nouyan et al. [23] [24]. The first change is the threshold update rule, where threshold updates happen based on the last job in queue instead of the one being processed. Secondly, they change the calculation of the force variable to include the potential need for a setup when the new job is accepted.

2. M2

   Model 2 introduces two additional changes on top of those in Model 1. Threshold updates are done on assignment of a job to a machine like in Campos et al. [6], instead of on time basis. More drastically, Meyyappan et al. [18] decided to allow thresholds only two possible values, zero or θ_{max}. So instead of decreasing or increasing by δ_1 or δ_2, as in Equations 10 and 11, Meyyappan et al. [18] set the threshold to zero or θ_{max} respectively. This was done with the aim of reducing the number of colour changes, which may result from a low threshold for a colour that was used before the current one. As a result of this change, θ_{min} is always zero and therefore no longer a parameter that needs to be tuned. Moreover, δ_1 and δ_2 are no longer used at all, leaving just the δ_3 and θ_{max} parameters.

3. M3

   In Model 3 Meyyappan et al. [18] aim to reduce the need for parameter re-tuning, by automating parameter changes. There is a need for parameter changes when there are changes in the production conditions, as this moves the optimum. Having already reduced the number of parameters from five to two in Model 2, this task has already been simplified. The δ_3 parameter is set to a constant. Dealing with changing situations is done by defining update rules for θ_{max}, using fuzzy logic. The combination of these two parameters then decides how quickly a threshold reaches zero for an idle machine, leading it to take on a new job.

Their two sentence description of their fuzzy system (for Model 3) leaves out essential parts required in reproducing their results. Despite the inclusion of Table 2 to show how θ_{max}

Throughput	Number of setups	Average flow time	θ_{max}
High	High	High	Low increase
High	High	Medium	Medium increase
High	High	Low	High increase
High	Medium	High	Low increase
High	Medium	Medium	Medium increase
High	Medium	Low	Low decrease
High	Low	High	Low decrease
High	Low	Medium	Low decrease
High	Low	Low	Do nothing
Medium	High	High	Do nothing
Medium	High	Medium	Low increase
Medium	High	Low	Low increase
Medium	Medium	High	Low decrease
Medium	Medium	Medium	Do nothing
Medium	Medium	Low	Do nothing
Medium	Low	High	High decrease
Medium	Low	Medium	Medium decrease
Medium	Low	Low	Low decrease
Low	High	High	Do nothing
Low	High	Medium	Do nothing
Low	High	Low	Do nothing
Low	Medium	High	Medium decrease
Low	Medium	Medium	Low decrease
Low	Medium	Low	Low decrease
Low	Low	High	High decrease
Low	Low	Medium	High decrease
Low	Low	Low	Medium decrease

Table 2: Fuzzy rule matrix, as shown in Meyyappan et al. [18].

should change when, for example, the throughput, number of setups and average flow time are all high, practically all other details are missing. Even when assuming the exact same membership functions are defined for the throughput, number of setups and average flow time terms (low, medium and high) based on simulation results, they provide no information at all about how they defuzzified θ_{max}. The range of values considered for the in- or decrease of θ_{max} is not provided which is an essential part of their system's behaviour. The used values of other parameters, such as the initial values of θ_{max} and δ_3 (in case of Model 2 or 3) are not specified either. For these reasons their approach is not included in the empirical comparison of this work, despite how promising the results they show are.

Meyyappan et al. [18] compared their three models and R-Wasps on Cicirello and Smith's [8] problem, which does not consider break downs. Both a situation with one minute setup times and a situation with ten minute setup times were considered.

Starting with the one minute situation, M1 showed a small increase in the number of setups compared to R-Wasps, while M2 and most significantly M3 showed lower numbers of required setups. The throughputs show minimal differences, those of M2 and M3 are slightly

better than the other two. Only M1 has a shorter flow time than R-Wasps, M2 and M3 use longer flow times. Longer flow times seem to correlate with lower numbers of setups.

The ten minute situation shows the same differences in the number of setups. Throughput differences of R-Wasps, M2 and M3 are also similar, though the gap is a bit larger. M1 has a significantly lower throughput than the others. Unlike the other two performance measures, the ordering of approaches by flow time is quite different in this situation. This time, M2 has the lowest flow time, followed by M1, M3 and finally R-Wasps.

In short, M2 and M3 decrease the number of setups required through their changes form the R-Wasps model, while leading to longer flow times for one minute setups and shorter flow times for ten minute setups. The changes in M1 by themselves achieve the reverse, more setups and a shorter flow time.

5.5 Other insect inspired models

In this subsection some other insect inspired models which have not yet been applied to the truck painting problem are described. They are altered where necessary to make it possible to use them on the considered problem.

5.5.1 Self reinforcement

In 1988 Plowright and Plowright [25] described the self reinforcement model, based largely on the same ideas as the (reinforced) threshold model of Bonabeau et al. [4] and Théraulaz et al. [31]. Both employ memory about previous task performance, referred to as internal reinforcement in the self reinforcement model and thresholds in the threshold model. The two models also each track the need for the various task types through stimulus levels.

Plowright and Plowright [25] described the probability to perform a task with Equation 19. The probability of task performance P is influenced by internal reinforcement I, a constant K and the external stimulus E.

$$P = 1 - e^{-IKE} \tag{19}$$

Figure 5 further describes the workings of the algorithm. At every time step, E is incremented by a value γ_1 for every task type. Additionally, every machine has I decremented by a value γ_2 for every task type. For every idle machine the external stimulus of a random task type is selected at every step. Following this, the probability is calculated as in Equation 19, if P is larger than a randomly generated number, the job is accepted. After completion of a job, I is increased by a value γ_3 for that task type. This results in the four parameters, $\gamma_1, \gamma_2, \gamma_3$ and K to tune.

An inconsistency with the considered problem is the incrementing of E for every task type at every time step, which may lead to task types that rarely occur having a high external stimulus. This can easily be resolved by only incrementing E for those task types that have an unassigned task waiting. Implementing this is done by keeping track of E for individual jobs and therefore incrementing E for all unassigned jobs. In order to fit the model with the queues used in the problem, any machine with a queue that is not full may encounter

a random external stimulus. To avoid the same machines taking on tasks all the time the update loop for the machines is executed in random order, instead of following the same sequence at every time step as Plowright and Plowright [25] originally considered. Another sensible change for the considered problem is to increment I based on the last task in queue, rather than the one which was just completed. This reflects how Nouyan et al. [23] [24] and Meyyappan et al. [18] handled threshold updates to reduce the number of setups.

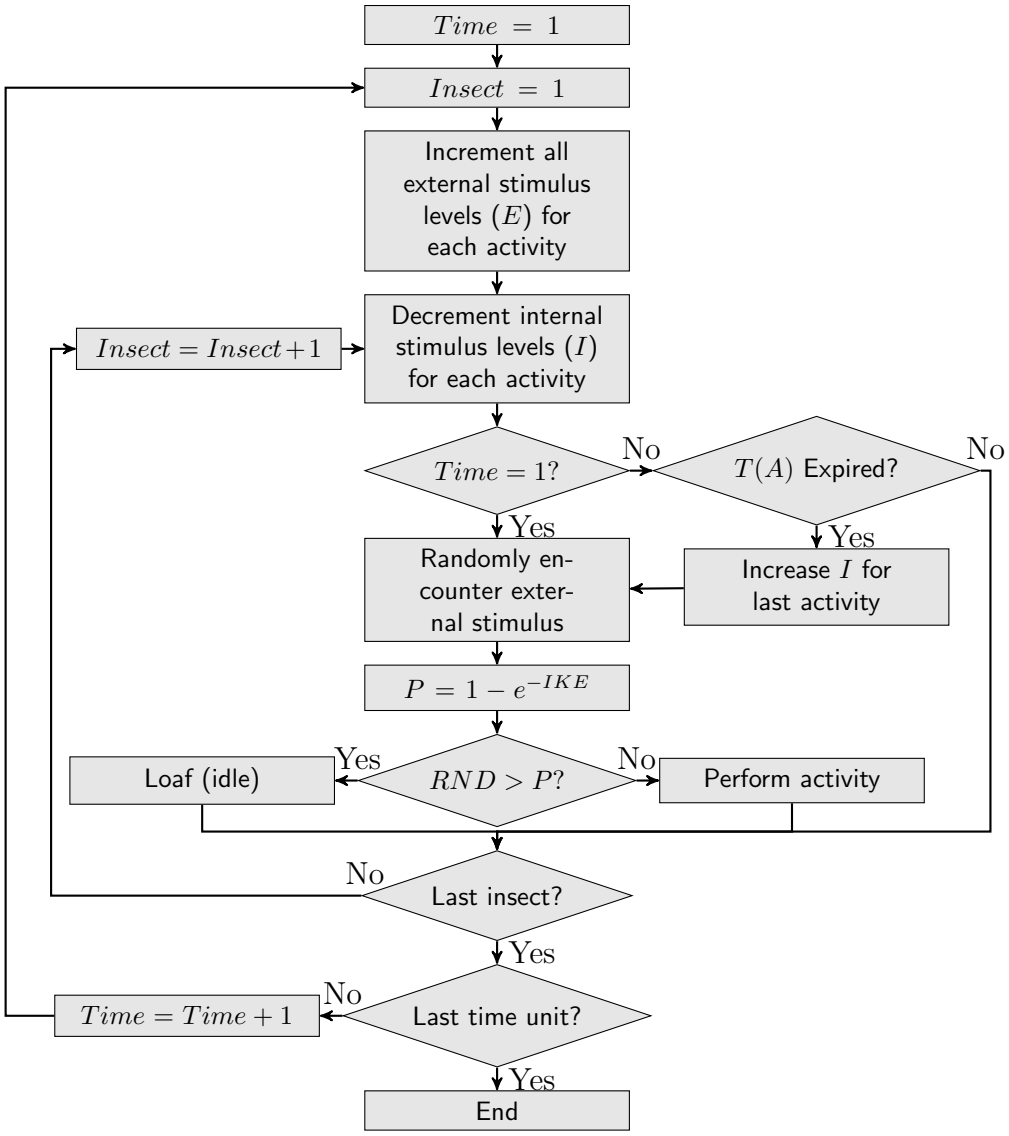


Figure 5: Flowchart of the self reinforcement model (reconstruction of the flowchart in Plowright and Plowright [25], including some minor alterations for clarity). $T(A)$ refers to the time T required for the execution of an activity A.

Comparing the Plowright and Plowright [25] model with those by Campos et al. [6], and Cicirello and Smith [8], the greatest difference is seen in task selection. Plowright and

Plowright [25] randomly show one task to a machine, while in the other two models all machines see all tasks. In Campos et al. [6] a machine does not bid when it is broken down, or when its queue is full. Cicirello and Smith [8] decide whether or not to bid based on a probability function that includes the stimulus and threshold, while they do not specify this, it is assumed they don't bid when a queue is full either. Plowright and Plowright [25] then decide whether or not to perform the task based on threshold and stimulus. The other two allow all machines that bid to compete, based on threshold and queue (Campos et al. [6]), and just queue (Cicirello and Smith [8]).

5.5.2 Foraging for work

Tofts [32] described the foraging for work (FFW) model, which simulates the spatial situation in an ant nest. Depending on their location in the nest, worker ants come across different tasks. The model sequentially orders the considered task types and allows workers to move back and forth over this sequence. Ants hatch deep in the nest, which relates to starting at the beginning of the sequence. When there is a task available at the current position in the sequence the worker will take this job. When the job is finished, the worker checks if there is another task available in the current position (i.e. is there a task of the same type as the last one?), as long as this is true, the worker keeps performing tasks of this type. When there is no task available at the current location, the worker moves either to the next, or the previous task in the sequence. If there is a task available at the new position the worker starts working there, otherwise it keeps moving until it arrives at a position where it finds work.

Task types can be related to colours and workers to machines to translate the situation to the truck painting problem. Since colours have no sequential relation a few changes are sensible. First off, the last task type will lead back to the first such that there are no begin or end positions. Second, instead of starting at one specific task type, each machine will be randomly assigned a starting position. Active (painting or in setup) machines with space in their queue will first check for jobs in their location and if they see one add it to their queue. Then idle machines look for a job in their current location. Finally, remaining idle machines, that did not find a job in their current location, may move and look for a job elsewhere. A maximum number of moves may be used, regulated by the only parameter for this algorithm, the step size. These idle machines will move in a randomly selected direction and take on the first job they see. If no job is found after moving the maximum number of steps, the machine remains in this new position and try again during the next time step. Only when a task in a new location is found a setup will take place, this is to avoid unnecessary setups while looking for a new job. For all of these checks, machines will be selected in a new random order every time step.

Note that these choices may not be optimal and will certainly have their advantages and disadvantages. Future work could investigate other options if foraging for work proves viable in some application. A line sequence could be maintained without connecting the start to the end, or all task types may be considered in a complete graph. Using a more sophisticated method for deciding the move direction might also be considered. For example, taking into

account the movement directions of other machines could be beneficial.

Foraging for work approaches task distribution in a vastly different way than the other models. The spatial distribution of work is an interesting concept which the other models don't employ. For the use in this problem a spatial distribution is however artificially introduced, making it questionable whether it would have a positive effect. A discrepancy with many application areas is that in the model machines move to tasks, as opposed to a factory setting where tasks would generally move to machines. On the other hand this is very much in line with areas such as swarm robotics.

