

## SCHEDULING SEQUENCE-DEPENDENT COLOUR PRINTING JOBS

J. Schuurman<sup>1#</sup> & J.H. van Vuuren<sup>1\*</sup>

### ARTICLE INFO

#### Article details

Submitted by authors 2 Jan 2015  
Accepted for publication 5 Apr 2016  
Available online 12 Aug 2016

#### Contact details

\* Corresponding author  
vuuren@sun.ac.za

#### Author affiliations

<sup>1</sup> Department of Industrial  
Engineering  
Stellenbosch University, South  
Africa

# Author was enrolled for a  
B Eng (Industrial) degree in the  
Department of Industrial  
Engineering at Stellenbosch  
University, South Africa

#### DOI

<http://dx.doi.org/10.7166/27-2-1119>

### ABSTRACT

A scheduling problem in the colour printing industry is considered in this paper. The problem is to find an optimal assignment of print jobs to each of a set of colour printers, as well as an optimal processing sequence for the set of jobs assigned to each printer. The objective is to minimise the makespan of the schedule to achieve a suitable balance between the workloads of the printers and the efficiencies of the job sequences assigned to the printers. A novel aspect of the problem is the way in which the printer set-up times associated with the jobs are job sequence-dependent – it is possible to exploit commonalities between the colours required for successive jobs on each machine. We solve this problem approximately by using a simple heuristic and three well-known metaheuristics. Besides colour printing, the scheduling problem considered here admits many other applications. Some of these alternative applications are also briefly described.

### OPSOMMING

'n Skeduleringsprobleem uit die kleurdrukwerkbedryf word in hierdie artikel oorweeg. Die probleem vra vir 'n optimale toewysing van take aan elk van 'n versameling kleurdrukkers, sowel as die spesifikasie van 'n optimale volgorde waarin die take wat aan elke drukker toegewys is, uitgevoer moet word. Die doel is om die prosestyd van die drukker wat laaste klaarmaak te minimeer om sodoende 'n aanvaarbare balans tussen die werkladings van die drukkers en die taakvolgorde vir elke drukker te bewerkstellig. 'n Nuwe aspek van die probleem is die manier waarop die opsteltjye van die drukkers vir die take volgorde-afhanklik is – dit is moontlik om gemeenskaplikhede tussen die kleure wat vir opeenvolgende take op elke masjien benodig word, uit te buit. Ons los hierdie probleem benaderd op deur gebruik te maak van 'n eenvoudige heuristiek asook drie bekende metaheuristieke. Behalwe vir kleurdrukwerk, het die skeduleringsprobleem wat hier beskou word vele ander toepassings. Sommige van hierdie toepassings word ook kortliks beskryf.

## 1 INTRODUCTION

Snack foods, such as candy bars and crisps, are usually packaged in convenient plastic or foil wrappers adorned with bold, colourful logos and other designs to draw the attention of consumers. In order to achieve the desired finish, colour overlay printing techniques are used. Fierce competition forces production facilities that specialise in this form of printing to improve the efficiency of their operations so that they can handle large printing volumes in a time-efficient and cost-effective manner. At the heart of realising this high level of efficiency at such a production facility lies a scheduling process in which print jobs have to be assigned to, and sequenced on, a collection of printers functioning in parallel and possibly at different speeds. This process aims to balance the workload among the printers to maximise print volume throughput.

Consider a collection of  $n$  printing jobs indexed by the set  $J = \{1, \dots, n\}$ . Associated with each job  $j \in J$  is a *colour set*  $C_j$ , indicating which colours are required for the job, and the volume  $w_j$  of the job (usually measured in kilograms of printing material). The colour set of each job is typically a small subset of a larger universal set  $C$  of all possible colours at the disposal of the printing company. Each of the jobs in  $J$  requires processing on one of  $m$  parallel printers indexed by the set  $M = \{1, \dots, m\}$ . Printer  $k \in M$  operates at a printing speed of  $q_k$  (measured in kilograms of printing material that can be produced per minute), and is equipped with a colour cartridge magazine of size  $b_k$  (which is the number of colours that can be accommodated simultaneously in the printer).

Job  $j \in J$  can only be processed on printer  $k \in M$  if the magazine of printer  $k$  is large enough to accommodate all the colours in the colour set of job  $j$  – *i.e.*, if  $|C_j| \leq b_k$ . All the colours required for a job have to be loaded into the printer processing that job before beginning the print job. Furthermore, it is assumed that the printer magazine is empty at the start of the scheduling period, and that it should be empty again at the end of the printing schedule; but it typically remains fully-loaded throughout the duration of the scheduling period, because a fixed downtime is associated with reloading any colour cartridge.

Every job processed on one of the printers involves a production time consisting of two phases. The first of these is the *printing phase*, whose duration is both printer-dependent (because of possible variation in the processing speeds of the printers) and job-dependent (determined by the print volume of the job). The duration of the printing phase of job  $j \in J$ , also called its *processing time*, when performed by printer  $k \in M$  is therefore

$$p_{jk} = \begin{cases} \frac{w_j}{q_k} & \text{if } |C_j| \leq b_k, \\ \infty & \text{if } |C_j| > b_k. \end{cases}$$

The second phase is the *set-up phase*. This involves removing from cartridges the ink colours not required for the next job scheduled for processing on that particular printer, to free up just enough space in the printer's magazine for colours that are indeed required for the next job but that are not present in the magazine. Each such colour change incurs a constant set-up time  $a$  (measured in minutes) that is required to empty, clean, and reload an ink cartridge, and is referred to as a *wash*. If, however, a colour required for a specific job was also required for the previous job processed on a particular printer (and is therefore already in the printer's magazine when the reloading of cartridges for the next job takes place), then the above set-up time may be avoided for that colour. A sequence-dependent set-up time  $s_{ijk}$  is thus incurred when following up job  $i$  by job  $j$  on printer  $k$ . This set-up time can be calculated as  $a$  multiplied by the required number of colour changes in order to be able to process job  $j$ . Given a set of jobs to be processed by printer  $k \in M$ , the completion time  $\Phi_k$  of the printer may therefore be reduced by a judicious choice of the sequence in which the jobs are to be performed on the printer.

A printing schedule for the *parallel machine colour print scheduling problem* (PMCPSP) described above therefore consists of an assignment of jobs to the various printers, and a processing sequence for the set of jobs assigned to each printer. The *makespan*  $\Omega_{\max}$  of the printing schedule is the completion time of the printer that finishes last – *i.e.*,  $\Omega_{\max} = \max \{\Phi_1, \dots, \Phi_m\}$ , assuming that all printers start processing at time zero. The objective in the PMCPSP is to compute a printing schedule achieving the smallest possible makespan that will achieve a suitable balance between the workloads of the printers and the efficiencies of the job sequences assigned to the printers.

Even the special case of the PMCPSP, in which there is only  $m = 1$  printer, is NP-hard [5,9]. Furthermore, it is difficult for even an experienced human operator to come up with high-quality solutions to the PMCPSP for the following reasons:

1. The duration of a print job only becomes known once it has been assigned to a printer. Moving a print job from one printer to another in a bid to balance the workload of printers in a schedule therefore requires non-intuitive calculations.
2. The set-up time associated with a print job only becomes known once the jobs assigned to the relevant printer have been sequenced, because it depends on the commonality of colours between a particular job and its predecessor.

For these reasons, it is desirable to aid production managers in the complex decisions associated with solving the PMCPSP by providing them with computerised decision support – based on mathematical modelling techniques – rather than requiring them to make scheduling decisions based purely on intuition or experience. Although the literature contains mathematical models for scheduling problems that conform to the particular structure and requirements of the PMCPSP, these models are solvable to optimality for only the tiniest of problem instances. Since realistic instances of the PMCPSP are well beyond the reach of exact solution methods within reasonable timeframes, our objectives in this paper are (1) to design a user-friendly *decision support system* (DSS) – based on various heuristic model solution approaches – that is capable of aiding production engineers to determine high-quality solutions to the PMCPSP; and (2) to demonstrate the practical workability of this DSS.