It is difficult to say how the performance of FFW would compare to the other models. The consistent performance of the same task until it is no longer available likely has a positive effect. However, outlying tasks in the spatial distribution that occur infrequently may have to wait a while until a machine reaches them. It is not clear how that would compare with waiting for a change in a threshold until a machine takes up an infrequent task.

5.5.3 Fixed threshold

Bonabeau et al. [4] described the fixed threshold model, which preceded the reinforced threshold model of Théraulaz et al. [31] which was used by Campos et al. [6], and Cicirello and Smith [8]. This did not yet contain update rules for the thresholds. As such, the diversity of the initially generated thresholds will largely decide the performance abilities of the algorithm.

What follows is the implementation as considered in this work's experiments. Thresholds are generated randomly in the range $[\theta_{min}, \theta_{max}]$, this should usually result in sufficient diversity in task preferences when the task distribution is (close to) uniform. Situations where some task types occur significantly more often than others would probably be more difficult for this approach. The stimulus is taken for individual tasks based on the time they have been waiting to be assigned to a machine, just like in R-Wasps [8]. Determining the probability to bid for a machine also works in the same manner as in the R-Wasps algorithm of Cicirello and Smith [8], using Equation 9 (which they got from Bonabeau et al. [4]). Following that, the force is calculated as proposed by Nouyan et al. [23] [24] and Meyyappan et al. [18]. Finally, a winner is selected through a tournament (if needed) of dominance contests, again as described for R-Wasps [8]. For this model the only parameters would then be θ_{min} and θ_{max}, where it makes sense to set θ_{min} to zero, leaving just one parameter to tune.

Without update rules this model is less computationally intensive than those of Campos et al. [6], and Cicirello and Smith [8], which could be advantageous in situations with large numbers of tasks and machines. The question remains whether the difference in computational effort is significant enough to consider the fixed threshold model as an option, despite its most probably less optimal scheduling.

6 Proposed algorithm

A variety of sensible ideas are involved in the various algorithms previously discussed, combining a number of these could be beneficial. For this proposed algorithm the stimulus and threshold reinforcement combination employed by most of the discussed algorithms is used.

This proposed algorithm works in four steps, first machines decide whether or not to bid, then those that do determine the height of their bid, a winner is selected from the bidders and finally thresholds are updated. The stimulus of a job S_j is equal to one plus the sum of the time (in seconds) that it has been awaiting assignment, just like in the R-Wasps model of Cicirello and Smith [8].

Part	Origin	Contribution of the proposed algorithm
Stimulus	Cicirello and Smith [8]	Unchanged
Competitors	Original	Allow broken machines to (consider making a) bid
P(bid)	Cicirello and Smith [8], Campos et al. [6], and Plowright and Plowright [25]	Probability to bid includes the job type
Bid height	Cicirello and Smith [8], and Campos et al. [6]	Compete (as opposed to simply taking the highest) based on both the threshold and ΔT (the time until painting starts)
ΔT	Cicirello and Smith [8], Nouyan et al. [23] [24], Meyyappan et al. [18] and original	Include the remaining down time in the time until painting starts
P(win)	Nouyan [23]	Unchanged
Updates	Cicirello and Smith [8], and Nouyan et al. [23] [24]	Unchanged

Table 3: Summary of changes.

The chance to bid is defined in Equation 20, based on Equation 9 from R-Wasps. It works the same, with the addition of the multiplier $c_{i,k}$, which evaluates to one when the colour of the last truck in queue of machine k is identical to that of truck i, or zero when they differ. This is the same as in the bid function Campos et al. [6] defined for the market-based approach. By including this colour in the equation, machines may avoid switching back an forth between a number of colours with low thresholds. This serves as an alternative to the approach used in Model 2 considered by Meyyappan et al. [18], which excluded the memory of previous tasks stored in the thresholds from the algorithm. In relation to nature this corresponds with the probability of an insect actually coming across a task. Seeing a task of the same type as the one currently being performed is more likely than coming across a task of an other type since tasks of the same type would generally occur in the same area. The chance to still observe opportunities for other task types relates to overlap in the areas in which multiple task types occur. Considering the problem definition, it makes sense to have the same overlap probability for all task types, since the cost to switch between them is also equal. Nature differs in this in that different in nest tasks would have a lower cost

(travelling time in this case) for switching between them than a switch from an in nest task to an out of nest task.

$$P_k(bid) = \frac{S_i^2}{(S_i^2 + \theta_{i,k}^2) \cdot (2 - c_{i,k})} \tag{20}$$

Step two is to determine a bid by which machines that decide to participate will compete. For this and the next step the force concept described by Cicirello and Smith [8] is employed. Therefore, as explained for their algorithm, the most fit individual is the one with the lowest force. The force F_k is computed using Equation 21 which works the same as the ant-based bid function of Campos et al. [6], minus the division. Contrary to the force from Cicirello and Smith [8], this includes the threshold θ_{k,c_i} of machine k for offered colour c of truck i. Like in Campos et al. [6], the α and β parameters serve as weights.

$$F_k = \alpha \cdot \theta_{k,c_i}^2 + \Delta T_k^{2 \cdot \beta} \tag{21}$$

The waiting time ΔT of a machine k here consists of four parts as in Equation 22. These are $T_{k,p}$ and $T_{k,s}$, respectively the processing time and setup time of all jobs in queue. Note that the queue is here defined as all jobs assigned to this machine k and thus includes the job that may currently be in processing. Next $T_{k,j}$ is any setup time that may be required for the offered job as proposed by both Nouyan et al. [23] [24] and Meyyappan et al. [18].

$$\Delta T_k = T_{k,p} + T_{k,s} + T_{k,j} + T_{k,d} \tag{22}$$

Finally there is the $T_{k,d}$ term, this refers to any down time the machine may currently be going through. This is part of a proposed change where broken machines are allowed to compete for jobs. Note that machines that are already broken are still included in the selection procedure of the break down probability and a such may have their down time lengthened. Although this will rarely happen, in this manner a reasonable representation of potential reassessments of the repair time is included. Due to the inclusion of this term a broken machine is however not likely to win a job, unless the down time is almost over. The aim here is to allow machines with potential short or even empty queues to continue processing quickly after being back in operation.

In the third step one of the machines is selected to take on the job using Equation 23, which is equivalent to the one used by Nouyan [23].

$$P_k(F_1, ..., F_n) = \frac{\frac{1}{F_k^2}}{\sum_{i=1}^{n} \frac{1}{F_i^2}} \tag{23}$$

The final fourth step contains the threshold updates, which will be done every time step. Time steps will coincide with the release time of new jobs, so if a new job is released every minute there will be a time step every minute. Updates consist of the same rules as in Nouyan et al. [23] [24]. Starting with the standard situation the threshold for the last colour in queue is decreased (Equation 24) as proposed by both Nouyan et al. [23] [24] and

Meyyappn et al. [18], and the other thresholds are increased (Equation 25) like in most of the discussed algorithms.

$$\theta_{k,i} = \theta_{k,i} - \delta_1 \tag{24}$$

$$\theta_{k,i} = \theta_{k,i} + \delta_2 \tag{25}$$

Next in order to promote idle machines to go back to work they have all their thresholds decreased as in Equation 26, which works the same as described by Cicirello and Smith [8]. Jobs for which there are no bidders at all reduce the threshold of all machines as in Equation 27. Additionally, idle machines that don't bid when a job is offered reduce the threshold for the corresponding job type by γ_2 (Equation 28). Where the last two rules correspond to respectively the no bid for a created job (BCJ) and idle machine does not compete (IMC) rules proposed by Nouyan et al. [23] [24].

$$\theta_{k,i} = \theta_{k,i} - \delta_3^t \tag{26}$$

$$\theta_{k,i} = \theta_{k,i} - \gamma_1 \tag{27}$$

$$\theta_{k,i} = \theta_{k,i} - \gamma_2 \tag{28}$$

7 Experiments

This section deals with the experimental setup considered in this work. First off, the experimental setup is described. This includes the used default values for all problem parameters, some experimental situations may specify an alternative value for one or more of these problem parameters. Secondly, some implementation details that may have some impact on the measured results are specified. In the third subsection, the considered measurements are detailed. Fourth, a description of the various considered problem situations is included. Lastly, the parameter values used for each algorithm are listed.

7.1 Experimental setup

The problem parameters considered by Campos et al. [6] will be used as basis for the experimental setup. Whenever there is a reason to deviate from this or something was not specified the problem parameters as in Cicirello and Smith [8] will serve as second option. When neither of these suffices some value that seems sensible is selected.

In order to measure throughput the runtime and duration of production will deviate from Campos et al. [6] who measured the makespan instead. As in Cicirello and Smith [8] simulation will take 1000 minutes with trucks being produced at a rate of one per minute and requiring a paint time of three minutes. This leads to a theoretical optimal throughput of 998 trucks (all trucks released from t_0 to t_{997} have a chance of completing their three minute paint time). Although even with optimal assignment, trucks released toward the end may have to deal with flushes or break downs, making this optimum an unrealistic target. Each truck will have to be painted in one of twenty colours, selected uniform randomly. Every minute there is a 0.05 probability for a randomly selected machine to break down for a uniform randomly selected number of minutes from $[1, ..., 20]$. Changing the colour a machine paints requires a flush with a duration of three minutes. Campos et al. [6] considered cases with between six and fifteen booths. They mention that when using eight booths or more, storage is rarely needed, therefore eight booths will be used. Queues with space for five trucks are kept in line with the description from Campos et al. [6]. In addition to there being no reason to change this, changes would also invalidate the comment about eight painting booths. The initial colour of every machine is chosen uniform randomly as in Cicirello and Smith [8], since Campos et al. [6] did not specify how they did this. Thresholds are also initialised as considered by Cicirello and Smith [8], for the same reason. The threshold belonging to the initial colour of a machine is set to θ_{min}. All other thresholds are drawn uniformly at random from the interval $[\frac{\theta_{max}}{2}, \theta_{max}]$. To allow all algorithms to work as intended there is no limit on the size of the storage space, it is however kept track of as detailed later in the measurements subsection.

7.2 Implementation details

Some minor implementation details are not strictly part of the algorithms, but may have a (minor) impact on the outcome of experiments. As such, these implementation details are

included here to provide an accurate representation of the experimental environment. This is not an exhaustive list, there may well be more details impacting the outcome.

When a machine needs to change colour in order to start the next job in queue this job waits in the queue until the setup is over. This keeps some queue space occupied for longer than may be strictly necessary for the simulation. Doing this may however provide a more accurate representation of a real painting booth. During the switch from one colour to another moving parts and possible paint splashing in such a machine could make it inconvenient to have a truck in the booth during this process. Of course, this doesn't necessarily provide an accurate representation of all real life systems, it is just one possible representation which is sufficient for the purpose of this simulation.

7.3 Measurements

Optimisation for a minimal total setup time (TST) is the primary goal. This should however be paired with a high throughput ($\sum U_j$). Ideally this would also be combined with a low average flow (or cycle) time (F) to allow optimal scheduling in other steps of the production process. Only optimising for the total setup time is not an option. First off, not painting anything at all leads to the, 'optimum', zero second setup time. Second, assigning all booths a different colour at the start and painting only those colours until no more jobs of those types arrive before switching to paint other colours would also achieve a low number of setups; along with an extreme delay in the processing of trucks requiring the non-starting colours. Clearly, neither of these situations are desired. As such, the goal can be summarised as minimising the total setup time while maintaining reasonable results for the throughput and average flow time.

In addition to those three measurements, another two are kept in mind as they would ideally also be low, but are considered less important. This concerns the average queue length (average queue length of all machines averaged over the number of measurements, e.g. one per time step) and the average used storage space (storage space in use averaged over the number of measurements). As long as these two measures do not reach too extreme values they are considered acceptable. Explicitly optimising for these can be considered for further work.

Analysis of the minimum, maximum, mean and standard deviation of the measurements will provide insight into the performance of the algorithms in different areas.

7.4 Considered situations

A total of six problem situations are considered as listed below.

1. Standard situation as resulting form the default problem parameters described in the experimental setup
2. As above, but with a colour distribution of colour occurrences: one colour 70%, one colour 15%, one colour 7%, one colour 4% and the remaining sixteen colours 0.25%
3. As above, but now two trucks are produced every minute

4. As above, but now the probability of a randomly selected booth going down is 0.25 per minute
5. Standard situation, but with no break downs
6. Standard situation, but with setup times of ten minutes

The first four are the experiments as described by Campos et al. [6], the only change being the different standard situation. Situations five and six are based on the problems presented by Cicirello and Smith [8]. Situations one through four present a (relatively) simple base situation with increasingly complex alterations for the machines to deal with. The fifth situation mimics the standard situation considered by Cicirello and Smith [8] which did not include machine break downs. This can be interesting for situations other than truck painting where break downs are unlikely, for example in software simulations. Finally, longer setup times are considered to analyse how the various algorithms handle this. All of these are executed for 1000 replications.

Problem	Characteristics
1	A standard situation with an unrealistic uniform colour distribution
2	A standard situation with a realistic colour distribution
3	A heavy load situation with a realistic colour distribution and twice as many trucks requiring painting
4	A heavy load situation with a realistic colour distribution, twice the trucks to be painted and larger machine break down probability
5	A standard situation with an unrealistic uniform colour distribution, without break downs
6	A standard situation with an unrealistic uniform colour distribution, with long setup times

Table 4: Summary of the considered problem situations.

Dynamic colour distributions, both abruptly changing as considered by Cicirello and Smith [8] and gradually changing, have not been included due to time constraints. They are certainly interesting to consider investigating in further work though.

Based on good preliminary results achieved with the foraging for work algorithm it was analysed further on the first two problems using step sizes from one through ten.

7.5 Algorithm parameters

Table 5 shows the parameters used to compare the algorithms on the situations described in the previous subsection. Note that these parameters were not optimised for this problem, therefore they may not provide an optimal representation of the algorithms. In particular the algorithms using original parameters may not show a representative performance. Also note that the proposed algorithm (KB) uses parameters from both Campos et al. [6] and Nouyan [23].

For comparison purposes a random approach was also included in the experiments. It is implemented such that newly produced jobs are immediately assigned to a randomly selected

Algorithm	Parameters			Parameter origin
Random	-			-
Market-based (MBC)	$P = 91.8106$	$C = 1791.99$	$L = 4.09477$	Campos et al. [6]
Ant-based (ABC)	$\alpha = 617.188$	$\beta = 4.66797$	$\xi = 7.85156$	Campos et al. [6]
	$\phi = 17.7344$	$\theta_{max} = 39.6875$	$\theta_{min} = 5.50781$	
R-Wasps	$\delta_1 = 100.0$	$\delta_2 = 10.0$	$\delta_3 = 1.05$	Cicirello and Smith [8]
	$\theta_{max} = 10,000.0$	$\theta_{min} = 1.0$		
Ant task allocation (ATA)	$\delta_1 = 14.0$	$\delta_2 = 14.0$	$\delta_3 = 1.01$	Nouyan [23]
	$\gamma_1 = 50.0$	$\gamma_2 = 50.0$	$\theta_{max} = 2000.0$	
	$\theta_{min} = 1.0$			
Self-Reinforcement (SRM)	$\gamma_1 = 0.5$	$\gamma_2 = 0.25$	$\gamma_3 = 0.5$	Original
	$K = 0.5$	$\theta_{max} = 10,000.0$	$\theta_{min} = 1.0$	
Foraging for work (FFW)	step size = 2			Original
Fixed threshold (FT)	$\theta_{max} = 100.0$	$\theta_{min} = 1.0$		Original
Proposed (KB)	$\alpha = 617.188$	$\beta = 4.66797$	$\xi = 7.85156$	Campos et al. [6]
	$\phi = 17.7344$	$\theta_{max} = 39.6875$	$\theta_{min} = 5.50781$	
	$\delta_3 = 1.01$	$\gamma_1 = 50.0$	$\gamma_2 = 50.0$	Nouyan [23]

Table 5: Used parameters in experimentation per algorithm.

machine that has space in its queue. After that it attempts to assign any jobs that may be in storage. Jobs are added to storage whenever no machines with queue space exist, or the only machines with queue space are broken.

Parameter optimisation is not considered here as it is a significant problem of its own right. Simple test showed that even averages over 100 replications would have notable variations when done twice for the same parameters. Therefore attempting to optimise even when using simple methods would be a time consuming matter. This is most certainly something to look at in further work.

8 Results

In this section the results are shown and analysed, with each problem being handled in their own subsection. A number of plots have many outliers with the exact same value. This primarily happens with the throughput and is a result of these values simply being far more likely to come out exactly the same compared to an averaged number with decimals.

8.1 Experiment 1

In terms of throughput the market and ant-based approaches as described by Campos et al. [6] perform similarly well. Next, KB does pretty well in this area compared to R-Wasps and ATA which are largely similar approaches. The performance of FT is also notable considering it does not adapt to changing situations to the extend most other algorithms do.

Clearly FFW performs exceptionally well compared to all other approaches in using a low number of setups. Unlike the other approaches, FFW will never add a job of a different type to the queue of an active machine. As such, it is much less likely to do unnecessary colour changes. Followed by that, KB has an advantage of around 100 setups on the rest. Those others perform similarly well to each other, having relatively small differences in the used setups. The exception to this is ABC, which somewhat surprisingly is worse than FT. It could be noted that the parameters proposed by Campos et al. [6] were used, which were not optimised solely for the number of setups. This is however also the case for the MBC approach, which does not experience this problem.

MBC and ABC both achieve nice and low flow times, with MBC doing ever so slightly better than ABC. Again KB is the best of the rest. Not surprisingly the R-Wasps and ATA approaches result in longer flow times. In this uniform case however this has not resulted in much better performance on the setup front compared to MBC and to a lesser degree ABC. Finally FFW and FT are somewhere in between, considering the limits of FT and the simplicity of FFW these are however not unexpected performances.

In terms of average queue length MBC, ABC and FFW are the best performers. KB does similarly well, but has a slightly larger spread in its results. The remaining algorithms seem to do reasonably well in not assigning jobs of other types to a queue unless it gets busy, resulting in no really long queues for any of the algorithms. Only SRM is a poor performer, which seems to be the case across all measures. This is likely due to a number of random factors being used in its implementation, along with the likely poor parameters.

The clear downside of FFW is shown here, it uses comparatively much storage space to achieve its low number of setups. As with the other measures, KB does not perform the best, but remains in an acceptable margin, this time together with FT. R-Wasps and ATA do explicitly use storage space, but by no means as excessively as FFW.

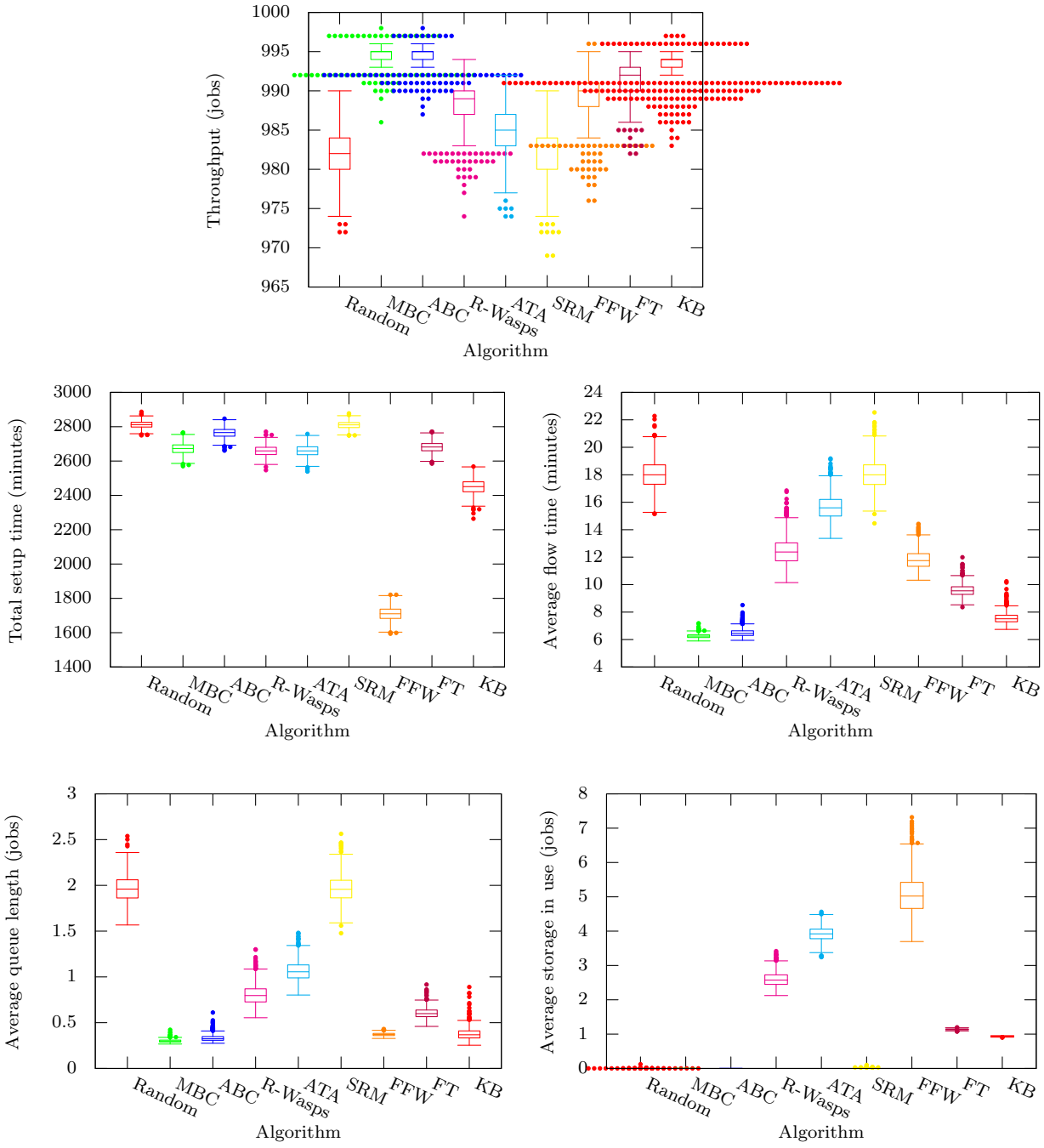


Figure 6: Experiment 1: Twenty job types with uniform random appearance rates.

8.2 Experiment 1 - Foraging for work

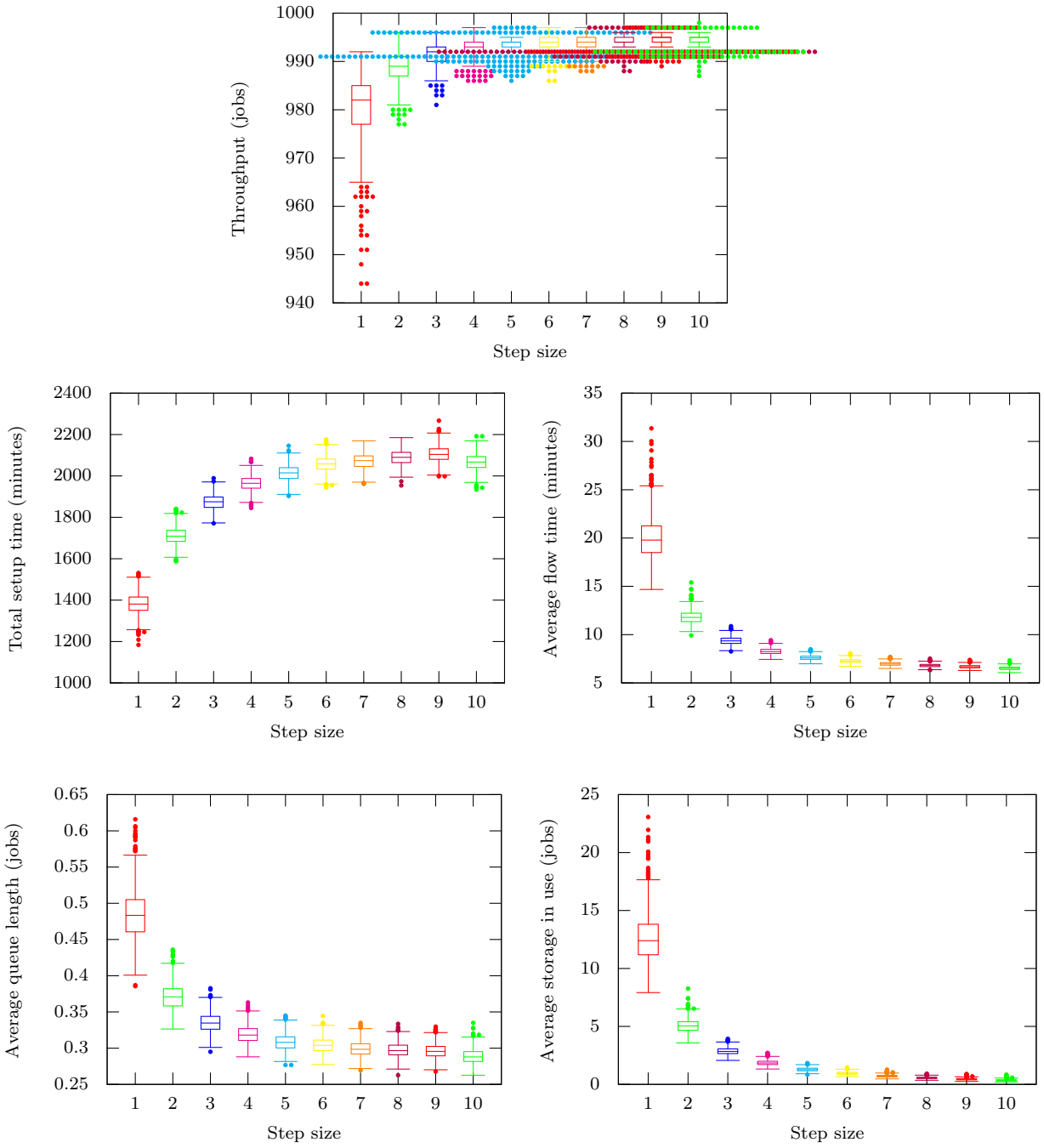


Figure 7: Experiment 1: Foraging for work with different step sizes on twenty job types with uniform random appearance rates.

Throughput seems to increase with greater step sizes, although the difference quickly becomes negligible. The required number of setups is by far the lowest with the smallest step size.

This can easily be explained by machines rarely being able to move far from their current colour. As such, even when they move they have a good chance to come back to their old position without finding a job elsewhere. A reduction of 100 setups compared to a step size just one greater is a lot. However, it remains a question whether this weights up to the delay in production reflected by the the throughput. Even so, step sizes of two or three still use quite a few setups less than even larger step sizes, while they cause a much smaller delay.

Looking at the average flow times, it can be seen that, even with a step size of one, production will on average rarely be delayed more than half an hour. Given operation costs and the previous results, this makes it possible to calculate which number of time steps leads to the lowest cost per painted truck.