This paper is organised as follows. Section 2 contains a brief review of two approaches to modelling the PMCPSP mathematically, while the working of one heuristic and three metaheuristic solution procedures, which are applied in this paper to the PMCPSP, are outlined in Section 3. In Section 4, these approximate solution approaches are compared with respect to their execution times and the qualities of solutions they produce in the context of a number of small test instances, after which we turn our attention in Section 5 to the design of a user-friendly DSS for solving larger instances of the PMCPSP. This DSS is applied to a realistic instance of the PMCPSP in the form of a case study in Section 6 to demonstrate its workability. In Section 7 the paper ends with a number of conclusions, ideas for possible future work, and descriptions of alternative applications conforming to the requirements of the PMCPSP. Mathematical PMCPSP model formulations are presented in an appendix.

## 2 LITERATURE REVIEW

There are two formulations of parallel machine scheduling problems with sequence-dependent set-up times in the literature that resemble the PMCPSP. The first was introduced by Helal *et al.* [8] in 2006. Avalos-Rosales *et al.* [2] proposed an improved reformulation of this original model in 2013 in response to a weakness discovered in the earlier formulation. To promote the general readability of the paper by non-mathematically-inclined readers, these two formulations are relegated to an appendix at the end of the paper.

In the model formulation proposed by Helal *et al.* [8], the makespan is linearised as the maximum of the completion times of all the jobs that have to be processed. Avalos-Rosales *et al.* [2] found, however, that the lower bound on the schedule's makespan produced by the linear relaxation of the model of Helal *et al.* [8] is very weak, and this prompted them to reformulate the model by linearising the makespan as the maximum of the spans of all the machines.

Avalos-Rosales *et al.* [2] performed computational tests to compare the efficacy of a commercial solver to solve their model reformulation for a suite of test problem instances (available online [18]), as opposed to solving the original model formulation of Helal *et al.* [8]. They also used the results thus obtained to determine the dimensions (*i.e.*, the values of the parameters  $m$  and  $n$ ) of problem instances that are realistically solvable by exact methods.

Avalos-Rosales *et al.* [2] generated two sets of test instances for this purpose: a set of small instances and a set of larger instances. The set of small instances involved  $m \in \{2,3,4,5\}$  machines and  $n \in \{6,8,10,12\}$  jobs. For each value of  $m$ , a total of ten instances of sets of  $n$  jobs were generated, with job durations drawn from uniform distributions with ranges 1–9, 1–49, 1–99, and 1–124. In all cases, the set-up times between jobs were drawn from a uniform distribution with range 1–99, thus giving rise to  $4 \times 4 \times 10 \times 4 = 640$  small problem instances. Using CPLEX 12.2 on a Pentium dual core 2 GHz processor with 3Gb of RAM running in the operating system Ubuntu 11.1, they were able to solve only 565 (88.28 per cent) of these small instances to optimality within one hour when adopting the earlier model formulation of Hilal *et al.* [8], and all (100 per cent) of the small instances when using their own model reformulation. In fact, the average time required to solve the small instances to optimality with the latter formulation was only 0.625 seconds. This led Avalos-Rosales *et al.* [2] to conclude that their formulation is superior from a practical perspective.

The set of larger instances again involved  $m \in \{2,3,4,5\}$  machines, but this time  $n \in \{15,20,25,30,35,40\}$  jobs. For each value of  $m$ , a total of ten instances of sets of  $n$  jobs were again generated, with job durations drawn from uniform distributions with ranges 1–49, 1–99, and 1–

124. In all cases, the set-up times between jobs were again drawn from a uniform distribution with range 1–99, thus giving rise to  $4 \times 6 \times 10 \times 3 = 720$  larger problem instances. In view of the fact that not all small instances of the model formulation of Helal *et al.* [8] were solvable within one hour of computation time, Alvalos-Rosales *et al.* [2] elected to solve the larger instances using only their own superior formulation. Using the same computing platform as above, they were able to solve only 636 (88.33 per cent) of the larger instances to optimality within one hour. For the largest instances where  $m = 5$  machines and  $n = 35$  or 40 jobs, an average optimality gap of between 3.3 per cent and 3.8 per cent was achieved within one hour of computing time.

In view of these results and the NP-hardness of even the special case of the PMCPSP with  $m = 1$  machine [6,13], the portion of instances that are solvable to optimality within one hour (or even a few hours, for that matter) is expected to decrease exponentially as  $m$  and/or  $n$  increases. Because realistic instances of the PMCPSP are expected to contain well in excess of  $n = 100$  jobs, we conclude that realistic instances of the PMCPSP are not solvable within a realistic timeframe using exact methods. Instead, therefore, we consider one heuristic and three metaheuristic solution approaches in the next section to solve the PMCPSP approximately.

### 3 APPROXIMATE SOLUTION METHODOLOGIES

The four approximate methods considered in this paper to solve the PMCPSP are the largest-processing-time-first rule, a local search, the method of tabu search, and simulated annealing, all described in more detail below.

#### 3.1 Largest-processing-time-first rule

The *largest-processing-time-first* (LPTF) rule is a well-known scheduling heuristic [15] that aims to minimise the makespan of a production schedule on parallel machines by scheduling shorter jobs towards the end of the scheduling period where they can be used to balance the production load of the various machines. According to the LPTF rule, the  $m$  largest jobs are scheduled first on the  $m$  machines, one job per machine. Afterwards, whenever a machine is freed, the largest of those jobs not yet scheduled is processed on that machine. Due to its inflexibility, the LPTF rule is rarely used in isolation; instead, it is often used as a starting point for more advanced scheduling metaheuristics.

#### 3.2 Improving local search

An improving local search is an iterative metaheuristic that moves from one candidate solution to a neighbouring solution of higher quality in solution space during every iteration [5,17]. The quality of a solution is measured in terms of the objective function of the optimisation problem at hand. To search within the solution space in a systematic manner, a problem-specific neighbourhood structure is induced around the current solution by a set of so-called *moves* (perturbations performed with respect to the current solution). Solutions in this neighbourhood are evaluated, and the highest-quality solution is adopted as the next current solution, provided that this solution is of better quality than the current solution. If there is no improving solution in the neighbourhood of the current solution, then the search terminates and the current solution is reported as the (locally) optimal solution to the optimisation problem. A major disadvantage of an improving local search is, of course, that it may easily become trapped at an inferior locally optimal solution, and hence the procedure is sensitive to the candidate solution chosen to initiate the search.

In this paper, we generate the initial (or first current) solution either randomly or by using the LPTF rule. The neighbourhood of the current solution results from applying so-called *displacement moves* [17] to the current solution. In the context of the PMCPSP, a displacement move consists of moving some job  $j$  from its current position in the production sequence of some printer  $k$  to any position in the production sequence of any printer  $k'$  (allowing for the possibility that perhaps  $k = k'$ ). All combinations of  $j$  and  $k$  are considered in turn, resulting in a potentially large neighbourhood for each candidate solution and a search procedure that is essentially deterministic from the initial solution onwards.

#### 3.3 Tabu search

Introduced by Glover [7] in 1986, the tabu search method is a metaheuristic inspired by the mechanisms of human memory. In contrast with an improving local search, the tabu search method – which is also an iterative search procedure – allows for the escape from locally optimal solutions by (possibly) accepting non-improving moves when moving from the current solution to the next

solution. To avoid cycling, a so-called *tabu list* of a fixed number of the most recent moves performed during the search is maintained, and the reversal of any move in this list is (temporarily) forbidden when deciding on the next move to perform. The tabu list is managed as a first-in-first-out queue of fixed length, called the *tabu tenure*, by removing the oldest entry from the list when a new entry is inserted into the list. The tabu tenure – denoted by  $\tau$  – is an important parameter that affects the efficiency of the search, is problem-specific, and is usually determined empirically by prior experimentation [7]. Because the search does not necessarily terminate at a locally optimal solution (as is the case in an improving local search), a pre-specified, fixed number  $I_{max}$  of search iterations is usually performed before terminating the search. Furthermore, since the last current solution is not necessarily the best candidate uncovered during the search, a record is maintained of the best solution found during the entire search. This solution is called the *incumbent*, and it is reported as the approximately optimal solution to the optimisation problem at hand at termination of the search.

In this paper, we generate the initial (or first current) solution either randomly or by using the LPTF rule, and again we employ displacement moves as described in Section 3.2, but with the added provision that the move should not be the reversal of any move recorded in the tabu list.