Average queue lengths show that for the foraging for work algorithm queues with five slots are quite excessive, at least with this colour distribution. Environments with this or similar colour distributions could therefore save some space by reducing their queue sizes.

While this algorithm seems to require at least some storage in any situation it can be kept in reasonable bounds when using step sizes of two or greater. Considering a situation where queues are allowed to have a length of three, saving sixteen (two times eight machines) spaces compared to this situation, assigning a storage area that allows for ten trucks still leads to a reduction in the required area.

8.3 Experiment 2

Compared to the first experiment, the throughput performances are much closer with this more realistic colour distribution. MBC and ABC are still the close first and second, followed closely by R-Wasps and ATA, with FT and KB right behind those. By comparison FFW is not really a competitor here, although it does not actually do extremely bad, it is just well behind the rest on this front. SRM keeps performing similarly to random, here and in the other measures.

Unlike in the uniform case, the FFW algorithm now has competition on the total setup time front, ATA and especially KB come close, though they still leave a reasonable gap. MBC is the best of the rest, but well behind the top three and as might be expected FT is quite bad. The surprise is that both ABC and R-Wasps perform very poorly. As mentioned for the first experiment, the ABC parameters were not solely optimised for the setup time. Being one of the first approaches to this problem, it may also simply not be as good as more sophisticated algorithms such as ATA. For R-Wasps this is likely in part a tuning problem as well. Cicirello and Smith [8] made comparisons on uniform and close to uniform colour distributions and as such the parameters may be more in line which such cases. Then again, as they did not make a comparison with a more realistic distribution it is entirely possible the algorithm simply doesn't handle those very well. After all, on their problems R-Wasps was at least able to improve on ABC (with parameters tuned for the setup time by Cicirello and Smith [8]) by a margin, whereas here it is a clear step behind.

While the average flow time wasn't very good in the uniform case either, FFW is now clearly the worst. The rest of the algorithms (ignoring SRM) all do quite well, being mostly in the four to six minute range. Of those, FT does relatively poorly, averaging just above six

minutes, while MBC followed by ABC perform best having their average flow time around the four minute mark.

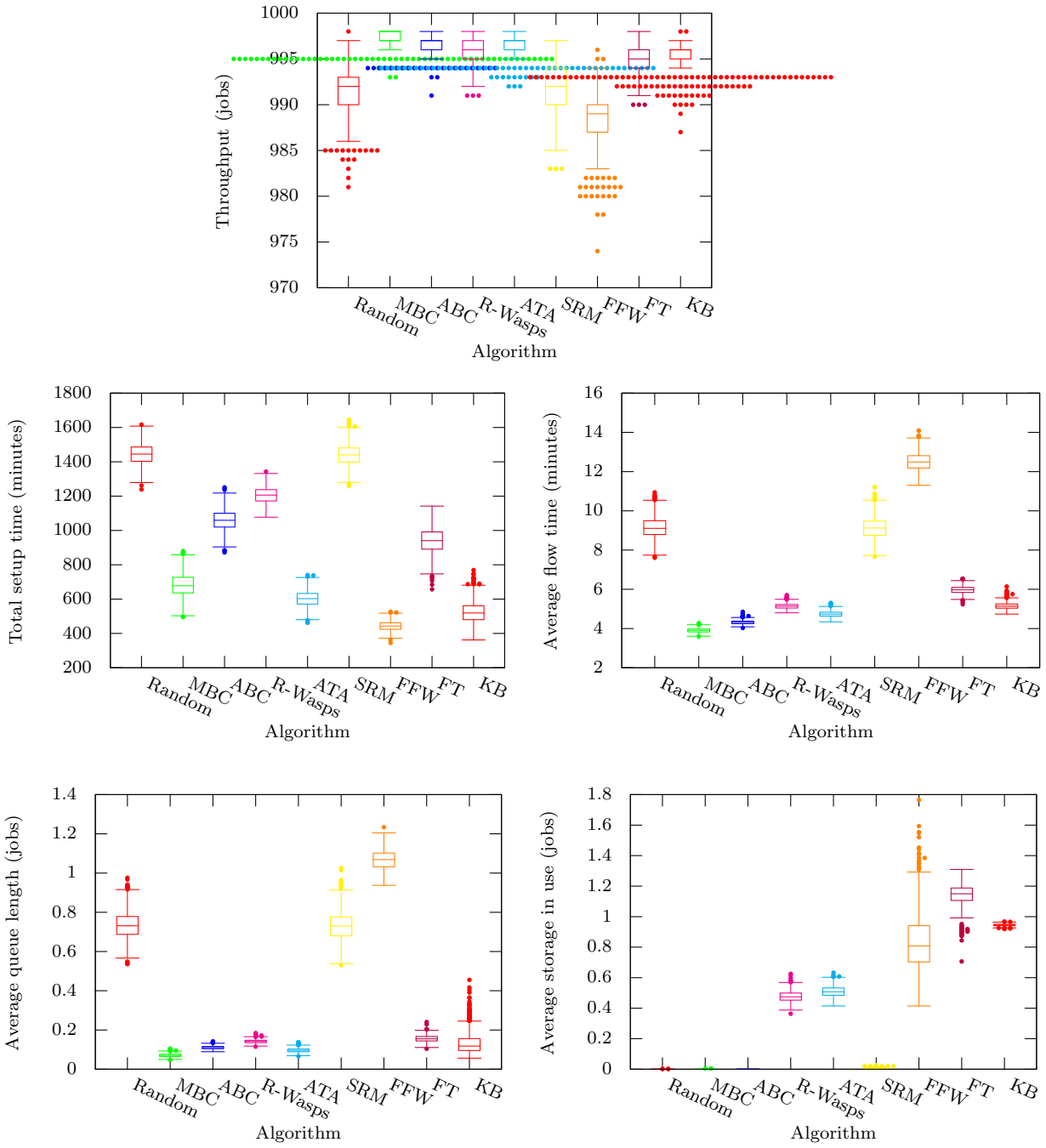


Figure 8: Experiment 2: Twenty job types with appearance rates of: one 70%, one 15%, one 7%, one 4% and sixteen 0.25%.

Considering these three metrics, KB seems like the best choice for this more realistic colour distribution. It is second best in terms of the total setup time and also performs well both for the throughput and the average flow time. Only FFW outperforms KB on the total setup time, but it is clearly worse in terms of both the throughput and the flow time.

Also for the average queue length, the differences are mostly small, with the same algorithms (in the competitive range) doing best and worst. Here too FFW is an outlier and by far the worst, despite having been one of the competitive algorithms in the uniform situation. It should however be noted that its queue sizes still average just above one and its queue usage cannot at all be considered excessive.

In terms of the average storage use, not that much changed compared to the uniform case, primarily the numbers became smaller. Notable is that FT and KB use around the same amount of storage as in the uniform case, but are now among the worst performers, along with FFW. Then, R-Wasps and ATA are in between those poor performers and the rest that rarely, if at all, use storage.

8.4 Experiment 2 - Foraging for work

Beyond a step size of one, all other step sizes perform reasonably similarly in terms of throughput. The differences between step sizes are far less clear with this more realistic colour distribution.

Unexpectedly, FFW with a step size of ten outperforms all other step sizes, including a step size of one on the front of setup time. Is it somehow more likely to arrive back at its position from the previous time step than other step sizes? There is no bias to movement in one direction, the direction is randomly selected every time a machine starts moving (when it couldn't find a job in its current position), but not for every step of this movement. The machine looks for a job only in this direction, it does not check the other side if it fails here. This means that, when a step size of greater or equal to half of the job types is considered (in this case ten or greater), a machine can arrive back at the position from the previous time step regardless of the movement direction. This is due to the connection from the last job type to the first job type considered for the used implementation of FFW. For smaller step sizes this is only possible in half of the cases. It should be noted that this will only happen when there are no jobs on the route to the type of the previous time step. Which explains why the effect is clearly visible in this experiment, while being barely of note in Experiment 1. Fluctuations in performance of step sizes between two and nine may be explained by two factors. First, when using a step size of five, moving in one direction for two time steps leaves the worker in a position where it has two ways to get back to the original point using another two time steps, since it doesn't matter whether it goes forward or backward. Considering a step size of two, again going in one direction for two time steps, only going back the same way for two time steps will reach the original point. Generally, a larger step size has a greater or equal number of possible paths back to its original position as any smaller step size. Second, however, the greater the step size, the greater the likelihood of coming across a job opportunity along this path before arriving back at the original position. Due to these two factors it is not immediately clear beforehand which step size in the two to nine range

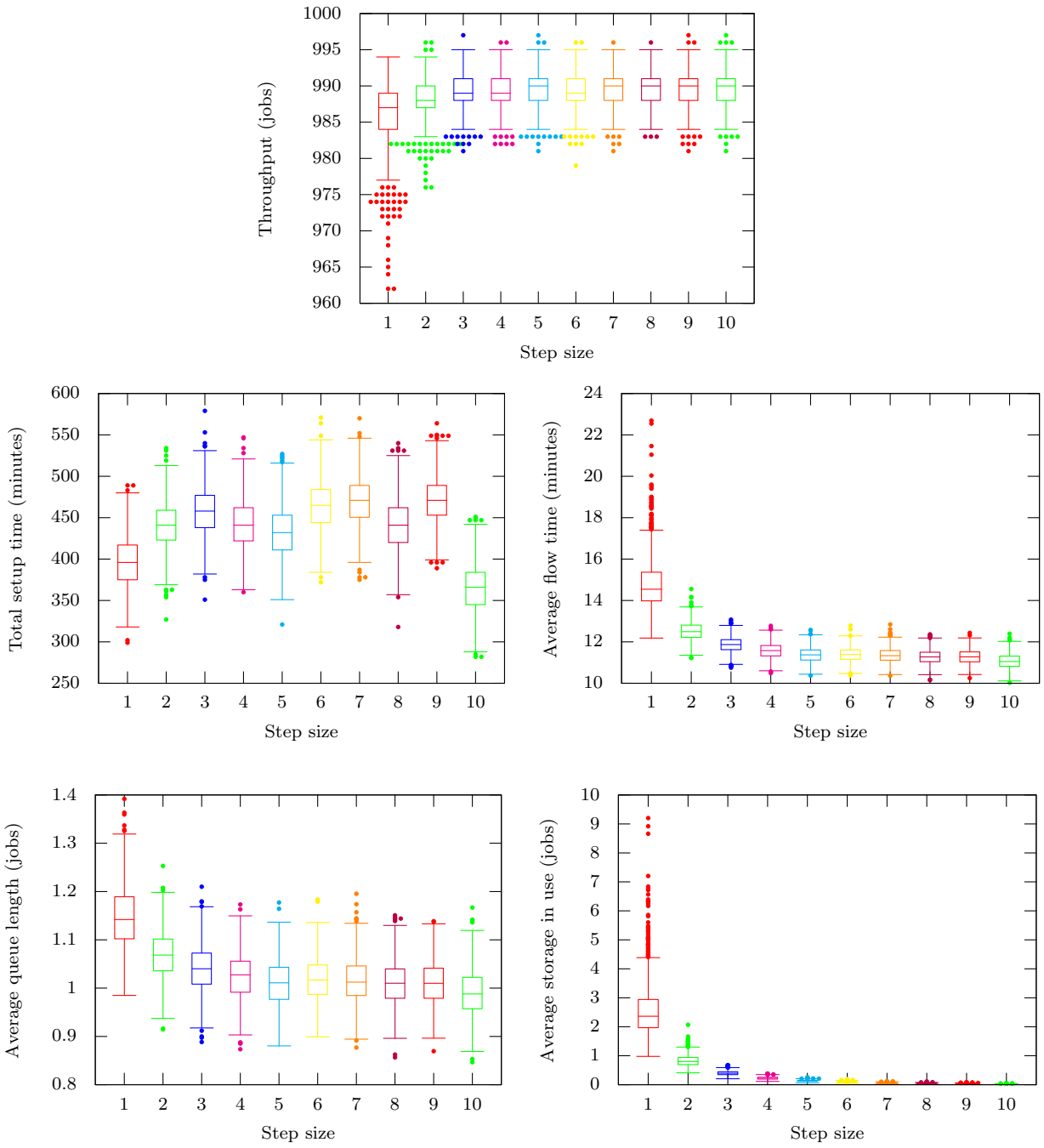


Figure 9: Experiment 2: Foraging for work with different step sizes on twenty job types with appearance rates of: one 70%, one 15%, one 7%, one 4% and sixteen 0.25%.

would require the lowest setup time. Additionally it should be noted that job opportunities are not fairly distributed in this experiment. On the type range from zero through nineteen the higher probabilities are all in consecutive positions, type zero being the 70% appearance

rate, type one 15%, etcetera. Therefore, moving in the negative direction from point zero (back to nineteen) on average presents fewer job opportunities compared to moving in the positive direction, although it is not obvious if this influences performance since movement in both directions is equally probable.

The trend for the flow time is clear and expected, with greater step sizes having lower flow times. This is the logical effect of them being quicker in taking on newly appearing jobs in different locations, simply because they are faster in reaching them. It should be noted that flow time differences quickly become negligibly small from around step size four and up, allowing step size selection to be made on basis of other factors.

For the average queue length the trend is similarly clear, but less obvious in its origin. Lower step sizes are slower to take on a new job after running out of work. This gives machines that are painting frequent colours more time to add work to their queue, even with a full queue a slot may become available if it takes more than a couple of time steps for another machine to take the job. Contrary, higher step sizes will quickly take any job after going idle, while maintaining a bias towards active machines which get to check for a job before those that are idle. Clearly, the setups induced by quickly taking any job when idle are less significant to average queue length than maintaining (a few) long queues for frequent types.

Like with the average queue length, the storage use is also greatly impacted by quickly taking on any job with larger step sizes. Due to this, step sizes of four and greater hardly use any storage at all.

8.5 Experiment 3

Only FFW seems to be able to deal with double the number of jobs being produced. Beyond that, R-Wasps deals with the situation best, but is still hundreds of jobs behind on production. MBC has some outliers that do pretty well, but in most situations, like the rest of the algorithms, it is 400 or more jobs behind after the 1000 simulated minutes of the experiment.

The algorithms show the same ranking for the total setup time, indicating that efficient use of setups is essential to high volume throughput. A side note about how FFW achieves this is that it may be delaying jobs of uncommon types longer than usual. This may happen because machines will not switch task while there are jobs of the same type available, with continued production this is increasingly likely. As a result of this, the setup time may remain low, but there could be a negative effect in respect to the average flow time.

These effects to the average flow time are visible, suggesting that FFW does occasionally delay some jobs for quite a while. Except for SRM, the remaining algorithms all seem to have an upper bound on the average flow time around 30 minutes. SRM's performance is very poor in terms of average flow time, but there is a good explanation for this, aside from the complete lack of tuning. Only SRM offers a job randomly selected from all waiting jobs to the machines, all other approaches (including random) first offer jobs produced in this time step and then use a first in first out system for jobs in storage. Quite obviously, as soon as there are more than a few jobs in storage, this will have the devastating effect on the average flow time seen in the plot.

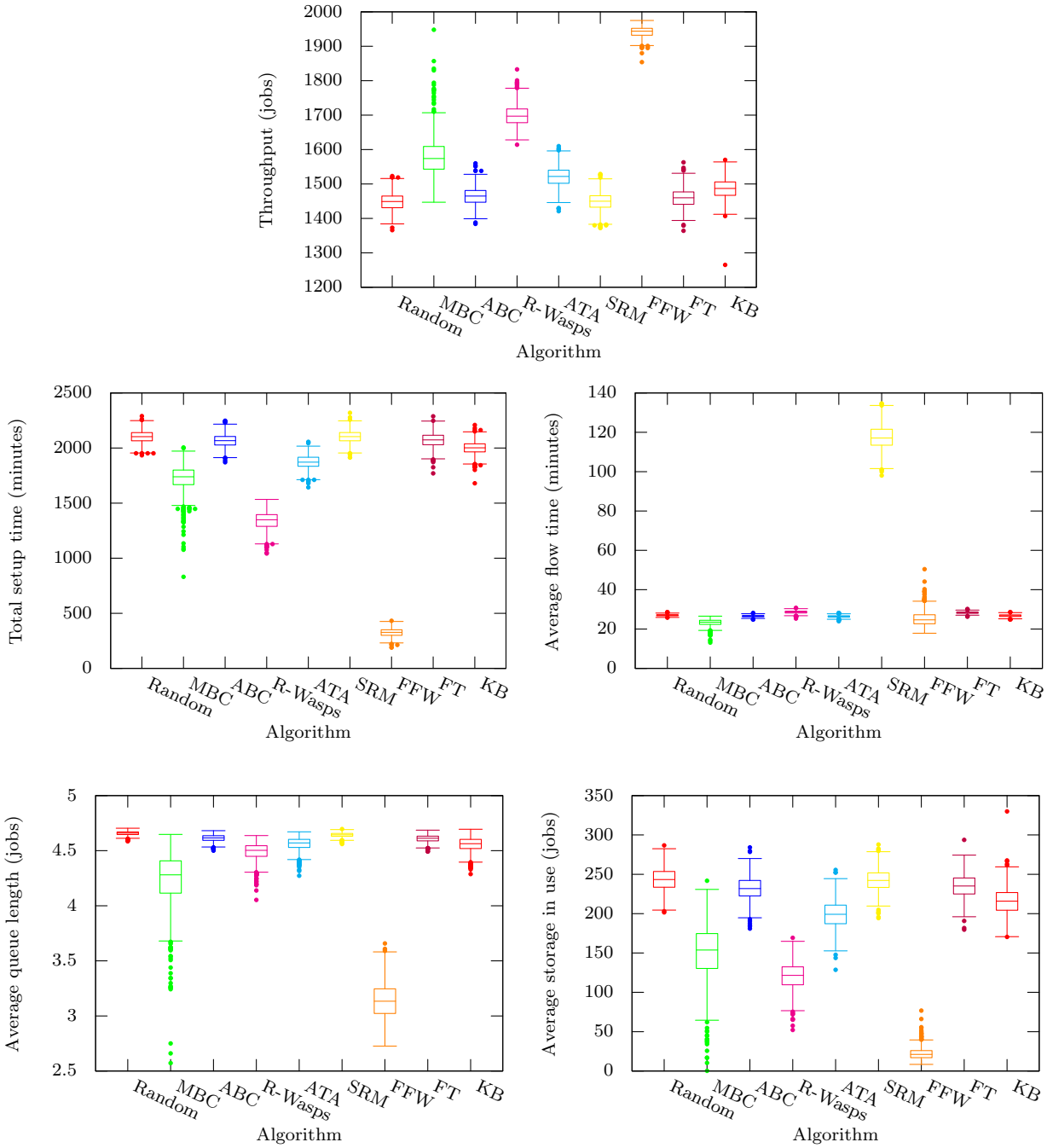


Figure 10: Experiment 3: Double the production, twenty job types with appearance rates of: one 70%, one 15%, one 7%, one 4% and sixteen 0.25%.

In addition to the average flow time, the average queue length shows that while FFW manages to deal with the situation, it wouldn't be able to handle much more. As might be expected in this situation, all other algorithms generally have very full queues.

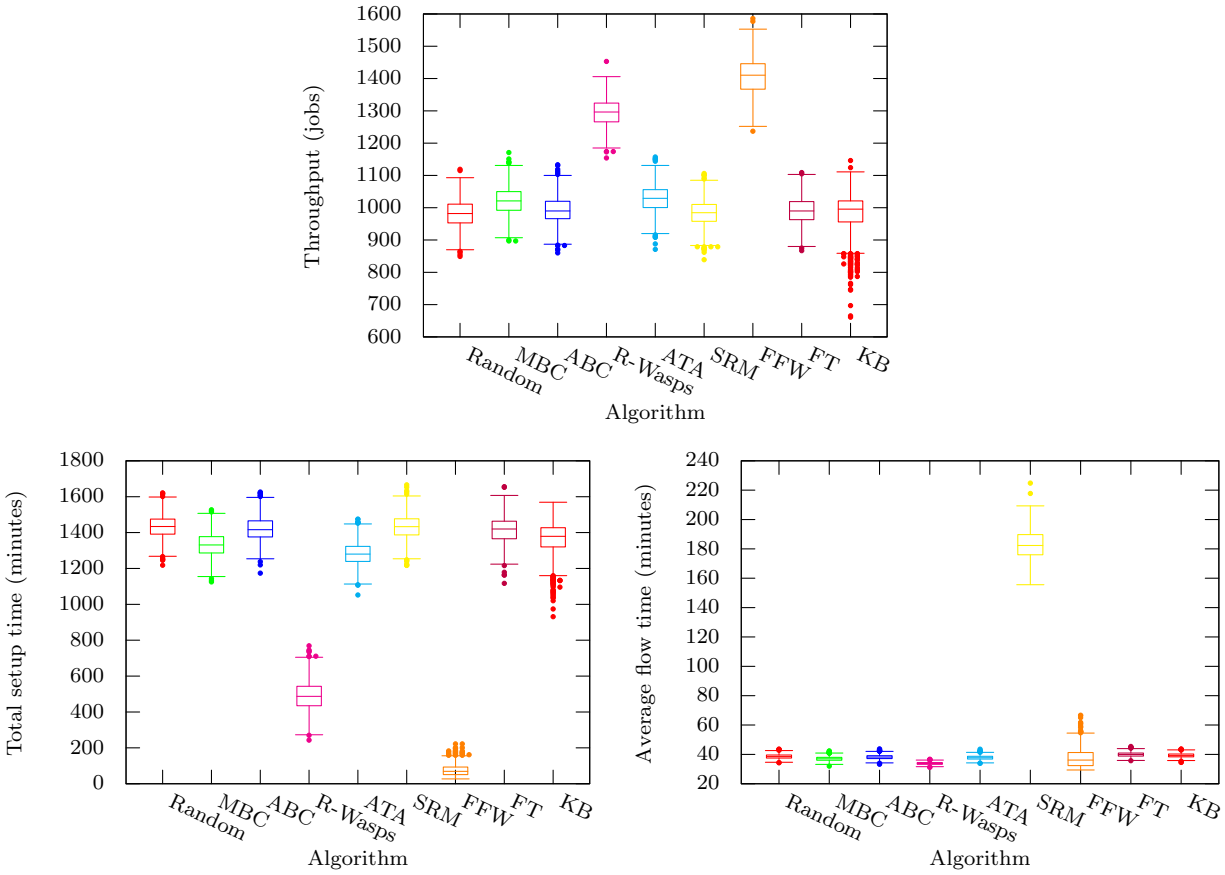
Storage usage is also as would be expected. Some outliers where FFW too has difficulty handling the production rate can be seen. Given how often it manages, it may be expected that in longer runs it may be able to recover from such situations, additional experimentation is required to ascertain this. MBC is the only other algorithm that has a few outliers with hardly any storage use, suggesting that it is sometimes able to deal with the situation for a while at least. This could mean that it is a viable approach in less extreme situations.

A final notable things is how SRM and random are not particularly worse than other approaches here (except for the average flow time). This may indicate that this is around the limit of how poor performance can get here.

8.6 Experiment 4

Seeing how FFW already showed some difficulty in the previous experiment, it is no surprise that with a higher break down probability added on it, like the rest of the algorithms before, is no longer able to process everything.

The total setup time, with how extremely low it is, now very clearly indicates that FFW simply isn't painting some job types. R-Wasps also manages an usually low total setup time,



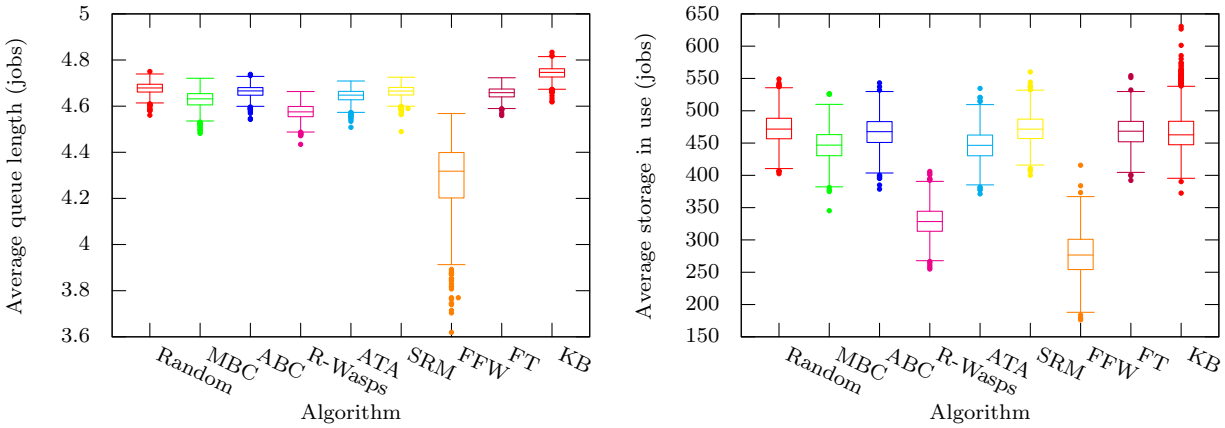


Figure 12: Experiment 4: Double the production, break down probability of 0.25 per minute, twenty job types with appearance rates of: one 70%, one 15%, one 7%, one 4% and sixteen 0.25%.

which likely has a similar reason. Machines become so extremely specialised in a single job type that they are highly unlikely to switch.

In terms of average flow time the same comments apply as for the previous experiment. FFW's average queue length suggest that it tends to be able to handle the situation for a short while, until the system ultimately overflows, as it does with all other algorithms. None of the average storage usage is surprising. FFW and R-Wasps were able to handle the situation best, but still require excessively large storage.

8.7 Experiment 5

As might be expected, not much is different from Experiment 1. The distance between the best and worst result of each algorithm has become smaller. This is primarily due to a shift upward of the lower bound of around five to ten jobs for most algorithms.

While there are no major differences in total setup time compared to Experiment 1 either, one thing is notable. For FFW the total setup time has increased, although not by much, while for all other algorithms it stayed about the same or improved. It might be that break downs give FFW some time to wait for another job of the same type that can be added to the queue to reduce the setup time.

With machines no longer being plagued by break downs the flow time is lower for everything and the variation between runs is also reduced. Once more, also in terms of the average queue length, both the numbers and ranges are smaller for all algorithms. Average storage use is smaller with no break downs for R-Wasps, ATA and FFW. For the rest it remains very similar to the situation in Experiment 1. With both FT and KB very stable around the average one job in storage and the rest using storage rarely or not at all.