**Table 1: Suitable values for the parameters of the tabu search in the context of the PMCPSP**

No of jobs	No of iterations	Tabu tenure
1–10	100	10
11–20	200	15
21–30	300	20
31–40	400	25
41–50	500	30

On experimentation with randomly-generated instances of the PMCPSP, it was found that an effective choice of values for the tabu tenure  $\tau$  and number of search iterations  $I_{max}$  is the easily extendable trend shown in Table 1. These values seem to deliver good results.

### 3.4 Simulated annealing

The method of simulated annealing was first proposed by Kirkpatrick *et al.* [9] in 1983 as a simulation model to describe the physical process of annealing condensed matter within the realm of metallurgy. This method is also an iterative search procedure in which a single, current solution is maintained during each iteration. In contrast with the deterministic method of tabu search, however, simulated annealing is a stochastic search process. Non-improving moves are also allowed – as in the method of tabu search. However, such moves are only accepted with probability

$$P = e^{-\Delta_{obj}/T_w}, \quad (1)$$

where  $\Delta_{obj}$  denotes the absolute value of the amount by which the objective function would deteriorate when moving from the current solution to the non-improving solution, and where  $T_w$  is the value of a parameter – called the *temperature* – during temperature stage  $w \in \{0, 1, 2, \dots\}$ , which controls the search progression. The acceptance probability in (1) is known as the *Metropolis acceptance criterion* and decreases as  $\Delta_{obj}$  increases or as  $T_w$  decreases. The initial temperature  $T_0$  is selected relatively large, and is then decreased in stages according to a so-called *cooling schedule* to allow the search to initially explore the solution space (much as a random search would do), but so that it becomes gradually harder to accept non-improving moves later during the search (*i.e.*, so that it resembles a randomised improving local search more towards the end of the search, thereby exploiting the vicinity of high-quality solutions) [17].

Busetti [4] states that a good value for the initial temperature is such that approximately 80 per cent of all non-improving moves are accepted at the start of the search. Such an initial temperature may be estimated by conducting a trial search during which all non-improving moves are accepted. The initial temperature for the full search may then be taken as

$$T_0 = \frac{\Delta_{obj}^+}{\ln 0.8}, \quad (2)$$

where  $\Delta_{obj}^+$  denotes the average of the changes in objective function values as a result of accepting non-improving moves during the trial search.

The number of iterations spent by the search in temperature stage  $w$  is determined by a Markov chain of length  $L_w$ . As Busetti [4] states, the value of  $L_w$  ( $= L$ , say) should ideally depend on the optimisation problem at hand, rather than being a function of  $w$ . It would, in fact, make sense to require a minimum of  $A_{\min}$  move acceptances during any temperature stage before lowering the temperature, where  $A_{\min}$  is a pre-specified parameter. As  $T_w$  approaches zero, however, non-improving moves are accepted with decreasing probability and so the number of trials expected before accepting  $A_{\min}$  moves is expected to increase without bound as the search progresses, no matter the value of the positive integer  $A_{\min}$ . A suitable compromise is, therefore, to terminate the search once  $L$  moves have been attempted or  $A_{\min}$  moves have been accepted (whichever occurs first), for some positive integers  $L$  and  $A_{\min}$  satisfying  $L > A_{\min}$ . Following the rule of thumb proposed by Dreo *et al.* [5], we take  $L = 100 N$  and  $A_{\min} = 12 N$ , where  $N$  is some measure of the number of degrees of freedom in the optimisation problem. Based on numerical experimentation involving random instances of the MPCPSP, we choose  $N = n$  in this paper, since this value seems to yield relatively good results.

The well-known geometric cooling update rule

$$T_{w+1} = \beta T_w, \quad w = 0, 1, 2, \dots \quad (3)$$

is adopted in this paper, where the *cooling parameter*  $\beta$  is required to assume a value close to, but smaller than, one. As originally proposed by Kirkpatrick *et al.* [9], we employ the value  $\beta = 0.95$  here.

According to Busetti [4], the final temperature for the cooling schedule should be taken as that temperature at which the search ceases to make significant progress. We adopt the approach suggested by Dreo *et al.* [5], where the search is considered to have ceased making significant progress when it has failed to accept any moves during three consecutive temperature stages, at which point the search is terminated.

Finally, the same incumbent solution reporting protocol is adopted for the method of simulated annealing in this paper as that described for the method of tabu search above.

#### 4 TEST RESULTS

As described in Section 2, the set-up times in the PMCPSP test instances of Avalos-Rosales *et al.* [2] are uniformly distributed and were drawn independently of one another. These set-up times, therefore, do not possess the special colour commonality characteristic, which is inherent to the PMCPSP. Hence we cannot reliably use their test instances for comparing the efficacies of the (meta)heuristic solution methods described in Section 3 in the context of the PMCPSP.

We consequently generated ten instances of print job sets for the PMCPSP, each containing 24 jobs and each job requiring at least two, and at most four, colours from the universal colour set

$$C = \{1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e\}. \quad (4)$$

For each job, a print volume was drawn from a uniform distribution in the range 1–1000 kg. These job sets are listed in Table 2. From these ten instances of job sets, we created twenty hypothetical instances of the PMCPSP by supposing that three printers are available to process the jobs in each case, one operating at a speed of 4.5 kg/min and two operating at a speed of 8 kg/min, with all three printers having either  $b_k = 4$  or  $b_k = 6$  colour cartridges, for  $k = 1, 2, 3$ . Finally, we assumed that it takes  $a = 30$  minutes to empty, wash, and refill a single colour cartridge.

The results in Table 3 were obtained on implementation of the three metaheuristic solution procedures of Section 3 (improving local search, tabu search, and simulated annealing) for the PMCPSP instances described above. The first twenty lines of the table contain results for the case where each printer has only  $b_k = 4$  colour cartridges and the metaheuristics are either initialised according to the LPTF rule (lines 1–10) or randomly (lines 11–20). The last twenty lines of the table similarly contain results for the case where each printer has  $b_k = 6$  colour cartridges and the metaheuristics are either initialised according to the LPTF rule (lines 21–30) or randomly (lines 31–40).

**Table 2: Ten instances of print job sets for the PMCPSP, each containing 24 jobs, with each job requiring at least two, and at most four, colours from the universal colour set  $C$  in (4). The numbers in brackets are the volumes of the print jobs (in kg).**