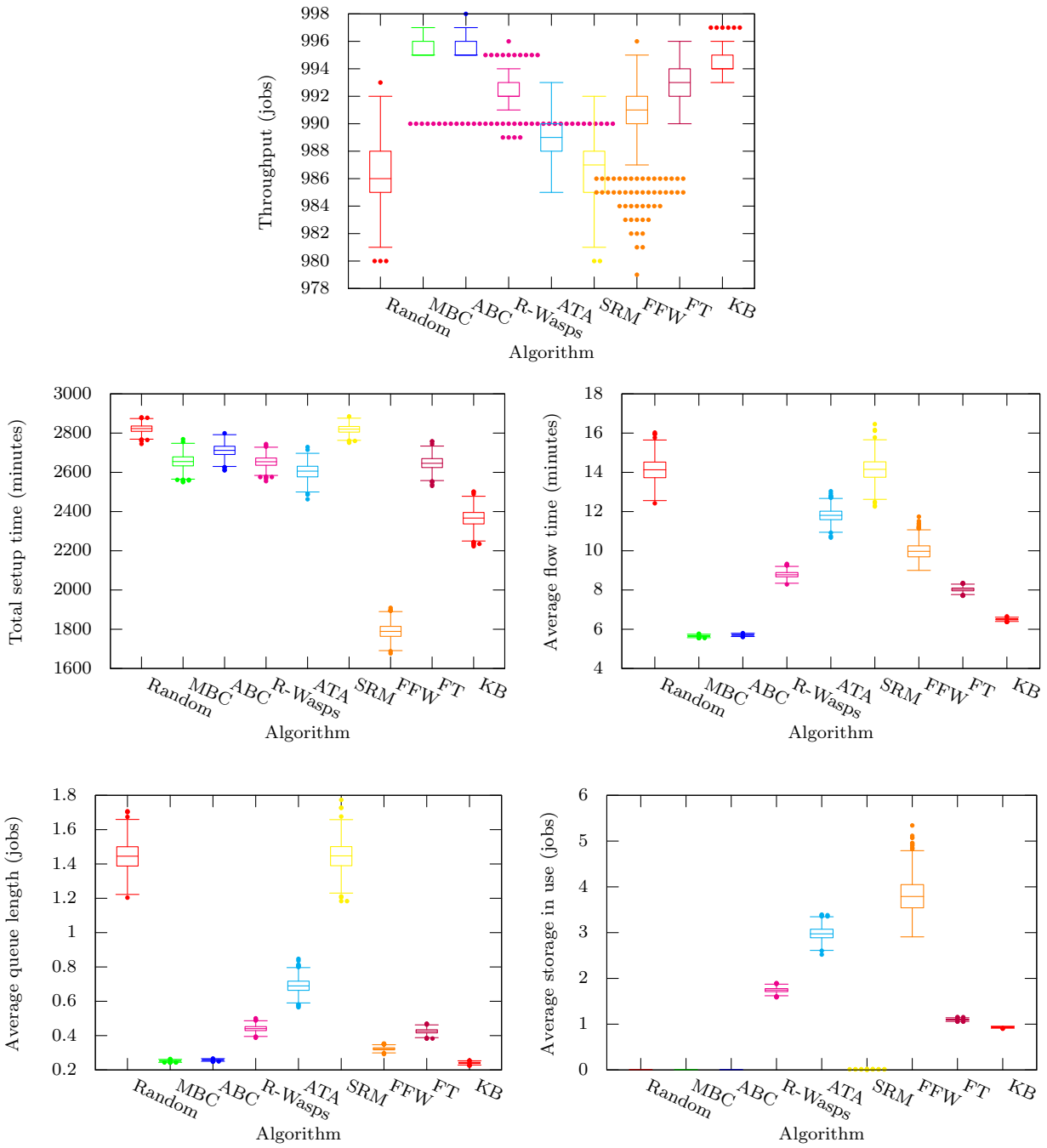


Figure 13: Experiment 5: No break downs, twenty job types with uniform random appearance rates.

8.8 Experiment 6

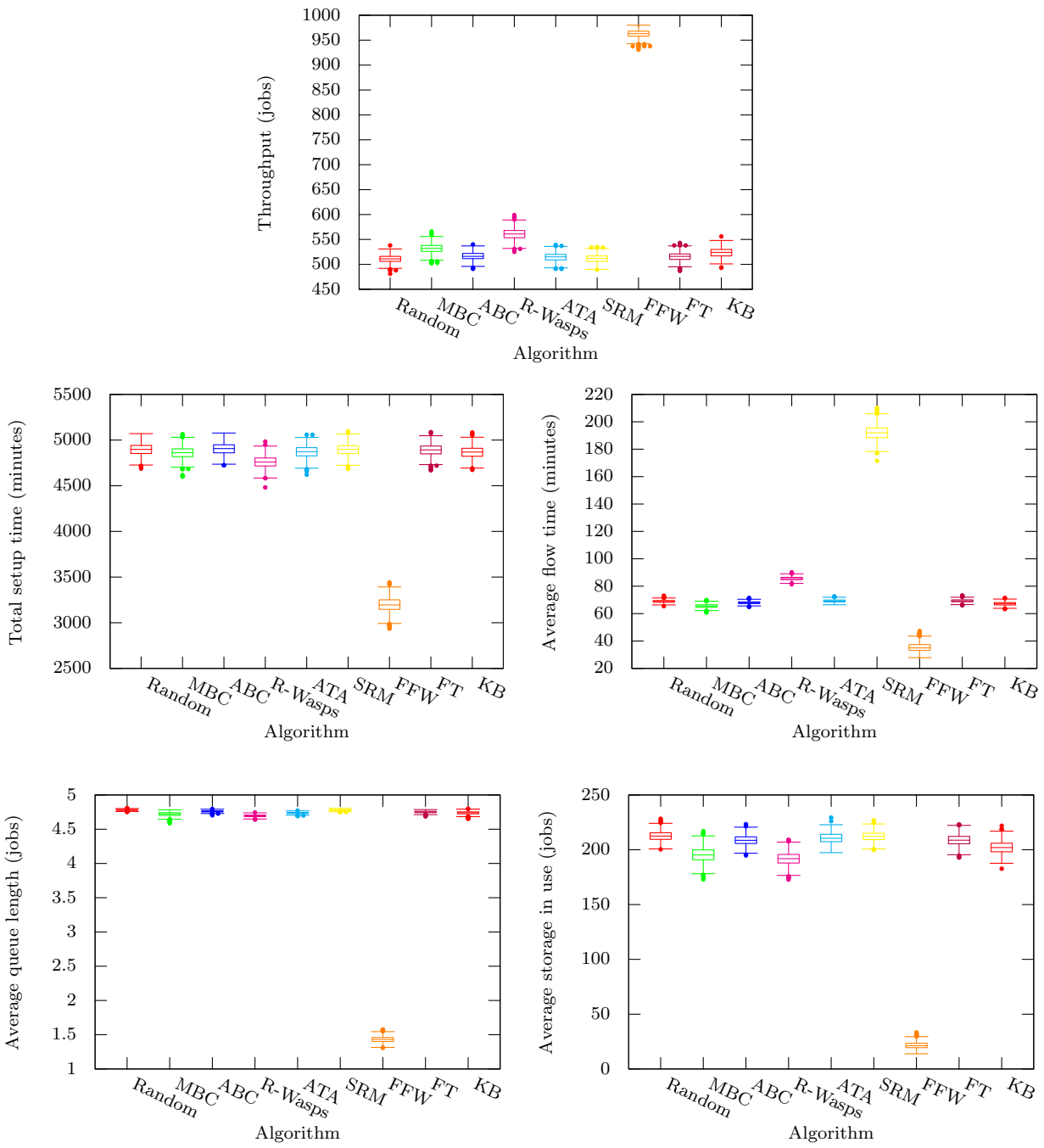


Figure 14: Experiment 6: Setup time of ten minutes, twenty job types with uniform random appearance rates.

Like with other situations where the problem was made more difficult, FFW is the only

algorithm able to produce a reasonable throughput with ten minute setup times. Logically, the total setup times are much longer and FFW is by far the best performing algorithm.

Unlike in Experiments 3 and 4, FFW now outperforms the rest in terms of average flow time. It is likely that these long setups have a much greater impact on the flow time than any delay in taking on jobs that may be induced by FFW. SRM is in the same situation as in the two previously mentioned experiments. Given the situation, both the average queue length and the average storage in use show no surprises.

9 Conclusion and discussion

This section starts of with a summary of the results. In the next subsection this is followed by the discussion of the contributions of this research. The final subsection then suggests a number of routes of interest for further work.

9.1 Result summary

Most notable in terms of results is the performance by the foraging for work (FFW) algorithm. On a uniform colour distribution it outperformed every other algorithm by an exceptional margin on the total setup time. Despite not performing particularly well in most other measures, performance in those is not poor enough to negate savings as a result from the reduction in setup time. Investigation of performance by FFW when using different step sizes resulted in very clear curves for all measures. As a result, optimisation for this algorithm, including multi-objective cases, should be quite simple.

When a more realistic colour distribution is considered FFW remains competitive in regards to the total setup time. In the other measures it is however a poor performer. For this reason the proposed algorithm (KB) seems like the most effective approach for this situation, as it is either competitive or not far behind in the most important measures. It does require some storage, but this also goes for the closest competitors. This shows that the proposed algorithm is effective, particularly since this situation is likely closest to the real world situation. Also notable is that this proves the viability of combining aspects from different algorithmic approaches for application to this problem. Additionally, being a new algorithm, the used parameters were never specifically tuned for KB, making it probable that it could perform even better. Experimentation with the step size for FFW showed that proper tuning could allow the algorithm a significant improvement in setup time over the one used in comparison to the other algorithms. This is however not even close to the margin in the uniform case, making algorithm selection for this more realistic situation a trade off between setup time and the other measures.

Despite ATA and KB being similarly effective in the standard situation with a realistic colour distribution in regards to the total setup time, FFW is the only algorithm that manages to utilise this to cope with the double production rate in Experiment 3. For the even more extreme situation where double production was considered together with an increased break down probability, none of the algorithms were able to process at a fast enough rate. Even so, once again FFW did show the best performance. With a tuned step size it might do even better, though it is questionable whether this would be significant enough to handle the situation.

It is no longer very surprising that FFW also performed best in experiments with a uniform distribution without break downs or with much longer setup times. A bit unexpected however is that none of the other approaches even come close to handling the uniform distribution with much longer setup times. At best they managed to reach a throughput slightly over half the produced jobs. Evidently, setups are a very significant part of this problem.

9.2 Discussion

Care needs to be taken in drawing conclusions from the experiments presented in this work. While they are solid experiments, the compared algorithms were given no tuning effort. Some parameters were taken as presented by the author of the respective algorithm and as such were optimised for a, sometimes slightly, different problem. While others were presented here originally without any significant thought behind them. Clearly, this is not a fair comparison. Even so, significant performance differences such as shown by the FFW algorithm on a number of cases cannot be ignored.

For use in the real world it is important to consider that it is not clear how well this problem fits such a situation. This is not just because of the lack of clarity about the exact problem originally faced by Morley et al. [22] [20] [21]. More significantly, the problem itself may well have changed since Morley et al. [22] [20] [21] dealt with it over fifteen years ago.

While this is not a biological study, the effectiveness of the foraging for work model may be relevant to the biological discussion of how realistically the model represents nature. The publication of the model and the ideas behind it was followed by a back and forth of critiques [26] [12] [27] [33] [13]. From a non-biologists point of view foraging for work does not seem like a bad explanation, even if this is likely not the only aspect influencing insect behaviour.

9.3 Further work

As should be evident, the painting problem is part of a larger whole in the automotive manufacturing industry. For future work it is important to take into account how painting fits in with the rest of the production process. The previously discussed related work (Section 3) may serve as a good starting point for this.

While there clearly is space for simpler algorithms such as FFW, parameter tuning is a very important problem for more complex algorithms. These more complex algorithms are required in optimising for situations with more sophisticated target measures, such as jobs with due dates. Therefore, further work focusing on tuning and a comparison after tuning of the approaches included here, as well as any others that are of interest, is needed. Despite being quite simple to tune, it may still be of interest to look at performance with different step sizes for FFW on more challenging problems, like those with double production rates considered in the experiments.

Another aspect of the FFW algorithm to look at is how it might be combined with other approaches, with the goal of taking the good parts from both worlds. For example a combination could consider machines that move around like in FFW, but go through a bidding process for jobs in their current position, rather than just assigning them.

Further work may also focus on problem variations where dynamic colour distributions are considered. These changes to the distribution during the production process may be abrupt, as previously considered by Cicirello and Smith [8], or more realistically, gradual.

Approaches that model division of labour, but were not included here, like the evolutionary threshold [11], network [3] and social inhibition [14] models should also be included in further work. As shown, untested approaches can sometimes produce exceptional results.