	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	Instance 6	Instance 7	Instance 8	Instance 9	Instance 10
1	955b (534)	132 (833)	9b3c (748)	1b (347)	85 (399)	434d (820)	7bbc (44)	39 (970)	ec (134)	41 (898)
2	Ca (761)	21 (332)	11 (957)	a6cb (279)	56 (808)	ba (256)	dc (469)	abb7 (110)	4a9e (997)	2a4c (37)
3	dc (415)	8c2 (658)	7223 (380)	65a (78)	6ba (842)	5cbd (759)	54c (463)	91 (500)	d34 (57)	89 (227)
4	d1e (962)	75 (679)	88 (715)	39 (799)	a84 (951)	85 (357)	92a (539)	35e (119)	8d8 (366)	e65 (101)
5	b19 (525)	e8 (705)	bb (469)	e2e9 (244)	7d4 (749)	92 (579)	4e (58)	671 (275)	6b97 (765)	76 (891)
6	9a4 (299)	67 (758)	b29 (904)	94d (903)	4615 (891)	c9 (635)	66 (648)	e43 (438)	66 (39)	81c7 (252)
7	c9 (830)	529 (208)	764e (16)	b4 (439)	9948 (762)	7eb (188)	36 (215)	864 (344)	373 (632)	3b4 (377)
8	4ae4 (911)	27 (928)	6b45 (697)	849 (398)	2862 (858)	8a1 (485)	a99 (242)	68 (642)	1ca4 (275)	6b (324)
9	ea1 (107)	b4 (873)	2a5 (945)	d2 (79)	76e (699)	526 (900)	34 (45)	9612 (424)	e47d (871)	c9 (436)
10	2c4 (101)	215c (257)	27 (186)	4ec (901)	8923 (592)	6b7b (359)	5cd (584)	b4 (145)	d96c (501)	198a (294)
11	65 (296)	74 (236)	7e46 (543)	bc6 (493)	36 (19)	42e (174)	143 (388)	ac (43)	3bad (896)	3d6 (634)
12	6433 (980)	73d (45)	828 (282)	63 (327)	b16 (559)	5c (619)	54d (982)	e81 (801)	b7b8 (778)	71bc (165)
13	6a5 (411)	d58 (755)	da7a (558)	3a8 (100)	98 (312)	35 (796)	99 (570)	4a6 (330)	7d (312)	81 (422)
14	392 (208)	6e (635)	a7 (757)	71 (992)	db9 (330)	79 (494)	26c4 (851)	d36 (206)	e8e7 (385)	2b1 (977)
15	4b6 (906)	95 (924)	5b (224)	bdc (289)	314 (733)	25 (319)	c7c4 (397)	b25 (550)	2d9d (309)	aa33 (1000)
16	99 (920)	4e (480)	bc (519)	13 (109)	5ac1 (952)	ec89 (628)	4eed (900)	d5 (784)	d1c4 (768)	d234 (695)
17	8ad (98)	58 (29)	ca (973)	ed9 (189)	39de (484)	59 (902)	73a (651)	36 (954)	de (53)	d4 (325)
18	8d7a (23)	6cca (539)	62ec (876)	41cb (146)	ae1 (15)	bab (768)	618 (854)	e98 (325)	53 (540)	2cc3 (831)
19	756 (514)	576a (997)	1338 (402)	78c8 (333)	59c1 (74)	88 (550)	75 (375)	1c (899)	99 (130)	29 (624)
20	4e (56)	8c (423)	4ac (107)	69 (208)	56 (565)	5e (416)	a981 (13)	e7b6 (245)	3975 (89)	359 (944)
21	67 (391)	849 (428)	d65 (188)	691 (529)	752 (734)	716 (583)	e565 (315)	87 (603)	e6cc (307)	21 (313)
22	49 (475)	d65 (681)	74d9 (219)	eee (207)	b9a (573)	33 (655)	9a4 (236)	132 (767)	9a9 (357)	677c (734)
23	ea8 (157)	84 (151)	d9e (390)	ce43 (410)	cbab (417)	9a7 (96)	ee4 (577)	e4 (546)	13 (841)	664 (707)
24	b7b (108)	d73 (364)	d44 (331)	5a22 (17)	14 (774)	a31 (337)	6977 (770)	74c (891)	6c (436)	cc (70)

Based on the results in Table 3, the method of simulated annealing clearly outperforms the other two algorithms, yielding the best results in 33 out of the 40 cases (82.5 per cent). The method of tabu search yields the best results in five out of the 40 cases (12.5 per cent), while the improving local search yields the best results in only two out of the 40 cases (5 per cent). The method of tabu search is the next-best method, because it yields the second-best results in 29 out of the 40 cases (72.5 per cent). The method of simulated annealing yields the second best results in seven out of the 40 cases (17.5 per cent), while the improving local search yields the second-best results in only four out of the 40 cases (10 per cent).

**Table 3: Solution statistics obtained when applying the metaheuristics of Section 3 to the ten instances of the PMCPSP in Table 2. All makespans are measured in minutes, and all execution times are given in seconds.**

	Local Search			Tabu search			Simulated annealing		
	Initial Make-span	Final Make-span	Execution Time	Initial Make-span	Final Make-span	Execution Time	Initial Make-span	Final Make-span	Execution Time
1	1280.9	1019.0	12848	1280.9	969.6	41891	1280.9	870.0	31656
2	1257.4	1028.5	7453	1257.4	992.9	42562	1257.4	953.8	525656
3	1414.4	1102.4	3844	1414.4	1022.2	44250	1414.4	983.3	95214
4	1378.2	1104.7	7845	1378.2	1005.7	40034	1378.2	984.3	32781
5	1454.9	1153.5	7656	1454.9	1121.5	44672	1454.9	1112.4	27016
6	1370.4	1014.3	5625	1370.4	995.8	43047	1370.4	975.6	87500
7	1415.8	943.1	7641	1415.8	895.3	41796	1415.8	902.3	5750
8	1150.6	1057.1	1922	1150.6	948.0	43859	1150.6	990.8	4703
9	1397.8	989.9	5203	1397.8	951.8	42657	1397.8	988.9	4391
10	1319.8	1128.0	6485	1319.8	994.8	43515	1319.8	934.4	49359
1'	1175.9	987.5	5784	1175.9	956.3	36520	1175.9	902.6	45852
2'	1172.8	1018.4	6109	1172.8	978.5	47265	1172.8	914.8	32219
3'	1325.5	1065.1	8495	1325.5	996.6	47627	1325.5	954.3	49359
4'	1345.6	896.5	8523	1345.6	863.4	46523	1345.6	837.2	69045
5'	1394.9	1102.6	8515	1394.9	1093.5	49625	1394.9	1000.6	48171
6'	1265.0	1006.3	8452	1265.0	978.0	48489	1265.0	952.7	62079
7'	1175.8	885.5	7687	1175.8	853.4	39975	1175.8	795.0	55749
8'	1060.6	941.1	5438	1060.6	913.6	47797	1060.6	880.6	70828
9'	1153.4	979.3	7743	1153.4	932.0	42367	1153.4	903.1	200741
10'	1295.4	1101.1	2401	1295.4	965.6	50102	1295.4	960.3	63373
1*	1396.3	1120.3	9001	1365.5	895.8	45053	1745.1	823.6	50132
2*	1478.4	1079.3	6344	1794.0	986.1	42969	1326.0	931.8	45696
3*	1802.0	1038.6	8969	1300.5	990.0	43672	1554.1	925.9	38741
4*	1297.0	1010.5	8501	1341.5	1201.3	42010	1401.2	956.0	603241
5*	1612.1	1180.4	5765	1805.8	1113.3	44235	1524.0	1118.3	28641
6*	1750.7	1067.3	5750	1432.6	987.4	42672	1658.6	895.0	59087
7*	1293.5	914.4	12125	1637.6	929.6	42172	1745.1	888.4	50140
8*	1275.9	956.0	12703	1275.9	974.6	43735	1532.2	958.8	43609
9*	1614.4	986.8	11578	1778.9	913.1	41515	1687.5	933.6	38500
10*	1257.6	1092.0	3203	1887.8	958.3	43437	1380.7	936.5	34203
1 <sup>†</sup>	1183.9	904.0	12848	1243.9	904.6	43250	1508.2	872.7	23250
2 <sup>†</sup>	1170.3	1002.8	6984	1248.6	986.1	43360	1614.0	930.5	40390
3 <sup>†</sup>	1419.6	1099.3	4531	1627.8	1004.1	43203	1584.5	825.1	55265
4 <sup>†</sup>	1702.3	1152.3	4123	1587.2	952.0	41120	1650.3	902.3	38701
5 <sup>†</sup>	1619.1	1180.4	5766	1805.8	1113.3	44281	1524.0	1018.3	28594
6 <sup>†</sup>	1750.7	1067.3	5781	1432.6	987.4	42625	1602.0	805.7	60012
7 <sup>†</sup>	1293.5	914.4	12062	1637.6	929.6	42219	1108.6	817.3	35523
8 <sup>†</sup>	1275.9	956.0	12671	1532.2	992.0	43109	1550.2	966.3	28610
9 <sup>†</sup>	1439.6	970.9	7734	1537.8	938.6	43485	1338.0	890.4	36578
10 <sup>†</sup>	1258.4	1034.5	4422	1289.1	974.7	43359	1855.6	921.3	55766
	Best	2/40		Best	5/40		Best	33/40	
	2 <sup>nd</sup>	4/40		2 <sup>nd</sup>	29/40		2 <sup>nd</sup> Best	7/40	
	Best			Best					
	Worst	34/40		Worst	6/40		Worst	0/40	

Line *i*: Instance *i* in Table 2 where each printer has 4 cartridges & algorithms were initialised by the LPTF rule.

Line *i*' : Instance *i* in Table 2 where each printer has 4 cartridges & algorithms were initialised randomly.

Line *i*\*: Instance *i* in Table 2 where each printer has 6 cartridges & algorithms were initialised by the LPTF rule.

Line *i*' : Job set *i* in Table 2 where each printer has 6 cartridges & algorithms were initialised randomly.

The advantage of the improving local search, however, is that it yields results in a fraction of the execution time required by the methods of both tabu search and simulated annealing (but often becomes trapped at an inferior, locally optimal solution).