A Measurements

A.1 Experiment 1

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	972	2750	15.1433	1.56729	0
max	990	2887	22.2805	2.53875	0.122877
mean	982.145	2811.685	18.05086	1.966513	0.0004005991
std	3.021772	20.81714	1.056894	0.144516	0.004237715

Table 6: Measurements of random from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	986	2570	5.89447	0.267163	0
max	998	2766	7.17387	0.422861	0
mean	994.677	2671.913	6.261265	0.3013263	0
std	1.102673	31.89829	0.1598183	0.01742917	0

Table 7: Measurements of MBC from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	987	2661	5.9407	0.275155	0
max	998	2847	8.51055	0.611104	0
mean	994.566	2764.322	6.504912	0.3329397	0
std	1.214545	28.69543	0.2988738	0.03776914	0

Table 8: Measurements of ABC from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	974	2547	10.1456	0.554303	2.12288
max	994	2771	16.8525	1.30141	3.41758
mean	988.163	2658.595	12.48666	0.8084503	2.601182
std	2.740792	30.2754	1.04634	0.1129722	0.2164518

Table 9: Measurements of R-Wasps from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	974	2539	13.366	0.802199	3.24276
max	992	2758	19.1836	1.4808	4.56144
mean	984.928	2659.253	15.64927	1.066892	3.92036
std	2.671695	34.27988	0.9497614	0.1113889	0.2149252

Table 10: Measurements of ATA from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	969	2749	14.4635	1.47724	0.002997
max	990	2878	22.5407	2.56315	0.0989011
mean	982.365	2809.88	18.06674	1.966597	0.01520978
std	3.140643	21.58146	1.13534	0.1545397	0.00676451

Table 11: Measurements of SRM from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	976	1594	10.3225	0.327671	3.6963
max	996	1821	14.4206	0.429997	7.31868
mean	989.159	1710.664	11.83037	0.3716516	5.083562
std	2.927848	40.18689	0.6866412	0.01695261	0.578387

Table 12: Measurements of FFW from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	982	2585	8.36052	0.458969	1.07892
max	995	2771	11.9899	0.916511	1.1968
mean	991.281	2681.092	9.584264	0.6049524	1.138011
std	1.963643	31.69138	0.4266495	0.05482387	0.01787564

Table 13: Measurements of FT from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	983	2264	6.73742	0.253605	0.903097
max	997	2568	10.2397	0.889539	0.955045
mean	993.305	2449.797	7.572391	0.3786516	0.9339901
std	1.83121	43.21087	0.4053925	0.06929763	0.007802134

Table 14: Measurements of KB from experiment 1.

A.2 Experiment 1 - Foraging for work

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	944	1184	14.6738	0.385613	7.92008
max	992	1531	31.3524	0.615955	23.0659
mean	980.594	1382.564	20.03083	0.4849286	12.66046
std	6.545992	49.9011	2.282294	0.03481235	2.109015

Table 15: Measurements of FFW step size 1 from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	977	1587	9.91886	0.326387	3.57542
max	996	1841	15.4018	0.435992	8.26873
mean	989.096	1710.844	11.81697	0.3711478	5.078257
std	2.992949	40.96677	0.6673296	0.01739954	0.5607201

Table 16: Measurements of FFW step size 2 from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	981	1771	8.25829	0.294991	2.06693
max	996	1989	10.8773	0.382616	3.93007
mean	991.625	1873.83	9.374111	0.3354865	2.864794
std	2.118694	36.95737	0.3863008	0.01358969	0.294224

Table 17: Measurements of FFW step size 3 from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	986	1845	7.43548	0.287998	1.32567
max	997	2083	9.4073	0.362922	2.70829
mean	992.724	1964.652	8.264133	0.318709	1.866667
std	1.759807	35.97857	0.3032297	0.01195036	0.213559

Table 18: Measurements of FFW step size 4 from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	986	1904	6.98189	0.276724	0.842158
max	997	2146	8.48238	0.34494	1.83217
mean	993.352	2013.759	7.608657	0.3084692	1.285304
std	1.600205	35.76154	0.2441685	0.01118667	0.1526311

Table 19: Measurements of FFW step size 5 from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	986	1943	6.71932	0.277437	0.67033
max	997	2176	8.08048	0.344511	1.44755
mean	993.718	2056.959	7.263582	0.3042179	0.9637715
std	1.480766	36.12817	0.2189547	0.01054936	0.1258221

Table 20: Measurements of FFW step size 6 from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	988	1962	6.49899	0.269445	0.475524
max	997	2170	7.67437	0.33495	1.27473
mean	993.941	2071.881	6.985692	0.2993082	0.7250352
std	1.368719	38.09394	0.2029869	0.01026412	0.1129089

Table 21: Measurements of FFW step size 7 from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	988	1954	6.34274	0.262596	0.356643
max	997	2185	7.5206	0.333523	0.893107
mean	994.198	2088.946	6.810093	0.2973957	0.5648454
std	1.307863	36.75851	0.1772944	0.009720578	0.09105404

Table 22: Measurements of FFW step size 8 from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	989	1998	6.28945	0.267733	0.257742
max	997	2267	7.36647	0.329812	0.896104
mean	994.341	2105.302	6.695399	0.2964176	0.4537031
std	1.226468	38.26014	0.1803298	0.009892409	0.08915931

Table 23: Measurements of FFW step size 9 from experiment 1.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	987	1934	6.0743	0.262596	0.212787
max	998	2192	7.32797	0.33495	0.834166
mean	994.461	2066.434	6.52217	0.2886136	0.3584942
std	1.269681	39.99028	0.1703579	0.01020265	0.0777287

Table 24: Measurements of FFW step size 10 from experiment 1.

A.3 Experiment 2

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	981	1239	7.60725	0.535892	0
max	998	1617	10.9393	0.975595	0.002997
mean	991.737	1444.38	9.151961	0.734822	7.992002e-06
std	2.423986	60.66226	0.533585	0.07119156	0.0001411243

Table 25: Measurements of random from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	993	495	3.59519	0.0472384	0
max	998	880	4.26179	0.104325	0.00499501
mean	997.107	681.371	3.905778	0.07119593	9.99002e-06
std	0.9351072	67.9813	0.1094645	0.008788512	0.0002232718

Table 26: Measurements of MBC from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	991	872	4.02909	0.0890541	0
max	998	1251	4.85456	0.141859	0
mean	996.656	1059.885	4.318006	0.1104724	0
std	0.990275	60.46026	0.1013303	0.008310656	0

Table 27: Measurements of ABC from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	991	1077	4.80221	0.115171	0.363636
max	998	1343	5.70352	0.184102	0.625375
mean	995.84	1204.228	5.142044	0.1417835	0.4765982
std	1.214032	46.92483	0.1368007	0.009859111	0.03540391

Table 28: Measurements of R-Wasps from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	992	462	4.34002	0.0660767	0.414585
max	998	740	5.3002	0.13829	0.632368
mean	996.377	602.457	4.737175	0.09643769	0.5091749
std	1.115399	47.56767	0.1503583	0.01079039	0.03625756

Table 29: Measurements of ATA from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	983	1260	7.65694	0.529899	0.001998
max	997	1646	11.2118	1.02569	0.025974
mean	991.842	1439.513	9.131598	0.7302834	0.00950649
std	2.329477	61.18792	0.5388569	0.07154701	0.003449321

Table 30: Measurements of SRM from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	974	345	11.3024	0.937491	0.414585
max	996	527	14.0961	1.23377	1.76523
mean	988.497	442.975	12.511	1.067637	0.8323009
std	2.632284	28.87029	0.4531935	0.04928173	0.1857662

Table 31: Measurements of FFW from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	990	657	5.23417	0.104182	0.706294
max	998	1142	6.54217	0.240617	1.30969
mean	995.038	939.431	5.967947	0.1556052	1.143799
std	1.297783	73.87013	0.1873239	0.01683805	0.06645584

Table 32: Measurements of FT from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	987	362	4.73494	0.0565148	0.919081
max	998	769	6.14156	0.455686	0.968032
mean	995.362	524.475	5.15996	0.1341815	0.9447493
std	1.413844	64.82394	0.1863172	0.05682659	0.007274022

Table 33: Measurements of KB from experiment 2.

A.4 Experiment 2 - Foraging for work

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	962	299	12.1828	0.984873	0.979021
max	994	489	22.6891	1.39189	9.2018
mean	985.971	396.068	14.81908	1.146646	2.595667
std	4.194013	30.40733	1.299896	0.06334993	0.9937119

Table 34: Measurements of FFW step size 1 from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	976	327	11.227	0.914513	0.410589
max	996	534	14.5541	1.25303	2.06993
mean	988.372	440.777	12.51038	1.068036	0.8295747
std	2.754488	27.66692	0.4454512	0.0482037	0.1893828

Table 35: Measurements of FFW step size 2 from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	981	351	10.7556	0.888398	0.206793
max	997	579	13.0759	1.20993	0.676324
mean	989.11	457.433	11.86923	1.039518	0.4029727
std	2.300259	28.97687	0.3612556	0.04616014	0.0769052

Table 36: Measurements of FFW step size 3 from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	982	360	10.494	0.873413	0.114885
max	996	547	12.7747	1.17325	0.384615
mean	989.413	442.318	11.56688	1.023563	0.2275432
std	2.324615	30.18486	0.3656035	0.04804161	0.04410933

Table 37: Measurements of FFW step size 4 from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	981	321	10.3734	0.880407	0.0639361
max	997	527	12.5744	1.17739	0.256743
mean	989.546	432.54	11.37432	1.010607	0.1373725
std	2.277514	31.80324	0.3577825	0.04685591	0.02770118

Table 38: Measurements of FFW step size 5 from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	979	372	10.3754	0.899242	0.03996
max	996	571	12.7923	1.18339	0.191808
mean	989.487	464.552	11.38926	1.016904	0.09764032
std	2.280577	29.91806	0.3351866	0.04405876	0.02135966

Table 39: Measurements of FFW step size 6 from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	981	375	10.36	0.876981	0.026973
max	996	570	12.8434	1.19538	0.121878
mean	989.537	470.588	11.34877	1.015113	0.06349152
std	2.288639	29.26932	0.3533837	0.0467189	0.01501718

Table 40: Measurements of FFW step size 7 from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	983	318	10.1488	0.856571	0.017982
max	996	540	12.3623	1.15085	0.112887
mean	989.658	441.145	11.27838	1.009792	0.04708291
std	2.221254	31.14874	0.3490861	0.04641109	0.01272525

Table 41: Measurements of FFW step size 8 from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	981	389	10.2492	0.869416	0.012987
max	997	564	12.4349	1.13843	0.0799201
mean	989.604	471.522	11.28241	1.010059	0.0347712
std	2.304017	28.57772	0.3365061	0.04448134	0.01063746

Table 42: Measurements of FFW step size 9 from experiment 2.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	981	282	10.0281	0.846581	0.001998
max	997	451	12.3937	1.16684	0.0569431
mean	989.804	365.227	11.06571	0.9899667	0.01600398
std	2.26334	30.69354	0.3734213	0.04952474	0.007084585

Table 43: Measurements of FFW step size 10 from experiment 2.

A.5 Experiment 3

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	1366	1934	25.8716	4.58557	201.591
max	1523	2290	28.6038	4.70416	286.739
mean	1448.339	2102.608	27.02523	4.657345	243.3286
std	24.71475	55.60155	0.4192486	0.01851059	13.93413

Table 44: Measurements of random from experiment 3.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	1447	831	13.117	2.57186	0.223776
max	1948	2005	26.5598	4.64764	241.788
mean	1580.627	1723.057	23.2382	4.23239	150.4549
std	54.93662	125.117	1.721183	0.2505092	36.41589

Table 45: Measurements of MBC from experiment 3.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	1384	1871	24.8417	4.50023	180.994
max	1560	2248	28.1124	4.68147	284.112
mean	1465.022	2066.698	26.55695	4.612931	232.0562
std	25.07366	58.73574	0.4505102	0.03096243	14.52358

Table 46: Measurements of ABC from experiment 3.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	1614	1043	25.3424	4.05339	52.1608
max	1833	1534	30.7082	4.63694	169.277
mean	1699.85	1339.214	28.58335	4.488826	120.4899
std	30.21911	81.86713	0.7039181	0.08013066	17.27197

Table 47: Measurements of R-Wasps from experiment 3.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	1421	1643	23.9484	4.27232	128.655
max	1610	2055	28.1351	4.67106	255.51
mean	1521.348	1873.05	26.37678	4.560945	198.7118
std	28.31152	61.84073	0.5763526	0.05831386	17.45025

Table 48: Measurements of ATA from experiment 3.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	1372	1914	98.049	4.56045	194.333
max	1529	2320	134.665	4.69603	287.814
mean	1449.406	2103.015	117.3809	4.643088	242.5445
std	25.09875	56.01043	5.911663	0.01969848	14.21875

Table 49: Measurements of SRM from experiment 3.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	1854	189	17.8498	2.7257	8.42358
max	1975	432	50.4452	3.65791	76.7892
mean	1941.37	324.723	25.26813	3.141106	22.30051
std	14.74464	36.07475	3.760449	0.1590964	7.579288

Table 50: Measurements of FFW from experiment 3.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	1364	1770	26.3372	4.4918	179.687
max	1563	2288	30.085	4.68632	293.691
mean	1459.371	2072.172	28.32656	4.608672	235.4903
std	25.94777	65.23708	0.5067027	0.03252948	14.6618

Table 51: Measurements of FT from experiment 3.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	1265	1680	24.911	4.28758	170.389
max	1570	2209	28.6447	4.6946	329.945
mean	1486.434	2000.793	26.79311	4.555803	215.7592
std	27.59495	59.13127	0.6017087	0.06383077	16.6348

Table 52: Measurements of KB from experiment 3.