An advantage of the tabu search method is its remarkable consistency in execution times (an average of 43,651.4 seconds with a standard deviation 6.0 per cent of this value). This is due to the stopping criterion adopted ( $I_{\max} = 300$  and  $\tau = 20$  according to Table 1). In contrast, the improving local search exhibits the superior average execution time of 7273.9 seconds, but with a standard deviation 38.6 per cent of this value, while the method of simulated annealing achieves an average execution time of 73903.0 seconds with a standard deviation 160.4 per cent of this value.

Finally, use of the LPTF rule as initialisation procedure seems to yield better results than a random initialisation for the improving local search and tabu search; but for the method of simulated annealing, the opposite is true.

## 5 DECISION SUPPORT SYSTEM

We designed a DSS in Microsoft Excel to solve instances of the PMCPSP approximately. Although the execution times typically achieved by Excel macros are far inferior to what can be achieved, for example, in C or C++, Microsoft Excel was nevertheless chosen as the computing platform because it was anticipated that production engineers – tasked with solving instances of the PMCPSP in practice – will typically have access to, and be familiar with, this environment. Moreover, it would be very easy for such engineers to provide input data directly into Excel, rather than having to create text files or files in other formats containing these data.

Although the results of the previous section indicate that the method of simulated annealing initialised randomly seems to be the best algorithm and initialisation mechanism combination for finding high-quality solutions to the PMCPSP, we decided, for the sake of flexibility, to incorporate all three metaheuristics of Section 3 into our DSS implementation, along with both initialisation mechanisms (random initialisation and initialisation by the LPTF rule). The DSS user has the freedom to choose the algorithm and initialisation mechanism (s)he would like to use when solving an instance of the PMCPSP.

SETUP DATA		SEARCH CODE KEY	INITIAL SEQUENCE KEY
SEARCH SETUP		L = Local search	R = Random assignment
Search code		S = Simulated annealing	L = Largest-processing-time-first rule
Initial sequence code		T = Tabu search	
JOB SETUP		<div style="text-align: center;"> <input type="button" value="Step 1: Load Setup Data"/>  <input type="button" value="Step 2: Load Job and Printer Data"/>  <input type="button" value="Step 3: Run the Search"/>  <input type="button" value="Step 4: Clear Data"/> </div>	
Number of print jobs			
Setup time (min)			
PRINTER SETUP			
Number of printers			

Figure 1: Input screen of the PMCPSP DSS in Microsoft Excel

The user is presented with the input screen shown in Figure 1 on initialisation of the DSS in Excel. In this screen, the user is required to provide input values in the fields labelled *Search code*, *Initial sequence code*, *Number of print jobs*, *Set-up time*, and *Number of printers*. The first two fields have to be populated with the letters L/S/T (representing the three solution methods of local improving search, simulated annealing, and tabu search, respectively) and R/L (representing the method of initialisation as either random or by the LPTF rule, respectively).

The four buttons labelled *Steps 1–4* then have to be selected (in that order) to solve an instance of the PMCPSP. On selection of the *Step 1* button, the user is presented with the input matrix in Figure 2 where the speeds (in kg/min) and cartridge magazine sizes of the printers can be specified. The number of printer columns in this matrix corresponds to the number of printers specified by the user in the initialisation window.

PRINTER DATA	Printer 1	Printer 2	Printer 3
Speed (kg/min)			
Magazine size			

Figure 2: Input matrix for printer data in the PMCPSP DSS

On selection of the *Step 2* button, the user is presented with the job set input matrix shown in Figure 3. In this matrix, each job must be assigned a number or unique identifier, and be associated with a string of characters representing colours required for the job as well as the print volume of the job (in kg).

JOB DATA		
Job Number	Colour Code	Volume (kg)

Figure 3: Input matrix for job set data in the PMCPSP DSS

Once all the input data have been provided by the user, the *Step 3* button may be selected. The specified solution procedure is then launched, yielding an approximate solution in a print schedule output window, as shown in Figure 4. The columns of this output window contain the jobs to be sequenced (in that order) on the various printers.

FINAL PRINTER SEQUENCE		
Printer 1	Printer 2	Printer 3

Figure 4: Print schedule output window of the PMCPSP DSS

Associated printing-schedule performance-measure statistics – such as the total processing time of jobs assigned to each machine, the number of ink cartridges that have to be washed for each machine, the total set-up time incurred on each machine, and the completion time of each machine – are also returned in a final results window, as shown in Figure 5.

FINAL RESULTS	Printer 1	Printer 2	Printer 3
Processing time			
Setup instances			
Setup time			
Makespan			

Figure 5: Final results window of the PMCPSP DSS

Although the user is also able to specify algorithmic parameters (such as the number of iterations  $I_{\max}$  and tabu tenure  $\tau$  for the tabu search, or the cooling parameter  $\beta$ , and the number of moves to be attempted  $L$  and/or accepted  $A_{\min}$  during each temperature stage of the method of simulated annealing), it is not necessary here to set out in detail how this is done in the DSS.

## 6 REALISTIC CASE STUDY

In this section we apply the DSS of Section 5 to a case study involving real PMCPSP data obtained from a printing company in the Western Cape Province of South Africa to demonstrate the practical workability of our approximate solution approaches to realistic instances of the problem.

An industrial data set comprising 149 printing jobs – each requiring (at most) eight colours from a universal colour set containing the 34 colours in Table 4 – was obtained. This data set is shown in Table 5.

The printing company has five high-speed printers at its disposal: one that can accommodate eight colours simultaneously and print at a speed of 5.833 kg/min, two that can each accommodate six colours simultaneously and print at a speed of 4.722 kg/min, and two that can each accommodate four colours simultaneously and print at a speed of 3.514 kg/min. The fastest of these machines may be seen in Figure 6. It takes approximately  $a = 30$  minutes to empty, wash, and refill a single ink cartridge. At the time of initiating this research project, the company made no attempt to schedule print jobs in advance. Instead, jobs were processed in the order in which they were received from customers. We were informed, however, that – since they battled to balance the workload of machines and thought that they were incurring excessively large set-up times due to the essentially random sequencing of jobs on the five printers – the company wished, in future, to schedule print jobs in advance on a weekly basis. The reason for this was that the rate at which they received orders for print jobs would justify a weekly schedule.

We were informed that supervisors operating the printers worked in around-the-clock shifts from Monday to Saturday (hence a work week of  $24 \times 6 = 144$  hours = 8640 minutes, which is, therefore, an upper bound on the makespan of a feasible weekly printing schedule), and that a separate work crew performed maintenance on the printers on a Sunday.

**Table 4: Universal colour set C for the case study PMCPSP instance**

Code	Colour	Code	Colour	Code	Colour	Code	Colour	Code	Colour
1	Black 1	8	Cyan	f	Lime	n	Pink 2	u	Varnish
2	Black 2	9	Frost	g	Magenta	o	Pink 3	v	White 1
3	Blue 1	a	Gold 1	h	Maroon	p	Purple 1	w	White 2
4	Blue 2	b	Gold 2	i	Orange 1	q	Purple 2	x	Yellow 1
5	Brown 1	c	Green 1	j	Orange 2	r	Red 1	y	Yellow 2
6	Brown 2	d	Green 2	k	Peach	s	Red 2	z	Yellow 3
7	Buff	e	Grey	m	Pink 1	t	Silver		

**Table 5: Industrial job data set for the PMCPSP. The colours required are coded according to Table 5, and the numbers in brackets are print volumes (in kg).**