A.6 Experiment 4

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	849	1218	34.3561	4.56102	402.584
max	1119	1621	43.6113	4.7504	549.131
mean	982.059	1432.855	38.5942	4.67703	472.4248
std	41.93698	65.1865	1.531227	0.02557375	23.43944

Table 53: Measurements of random from experiment 4.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	897	1125	31.9744	4.48196	345.275
max	1171	1527	42.3356	4.72086	526.483
mean	1020.965	1330.407	37.04383	4.628192	446.6468
std	42.05761	64.77347	1.47674	0.0381096	24.86791

Table 54: Measurements of MBC from experiment 4.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	860	1174	33.3036	4.54247	378.737
max	1133	1626	43.7442	4.7377	543.143
mean	991.746	1420.019	38.18308	4.663307	466.6708
std	40.92342	67.14405	1.493585	0.02581459	23.61977

Table 55: Measurements of ABC from experiment 4.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	1154	243	31.1535	4.4343	255.022
max	1453	769	36.6932	4.66335	406.056
mean	1294.094	488.25	33.8437	4.573762	329.7372
std	41.3192	80.0602	0.8688697	0.03299869	23.57464

Table 56: Measurements of R-Wasps from experiment 4.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	871	1052	33.9454	4.50807	371.292
max	1157	1475	43.509	4.70902	534.563
mean	1028.399	1280.144	37.78045	4.644593	446.4115
std	41.86912	62.76468	1.415546	0.02847825	24.56684

Table 57: Measurements of ATA from experiment 4.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	839	1217	155.62	4.48966	400.224
max	1106	1666	224.829	4.72528	560.1
mean	983.103	1431.04	182.7486	4.662677	471.9263
std	40.406	69.54166	9.591807	0.02622584	22.70943

Table 58: Measurements of SRM from experiment 4.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	1237	27	29.3892	3.61938	176.159
max	1586	222	66.6659	4.56816	415.713
mean	1407.737	75.872	37.51526	4.28865	277.9881
std	57.40461	32.65492	6.225449	0.1551308	34.70102

Table 59: Measurements of FFW from experiment 4.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	867	1117	35.7024	4.55917	392.103
max	1109	1656	45.4002	4.72314	554.417
mean	990.511	1415.726	39.91931	4.656791	467.7818
std	42.38892	75.27649	1.554148	0.02500834	23.84415

Table 60: Measurements of FT from experiment 4.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	661	931	34.5419	4.6181	372.493
max	1146	1569	43.5417	4.83359	630.691
mean	980.385	1363.237	39.31544	4.743606	469.3271
std	65.70878	94.60112	1.489828	0.02808497	34.92635

Table 61: Measurements of KB from experiment 4.

A.7 Experiment 5

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	980	2745	12.4216	1.20366	0
max	993	2880	16.0263	1.707	0
mean	986.48	2822.797	14.14429	1.447038	0
std	2.027736	20.3944	0.5737676	0.08110192	0

Table 62: Measurements of random from experiment 5.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	995	2550	5.55075	0.242759	0
max	997	2769	5.77085	0.263596	0
mean	995.335	2655.812	5.657568	0.2528097	0
std	0.5450432	35.28044	0.03538875	0.003359736	0

Table 63: Measurements of MBC from experiment 5.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	995	2610	5.61145	0.248468	0
max	998	2799	5.8012	0.26645	0
mean	995.274	2711.936	5.713853	0.2581536	0
std	0.4951456	30.87911	0.03098626	0.002939543	0

Table 64: Measurements of ABC from experiment 5.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	989	2555	8.29032	0.387185	1.58741
max	996	2744	9.33266	0.500928	1.8981
mean	992.193	2653.871	8.780827	0.4405759	1.74447
std	1.022153	27.55892	0.1647668	0.01737378	0.04877564

Table 65: Measurements of R-Wasps from experiment 5.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	985	2463	10.6714	0.565294	2.52048
max	993	2729	13.0343	0.847152	3.39261
mean	989.186	2604.692	11.80507	0.6913281	2.980095
std	1.373787	39.03964	0.3286513	0.04022822	0.1372903

Table 66: Measurements of ATA from experiment 5.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	980	2751	12.2591	1.18353	0
max	992	2885	16.4554	1.77337	0.016983
mean	986.516	2819.401	14.15448	1.447496	0.006576424
std	2.060094	21.39392	0.5888896	0.08311184	0.002619971

Table 67: Measurements of SRM from experiment 5.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	979	1677	9.00101	0.292136	2.90709
max	996	1909	11.7394	0.353646	5.34166
mean	990.813	1789.346	9.991924	0.3232918	3.816048
std	2.3672	37.37862	0.4209856	0.009348192	0.377438

Table 68: Measurements of FFW from experiment 5.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	990	2531	7.70624	0.382759	1.05495
max	996	2758	8.35915	0.468958	1.15285
mean	992.958	2646.73	8.029806	0.4250929	1.09942
std	0.9279532	33.45119	0.1032444	0.01381487	0.01571101

Table 69: Measurements of FT from experiment 5.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	993	2223	6.36884	0.225204	0.9001
max	997	2502	6.65493	0.254319	0.955045
mean	994.475	2365.346	6.506397	0.2408722	0.9334366
std	0.6781112	45.45849	0.04141832	0.004589916	0.007929449

Table 70: Measurements of KB from experiment 5.

A.8 Experiment 6

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	481	4688	65.4758	4.7541	200.368
max	538	5071	73.0915	4.80705	228.282
mean	511.208	4895.711	68.92227	4.780941	212.593
std	7.612929	63.93747	0.9761201	0.008775333	4.352763

Table 71: Measurements of random from experiment 6.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	502	4599	60.8755	4.58599	172.946
max	566	5065	69.9243	4.7835	216.984
mean	532.13	4859.85	65.56966	4.719833	195.2864
std	9.844695	64.21017	1.375697	0.02899462	6.957846

Table 72: Measurements of MBC from experiment 6.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	491	4724	64.9125	4.70273	194.849
max	540	5077	71.3083	4.79449	223.393
mean	516.52	4904.923	67.97606	4.760523	208.7157
std	7.475392	63.00929	0.9632836	0.01177013	4.36982

Table 73: Measurements of ABC from experiment 6.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	525	4482	81.3718	4.64264	172.837
max	599	4984	90.1098	4.74426	209.218
mean	560.849	4759.029	85.54633	4.695047	191.7586
std	11.117	65.49701	1.370627	0.01703461	5.890342

Table 74: Measurements of R-Wasps from experiment 6.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	491	4621	66.4339	4.69274	197.408
max	539	5058	72.5316	4.77222	229.325
mean	514.416	4870.396	69.1769	4.736749	210.6341
std	8.009563	65.99549	1.016082	0.01232856	4.766408

Table 75: Measurements of ATA from experiment 6.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	489	4685	171.483	4.74768	199.812
max	535	5096	210.169	4.80391	226.8
mean	511.668	4895.631	192.1301	4.778518	212.3086
std	7.78411	63.36426	5.294456	0.009384517	4.410509

Table 76: Measurements of SRM from experiment 6.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	931	2935	27.8555	1.30926	13.8042
max	980	3441	47.2986	1.57514	33.3057
mean	962.387	3195.686	35.45724	1.431731	21.64107
std	7.465317	78.27701	3.296882	0.04358313	3.123329

Table 77: Measurements of FFW from experiment 6.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	487	4670	66.0221	4.68745	192.88
max	543	5087	73.191	4.78678	223.014
mean	515.701	4890.511	69.34823	4.750211	208.7466
std	8.096985	65.05453	1.061029	0.01472438	4.833845

Table 78: Measurements of FT from experiment 6.

measure	throughput	total setup time	average flow time	average queue length	average storage in use
min	493	4674	63.286	4.65135	182.684
max	556	5083	71.5071	4.79849	221.936
mean	523.774	4867.571	67.17855	4.737762	202.1623
std	9.169897	65.55942	1.241155	0.02226406	5.707443

Table 79: Measurements of KB from experiment 6.

References

[1] Ali Allahverdi, Jatinder N.D. Gupta, and Tariq Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239, 1999.

[2] Ali Allahverdi, C.T. Ng, T.C. Edwin Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008.

[3] Samuel N. Beshers and Jennifer H. Fewell. Models of division of labor in social insects. *Annual review of entomology*, 46(1):413–440, 2001.

[4] Eric Bonabeau, Guy Théraulaz, and Jean-Louis Deneubourg. Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 263(1376): 1565–1569, 1996.

[5] Nils Boysen, Armin Scholl, and Nico Wopperer. Resequencing of mixed-model assembly lines: Survey and research agenda. *European Journal of Operational Research*, 216(3): 594–604, 2012.

[6] Mike Campos, Eric Bonabeau, Guy Théraulaz, and Jean-Louis Deneubourg. Dynamic scheduling and division of labor in social insects. *Adaptive Behavior*, 8(2):83–95, 2000.

[7] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. A review of machine scheduling: Complexity, algorithms and approximability. In *Handbook of combinatorial optimization*, pages 1493–1641. Springer, 1999.

[8] Vincent A. Cicirello and Stephen F. Smith. Wasp-like agents for distributed factory coordination. *Autonomous Agents and Multi-agent systems*, 8(3):237–266, 2004.

[9] Fong-Yuen. Ding and Hui Sun. Sequence alteration and restoration related to sequenced parts delivery on an automobile mixed-model assembly line with multiple departments. *International Journal of Production Research*, 42(8):1525–1543, 2004.

[10] Ana Duarte, Franz J. Weissing, Ido Pen, and Laurent Keller. An evolutionary perspective on self-organized division of labor in social insects. *Annual Review of Ecology, Evolution, and Systematics*, 42:91–110, 2011.

[11] Ana Duarte, Ido Pen, Laurent Keller, and Franz J. Weissing. Evolution of self-organized division of labor in a response threshold model. *Behavioral ecology and sociobiology*, 66 (6):947–957, 2012.

[12] Nigel R. Franks and Chris Tofts. Foraging for work: how tasks allocate workers. *Animal Behaviour*, 48(2):470–472, 1994.

[13] Nigel R. Franks, Chirs Tofts, and Ana B. Sendova-Franks. Studies of the division of labour: neither physics nor stamp collecting. *Animal behaviour*, 53(1):219–224, 1997.

[14] Deborah M. Gordon, Brian C. Goodwin, and Lynn E.H. Trainor. A parallel distributed model of the behaviour of ant colonies. *Journal of theoretical Biology*, 156(3):293–307, 1992.

[15] Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

[16] Robert R. Inman and D.M. Schmeling. Algorithm for agile assembling-to-order in the automotive industry. *International Journal of Production Research*, 41(16):3831–3848, 2003.

[17] Oran Kittithreerapronchai and Carl Anderson. Do ants paint trucks better than chickens? markets versus response thresholds for distributed dynamic scheduling. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 2, pages 1431–1439. IEEE, 2003.

[18] Lakshmanan Meyyappan, Can Saygin, and Cihan H. Dagli. Real-time routing in flexible flow shops: a self-adaptive swarm-based control model. *International Journal of Production Research*, 45(21):5157–5172, 2007.

[19] Dug Hee Moon, Ha Seok Kim, and Cheng Song. A simulation study for implementing color rescheduling storage in an automotive factory. *Simulation*, 81(9):625–635, 2005.

[20] Richard E. Morley. Painting trucks at general motors: The effectiveness of a complexity-based approach. *Embracing Complexity: Exploring the application of complex adaptive systems to business*, pages 53–58, 1996.

[21] Richard E. Morley and Gregg Ekberg. Cases in chaos: complexity-based approaches to manufacturing. In *Embracing complexity: A colloquium on the application of complex adaptive systems to business*, pages 97–702, 1998.

[22] Richard E. Morley and Charles C. Schelberg. An analysis of a plant-specific dynamic scheduler. In *Proceedings of the NSF workshop on dynamic scheduling*, pages 115–122, 1993.

[23] Shervin Nouyan. Agent-based approach to dynamic task allocation. In *Ant Algorithms*, pages 28–39. Springer, 2002.

[24] Shervin Nouyan, Roberto Ghizzioli, Mauro Birattari, and Marco Dorigo. An insect-based algorithm for the dynamic task allocation problem. *KI*, 19(4):25–31, 2005.

[25] R. Christopher Plowright and Catherine M.S. Plowright. Elitism in social insects: a positive feedback model. In *Interindividual behavioral variability in social insects*, pages 419–431. Westview Press, 1988.

[26] Gene E. Robinson, Robert E. Page Jr., and Zhi-Yong Huang. Temporal polyethism in social insects is a developmental process. *Animal Behaviour*, 48(2):467–469, 1994.

[27] Simon K. Robson and Samuel N. Beshers. Division of labour and 'foraging for work': simulating reality versus the reality of simulations. *Animal Behaviour*, 53(1):214–218, 1997.

[28] Rubén Ruiz and José Antonio Vázquez-Rodríguez. The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1):1–18, 2010.

[29] Guy Theraulaz, Simon Goss, Jacques Gervet, and Jean-Louis Deneubourg. Task differentiation in polistes wasp colonies: a model for self-organizing groups of robots. In *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, pages 346–355. MIT Press, 1991.

[30] Guy Théraulaz, Eric Bonabeau, and Jean-Louis Deneubourg. Self-organization of hierarchies in animal societies: the case of the primitively eusocial wasp polistes dominulus christ. *Journal of theoretical Biology*, 174(3):313–323, 1995.

[31] Guy Théraulaz, Eric Bonabeau, and Jean-Louis Deneubourg. Response threshold reinforcements and division of labour in insect societies. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 265(1393):327–332, 1998.

[32] Chris Tofts. Algorithms for task allocation in ants. (a study of temporal polyethism: theory). *Bulletin of Mathematical Biology*, 55(5):891–918, 1993.

[33] James F.A. Traniello and Rebeca B. Rosengaus. Ecology, evolution and division of labour in social insects. *Animal Behaviour*, 53(1):209–213, 1997.

[34] Xuefeng Yu and Bala Ram. Bio-inspired scheduling for dynamic job shops with flexible routing and sequence-dependent setups. *International journal of production research*, 44(22):4793–4813, 2006.