Job	Colours	Volume	Job	Colours	Volume	Job	Colours	Volume	Job	Colours	Volume
1	128gipry	(516)	39	13arvx	(125)	76	38gix	(2214)	113	1rv	(259)
2	128gipry	(519)	40	13arvx	(128)	77	3acuv	(3415)	114	1rv	(912)
3	129acivy	(103)	41	13arvx	(85)	78	3eirv	(242)	115	1rx	(272)
4	129acivy	(106)	42	158gir	(1147)	79	58gix	(3451)	116	1vx	(711)
5	129acivy	(147)	43	158grs	(2093)	80	acruv	(3656)	117	1vx	(718)
6	129acivy	(63)	44	158gry	(1042)	81	apqrv	(2421)	118	34v	(837)
7	137aruvx	(1021)	45	158gry	(1073)	82	12av	(98)	119	3rv	(610)
8	137aruvx	(1170)	46	158gry	(2665)	83	12vx	(348)	120	3rv	(615)
9	137aruvx	(723)	47	15cgrv	(722)	84	139v	(260)	121	3rv	(620)
10	18egrwvy	(133)	48	15grvx	(1166)	85	139v	(266)	122	3rv	(628)
11	18egrwvy	(164)	49	15grvx	(1171)	86	139v	(407)	123	13	(1700)
12	18egrwvy	(169)	50	15grvx	(1258)	87	15ir	(698)	124	1c	(1122)
13	18egrwvy	(227)	51	15grvx	(1362)	88	19cv	(660)	125	1i	(315)
14	18egrwvy	(227)	52	15grvx	(2122)	89	19fv	(230)	126	1i	(871)
15	18egrwvy	(90)	53	15grvx	(2329)	90	19hv	(407)	127	1p	(1374)
16	18egrwvy	(99)	54	15grvx	(4670)	91	19iv	(479)	128	1r	(1682)
17	34abrsuv	(4450)	55	15grvx	(4795)	92	19iv	(807)	129	1r	(396)
18	34agruxv	(11059)	56	15grvx	(735)	93	19pv	(258)	130	1v	(274)
19	34agruxv	(2998)	57	15grvx	(7599)	94	19vx	(268)	131	1v	(691)
20	128gpry	(577)	58	18grty	(1097)	95	1rvx	(135)	132	1v	(702)
21	128grvx	(226)	59	1aruvx	(1123)	96	34rv	(268)	133	1v	(706)
22	128grvy	(178)	60	34ckrv	(115)	97	3ruv	(697)	134	1x	(207)
23	128grvy	(202)	61	35airv	(1453)	98	3rvx	(925)	135	3i	(6073)
24	128grvy	(208)	62	35airv	(948)	99	12c	(1358)	136	3r	(610)
25	128grvy	(223)	63	12giv	(1842)	100	156	(1067)	137	1	(128)
26	1348gyz	(2564)	64	12gvx	(427)	101	156	(1091)	138	1	(1533)
27	1348gyz	(2584)	65	12rvx	(131)	102	156	(905)	139	1	(1638)
28	1348gyz	(2689)	66	139rv	(256)	103	1cd	(1129)	140	1	(229)
29	1348gyz	(2754)	67	19cdv	(849)	104	1cd	(609)	141	1	(507)
30	138gvwy	(370)	68	19crv	(365)	105	1cd	(713)	142	1	(828)
31	1acruvy	(1764)	69	19ctv	(710)	106	1cv	(250)	143	3	(5110)
32	3ciruvy	(627)	70	19hrv	(457)	107	1cv	(720)	144	c	(279)
33	12emnv	(612)	71	19ijv	(244)	108	1ij	(1163)	145	p	(179)
34	12grvy	(1068)	72	19pqv	(311)	109	1ij	(1202)	146	p	(633)
35	12grvy	(1909)	73	19ptv	(314)	110	1ij	(1209)	147	r	(328)
36	12mnov	(603)	74	1ruvx	(2898)	111	1ir	(617)	148	r	(352)
37	139crv	(349)	75	1ruvy	(1522)	112	1mv	(718)	149	r	(767)
38	13arvx	(110)									

We solved this instance of the PMCPSP (approximately) using the DSS described in Section 5, selecting simulated annealing (with parameter values as described in Section 3.4) and random initialisation as our solution methodology combination, based on the superior performance of this combination observed in the results of Section 4.

The search was initiated with a random solution exhibiting a makespan of 16053 minutes  $\approx$  11.15 days, and ran for 39572 seconds  $\approx$  11 hours (just over 900000 iterations) before reaching a state of thermal equilibrium. The progression of the various schedule makespans over the course of the search is shown in Figure 7. The incumbent solution, shown in Table 6, has a makespan of 8371 minutes  $\approx$  5.8 days, which is comfortably within the 6-day feasibility limit for a one-week schedule.



Figure 6: A high-speed flexico printer equipped with 8 colour cartridges that can print at a speed of 5.833 kg/min

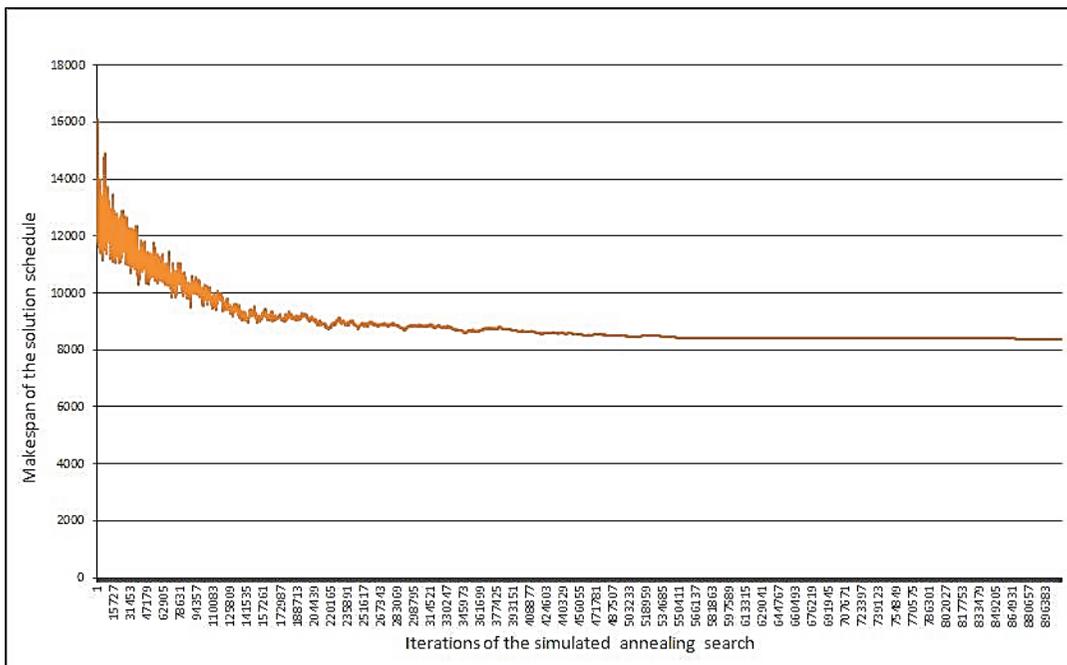


Figure 7: Progression of the simulated annealing search for the case study data set

## 7 CONCLUSION AND APPLICATION EXTENSIONS

A scheduling problem from the colour printing industry was considered in this paper. The problem, here called the *parallel machine colour print scheduling problem*, is to find an optimal assignment of print jobs to each of a set of colour printers, and a processing sequence for the set of jobs assigned to each printer. The objective is to minimise the makespan of the schedule to achieve a suitable balance between the workloads of the printers and the efficiencies of the job sequences assigned to the printers. A novel aspect of the problem is the way in which the printer set-up times associated with the jobs are job sequence-dependent: it is possible to exploit commonalities between the colours required for successive jobs on each machine.

**Table 6: Incumbent solution returned by the simulated annealing search. Jobs are numbered according to Table 5; the numbers in brackets are print job durations.**

	Printer 1	Printer 2	Printer 3	Printer 4	Printer 5
# of cartridges	8	6	6	4	4
Print speed (kg/min)	5.833	4.722	4.722	3.514	3.514
Job 1 (duration)	5 (25.2)	69 (150.4)	72 (65.9)	106 (71.1)	148 (100.2)
Job 2 (duration)	3 (17.7)	67 (179.8)	88 (139.8)	107 (204.9)	87 (198.6)
Job 3 (duration)	4 (18.2)	70 (96.8)	93 (54.6)	103 (321.3)	111 (175.6)
Job 4 (duration)	6 (10.8)	37 (73.9)	73 (66.5)	136 (173.6)	126 (247.9)
Job 5 (duration)	82 (16.8)	66 (54.2)	55 (1015.5)	125 (89.6)	101 (310.5)
Job 6 (duration)	71 (41.8)	68 (77.3)	57 (1609.3)	121 (176.4)	100 (303.6)
Job 7 (duration)	31 (302.4)	61 (307.7)	51 (288.4)	119 (173.6)	149 (218.3)
Job 8 (duration)	32 (107.5)	81 (512.7)	49 (248.0)	128 (478.7)	114 (259.5)
Job 9 (duration)	60 (19.9)	38 (23.3)	79 (730.8)	122 (178.7)	147 (93.3)
Job 10 (duration)	17 (762.9)	59 (237.8)	42 (242.9)	146 (180.1)	113 (73.7)
Job 11 (duration)	18 (1895.9)	74 (613.7)	45 (227.2)	108 (331.0)	118 (238.2)
Job 12 (duration)	19 (514.0)	52 (449.4)	44 (220.7)	135 (1728.2)	83 (99.0)
Job 13 (duration)	39 (21.4)	56 (155.7)	58 (232.3)	110 (344.1)	134 (58.9)
Job 14 (duration)	41 (14.6)	50 (266.4)	75 (322.3)	102 (257.5)	142 (235.6)
Job 15 (duration)	40 (21.9)	53 (493.2)	36 (127.7)	117 (204.3)	94 (76.3)
Job 16 (duration)	129 (67.9)	54 (989.0)	33 (129.6)	116 (202.3)	141 (144.3)
Job 17 (duration)	95 (23.1)	47 (152.9)	78 (51.2)	92 (229.7)	86 (115.8)
Job 18 (duration)	48 (199.9)	43 (443.2)	62 (200.8)	90 (115.8)	137 (36.4)
Job 19 (duration)	7 (175.0)	46 (564.4)	80 (774.2)	132 (199.8)	85 (75.7)
Job 20 (duration)	8 (200.6)	76 (468.9)	77 (723.2)	91 (136.3)	143 (1454.2)
Job 21 (duration)	9 (123.9)	64 (90.4)		89 (65.5)	120 (175.0)
Job 22 (duration)	28 (461.0)	35 (404.3)		112 (204.3)	109 (342.1)
Job 23 (duration)	29 (472.1)	34 (226.2)		133 (200.9)	123 (483.8)
Job 24 (duration)	27 (443.0)	63 (390.1)		138 (436.3)	145 (50.9)
Job 25 (duration)	26 (439.6)			131 (196.6)	144 (79.4)
Job 26 (duration)	30 (63.4)			84 (74.0)	127 (391.0)
Job 27 (duration)	12 (29.0)			97 (198.3)	139 (466.1)
Job 28 (duration)	130 (47.0)			98 (263.2)	99 (386.5)
Job 29 (duration)	14 (38.9)			96 (76.3)	105 (202.9)
Job 30 (duration)	16 (17.0)				104 (173.3)
Job 31 (duration)	10 (22.8)				115 (77.4)
Job 32 (duration)	15 (15.4)				124 (319.3)
Job 33 (duration)	13 (38.9)				140 (65.2)
Job 34 (duration)	11 (28.1)				
Job 35 (duration)	25 (38.2)				
Job 36 (duration)	22 (30.5)				
Job 37 (duration)	23 (34.6)				
Job 38 (duration)	21 (38.7)				
Job 39 (duration)	65 (22.5)				
Job 40 (duration)	24 (35.7)				
Job 41 (duration)	2 (89.0)				
Job 42 (duration)	1 (88.5)				
Job 43 (duration)	20 (98.9)				
Process time (min)	7174.4	7421.6	7471.0	7512.5	7728.5
Cartridges washed	39	31	30	28	21
Set-up time (min)	1170.0	930.0	900.0	840.0	630.0
Total time (min)	8344.4	8351.6	8371.0	8352.5	8358.5

Any PMCPSP model is, however, rich in application possibilities, in the sense that it admits a large variety of applications that, when first encountered, seem to be vastly different from decisions related to colour print scheduling. We mention four such application extensions in closing, but many others may also be found in the literature on flexible manufacturing systems.

An alternative application that fits naturally into the modelling framework of this paper is the problem of mailroom insert planning [1]. In this application, a number of lines of feeders are used to insert various pre-determined sets of distinct advertising brochures into envelopes destined for various categories of clients. Since each client category requires a different subset of brochures in their envelopes, the problem can be modelled as an instance of the PMCPSP in which each client-

required subset of brochures corresponds to a print job, each brochure corresponds to a different ink colour, each line of feeders corresponds to a printer, and each individual brochure feeder corresponds to an ink cartridge. Each brochure type change incurs a constant set-up time that is required to empty, clean, and reload a feeder; but if a brochure required for a specific client category were also required for the previous client category processed on a particular line of feeders, then the above set-up time could be avoided for that brochure. This gives rise to the same type of sequence-dependent set-up times that was encountered in the PMCPSP.

A further application alternative is encountered in a pharmaceutical packaging facility [10,13] where different types of pills are bundled together into patient- or region-specific boxes by flexible packaging units. Different types of pills are stored in containers and can be hooked up to the feeding holes of a packaging unit. The different pill combinations required by the patient types or regions correspond to a print job of the PMCPSP, each type of pill corresponds to a different ink colour, each packaging unit corresponds to a printer, and each individual pill container corresponds to an ink cartridge. There is an advantage to be gained in saved set-up time if two consecutively-processed pill combinations share the same type of constituent pill: this gives rise to a multiple-machine scheduling problem with sequence-dependent set-up times for which the models considered in this paper are applicable.

Our next example of an alternative application of PMCPSP models occurs in the assembly of *printed circuit boards* (PCBs) [16]. Flexible component placement machines are used to mount various electronic components, stored in large containers, on to a bare PCB. These machines typically contain a number of task-specific tools in a magazine. Since the machines are highly automated, configurable, and suitable for the assembly of a wide variety of PCB product types, the problem can be modelled as an instance of the PMCPSP in which each PCB product corresponds to a print job of the PMCPSP, each type of electronic component corresponds to a different ink colour, each component placement machine corresponds to a printer, and each electronic component container corresponds to an ink cartridge. In this context, the problem is known as the *Tool Switching Problem*, and again, the exploitation of tool commonality requirements for consecutive jobs is a central feature of the application.

As a last application extension, we mention a scheduling problem in the chemical manufacturing industry where various chemicals, each consisting of a particular set of constituents, are produced by feeding these constituents from containers to a number of mixing and reaction chambers via supply pipes [10,13]. Set-up cost is incurred by retooling and cleaning a pipe whenever a chemical constituent is changed. Here the various constituent combinations required for a specific chemical correspond to a print job of the PMCPSP, each constituent corresponds to a different ink colour, each mixing and reaction chamber corresponds to a printer, and each constituent container corresponds to an ink cartridge. In this case, the sequence-dependent set-up times between jobs are incurred as a result of having to fit new constituent containers and having to clean the feeding pipes. PMCPSP models are clearly also applicable in this case.

The three metaheuristics (a local improving search, a tabu search, and simulated annealing) designed, implemented, and tested in this paper for their ability to uncover high-quality (approximate) solutions to small instances of the PMCPSP are therefore also applicable to the application extensions mentioned above. It was found that the method of simulated annealing is the best of the three approximate solution approaches. An automated decision support tool for solving the PMCPSP approximately was also designed and implemented in Microsoft Excel [12]. This tool uses all three of the above metaheuristics to find good approximate solutions to the scheduling problem. Finally, it was demonstrated that the decision support tool developed here is capable of uncovering high-quality solutions to industrial-size instances of the PMCPSP.

Although the local search metaheuristics were able to uncover satisfactory solutions to the problem instances considered in this paper, it would be interesting to investigate whether superior solutions might be uncovered for even larger problem instances by incorporating population-based metaheuristics (such as a genetic algorithm or a particle swarm optimisation algorithm), or purpose-built heuristics within a branch-and-bound scheme, or a dynamic programming solution approach in the DSS.

## REFERENCES

- [1] Adjashvili, D., Bosio, S. & Zemmer, K. 2015. Minimizing the number of switch instances on a flexible machine in polynomial time, *Operations Research Letters*, 43(3), pp 317–322.
- [2] Avalos-Rosales, O., Alvarez, A.M. & Angel-Bello, F. 2013. A reformulation for the problem of scheduling unrelated parallel machines with sequence and machine dependent setup times, *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling*, pp 278–282.
- [3] Bruckner, P. 2007. *Scheduling algorithms*, 5<sup>th</sup> edition, Springer.
- [4] Buseti, F. 2003. *Simulated annealing overview*. Online reference available at: [http://www.cs.ubbcluj.ro/~csatol/mestint/pdfs/Buseti\\_AnnealingIntro.pdf](http://www.cs.ubbcluj.ro/~csatol/mestint/pdfs/Buseti_AnnealingIntro.pdf)
- [5] Dreo, J., Petrowski, A., Siarry, P. & Taillard, E. 2006. *Metaheuristics for hard optimization: Methods and case studies*, Springer.
- [6] Gray, A.E., Seidmann A. & Stecke, K.E. 1993. A synthesis of decision models for tool management in automated manufacturing, *Management Science*, 39, pp 549–567.
- [7] Glover, F. 1986. Future paths for integer programming and links to artificial intelligence, *Computers & Operations Research*, 13, pp 533–549.
- [8] Helal, M., Rabadi, G. & Al-Saken, A. 2006. A tabu search algorithm to minimize the makespan for unrelated parallel machines scheduling problem with setup times, *Operations Research*, 3, pp 182–192.
- [9] Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P. 1983. Optimization by simulated annealing, *Science*, 220, pp 671–680.
- [10] Laarhoven, P. & Zijm, W. 1993. Production preparation and numerical control in PCB assembly, *International Journal of Flexible Manufacturing Systems*, 5(3), pp 187–207.
- [11] Hertz, A., Laporte, G., Mittaz, M. & Stecke K. 1998. Heuristics for minimising tool switches when scheduling part types on a flexible machine, *IIE Transactions*, 30(8), pp 689–694.
- [12] Microsoft Office Products. 2016. Download Center. Online reference available at: <https://www.microsoft.com/en-za/download/details.aspx?id=10>
- [13] Mütze, T. 2014. Scheduling with few changes, *European Journal of Operational Research*, 236, pp 37–50.
- [14] Oerlemans, A.G. 1992. *Production planning for flexible manufacturing systems*, PhD Dissertation, University of Limburg, Maastricht.
- [15] Pinedo, M. 2002. *Scheduling: Theory, algorithms and systems*, 2<sup>nd</sup> edition, Prentice-Hall.
- [16] Raduly-Baka, C. & Nevalainen, O.S. 2015. The modular tool switching problem, *European Journal of Operational Research*, 242, pp 100–106.
- [17] Rardin, R. 1998. *Optimisation in operations research*, Prentice-Hall.
- [18] Sistemas de Optimización Aplicada, 2011. *Problem instances for unrelated parallel machines with sequence-dependent setup times and makespan criterion*. Online reference available at: <http://soa.iti.es/problem-instances>

## APPENDIX: MODEL FORMULATIONS

For the sake of completeness, this appendix contains the mathematical formulations of the 2006 PMCPSP model by Helal *et al.* [8] and the 2013 PMCPSP model by Avalos-Rosales *et al.* [2], starting with the earlier formulation.

Let  $\Omega_j$  denote the completion time of job  $j \in J$  and define the decision variables

$$v_{ijk} = \begin{cases} 1 & \text{if job } j \text{ is processed immediately after job } i \text{ on machine } k \\ 0 & \text{otherwise} \end{cases}$$

for all  $i, j \in J_0$  and  $k \in M$ , where  $J_0 = J \cup \{0\}$ . Here the dummy job 0 denotes the empty magazine state of a printer and the variables  $v_{0jk}$  and  $v_{i0k}$  represent jobs that are to be processed on printer  $k \in M$  at the start and end of the printing schedule, respectively. Similarly,  $s_{0jk}$  denotes the set-up time incurred when processing job  $j \in J$  first on machine  $k \in M$ . Then the objective in the PMCPSP formulation of Helal *et al.* [8] is to

$$\text{minimise } \Omega_{\max} \quad (5)$$

subject to the constraints

$$\sum_{i \in J_0 \setminus \{j\}} \sum_{k \in M} v_{ijk} = 1, \quad j \in J, \quad (6)$$

$$\sum_{i \in J_0 \setminus \{h\}} v_{ihk} - \sum_{i \in J_0 \setminus \{h\}} v_{hjk} = 0, \quad h \in J, k \in M, \quad (7)$$

$$\Omega_i + \sum_{k \in M} v_{ijk}(s_{ijk} + p_{jk}) + Z \left( \sum_{k \in M} v_{ijk} - 1 \right) \leq \Omega_j, \quad i \in J_0, j \in J, \quad (8)$$

$$\sum_{j \in J_0} v_{0jk} = 1, \quad k \in M, \quad (9)$$

$$\Omega_j \leq \Omega_{\max}, \quad j \in J, \quad (10)$$

$$\Omega_0 = 0, \quad (11)$$

$$\Omega_j \geq 0, \quad j \in J, \quad (12)$$

$$v_{ijk} \in \{0, 1\}, \quad i, j \in J, k \in M, \quad (13)$$

where  $Z$  is a large positive constant. The objective in (5) is to minimise the makespan of the schedule. Constraint set (6) ensures that each job is scheduled for processing exactly once, on one machine, while constraint set (7) ensures that each job is neither preceded nor succeeded by more than one other job. Constraint set (8), in which  $Z$  is a large positive integer, ensures that the job completion times are calculated correctly as the various jobs follow one another on each machine. Constraint set (9) ensures that exactly one job is scheduled first on each machine, while constraint sets (10)–(12) guarantee that the makespan of the schedule is calculated correctly. Finally, constraint set (13) enforces the binary nature of the decision variables.

In the model formulation (5)–(13), the makespan is linearised as the maximum of the completion times of all the jobs, as specified by constraint set (10). Avalos-Rosales *et al.* [2] found, however, that the lower bound on  $\Omega_{\max}$  produced by the linear relaxation of (5)–(13) is very weak. This prompted them to reformulate the model by linearising the makespan as the maximum of the spans of all the machines. Using the same notation as above, together with the additional symbol  $\Phi_k$  denoting the completion time of printer  $k \in M$ , the objective in the model of Avalos-Rosales *et al.* [2] is again to

$$\text{minimise } \Omega_{\max}, \quad (14)$$

but this time subject to the constraints

$$\sum_{i \in J_0 \setminus \{j\}} \sum_{k \in M} v_{ijk} = 1, \quad j \in J, \quad (15)$$

$$\sum_{j \in J_0 \setminus \{i\}} \sum_{k \in M} v_{ijk} = 1, \quad i \in J, \quad (16)$$

$$\sum_{j \in J} v_{0jk} \leq 1, \quad k \in M, \quad (17)$$

$$\sum_{j \in J_0 \setminus \{i\}} v_{ijk} - \sum_{h \in J_0 \setminus \{i\}} v_{hik} = 0, \quad i \in J, k \in M, \quad (18)$$

$$\Phi_j - \Phi_i + Z(1 - v_{ijk}) \leq \begin{matrix} s_{ijk} & i \in J_0, j \in J, i \neq j, k \in \\ + p_{jk}, & M, \end{matrix} \quad (19)$$

$$\Phi_0 = 0, \quad (20)$$

$$\sum_{i \in J_0 \setminus \{j\}} \sum_{j \in J} (s_{ijk} + p_{jk}) v_{ijk} \geq \Phi_k, \quad j \in J, \quad (21)$$

$$\Phi_k \leq \Omega_{\max}, \quad k \in M, \quad (22)$$

$$v_{ijk} \in \{0, 1\}, \quad i, j \in J_0, k \in M. \quad (23)$$

Constraint sets (15) and (16) respectively ensure that each job has exactly one predecessor and one successor. Furthermore, constraint set (17) specifies that at most one job should be scheduled first on each printer, while constraint set (18) embodies a collection of conservation of flow constraints that ensure that if a job is scheduled for processing on some printer, then a predecessor and a successor must exist for that job on the same printer. Constraint set (19) determines the correct processing order by establishing that, if  $v_{ijk} = 1$ , then the completion time of job  $j$  must be greater than the completion time of job  $i$  on machine  $k$ . Whenever  $v_{ijk} = 0$ , however, the corresponding constraint in (19) is redundant in view of the magnitude of  $Z$ . Constraint set (20) fixes the completion time of the dummy job (job 0) as zero and further serves, in conjunction with constraint set (19), to guarantee that the completion times of all jobs are positive. Constraint set (21) computes the completion time of the last job scheduled for processing on printer  $k$ , while constraint set (22) ensures a valid makespan for the schedule.