Int J Adv Manuf Technol (2006) 31: 328–334
DOI 10.1007/s00170-005-0188-5

ORIGINAL ARTICLE

Wen-Chiung Lee · Chin-Chia Wu · Peter Chen

A simulated annealing approach to makespan minimization on identical parallel machines

Received: 17 September 2004 / Accepted: 3 June 2005 / Published online: 13 December 2005

Abstract This paper addresses a makespan minimization scheduling problem on identical parallel machines. Several heuristic algorithms have been proposed to tackle the problem. In this paper, a very effective simulated annealing method is proposed to generate the near-optimal solution. Computational results demonstrate that the proposed heuristic is very accurate and that it outperforms the existing methods.

Keywords Scheduling · Parallel machines · Makespan · Simulated annealing

1 Introduction

Pinedo [1] pointed out that a bank of machines in parallel is a situation that is important from both a theoretical and a practical point of view. From a theoretical point of view, it is a generalization of the single machine and a special case of the flexible flowshop. From a practical point of view, it is important because the occurrence of resources in parallel is common in the real world. Also, techniques for studying machines in parallel are often used in decomposition procedures for multistage systems. In particular, the makespan becomes an objective of considerable interest when dealing with machines in parallel. In practice, one often has to deal with the problem of balancing the load on machines in parallel; by minimizing the makespan, the scheduler ensures a good balance. In this paper, we consider a makespan scheduling problem on identical parallel machines when preemption is not allowed. This problem is well known to be NP-hard. Several heuristic algorithms have been proposed to tackle the problem. However, their performances are not very consistent [2]. This observation, coupled with recent successes in simulated annealing methods to solve many combinatorial problems, motivates us to provide an effective simulated annealing method for this problem.

It is well known that the problem is NP-hard [3]. Thus, it is unlikely to obtain the optimal schedule through polynomial–time-bounded algorithms. Several polynomial–time heuristic algorithms have been proposed to tackle this problem. The first was probably the well known longest processing time first (LPT) rule proposed by Graham [4], and has received extensive attention since. Later, Coffman et al. [5] presented a MULTIFIT algorithm, which utilized the close relation between bin-packing and maximum completion time problems. Although the MULTIFIT heuristic does not always give better performance than the LPT rule, it has been shown by Friesen [6] and Yue [7] that its worst case performs better than that of the LPT rule. Lee and Massey [8] provided a COMBINE algorithm, which uses the LPT solution as the incumbent and has been further improved by the MULTIFIT method. Blazewicz et al. [9] showed that the LPT rule performs well for the single-criterion makespan problem. Riera et al. [10] proposed two improvements for the LPT rule using pairwise interchange techniques. The execution times of their algorithms are less than those of the MULTIFIT algorithm. However, the quality of their performances is similar to that of the MULTIFIT method. Fatemi Ghomi and Jolai Ghazvini [11] utilized the idea that the optimal makespan is obtained by a schedule such that the last assigned jobs on each machine are finished at the same moment if preemption is allowed. In other words, the variance of the completion times of the last job on each machine is zero. Thus, they tried to minimize a sum of ranges of machine finish times instead of the makespan. They utilized pairwise interchange techniques to yield maximum reduction in the selected function. Gupta and Ruiz-Torres [2] proposed a LISTFIT heuristic algorithm based on bin-packing and list scheduling, two common methods used to solve this problem. Their computational results showed that their heuristic outperformed the LPT, the MULTIFIT, and the COMBINE methods.

In this paper, a simulated annealing approach is proposed to tackle the makespan problem on identical

W.-C. Lee (✉) · C.-C. Wu · P. Chen
Department of Statistics, Feng Chia University,
Taichung, Taiwan, Republic of China
e-mail: wclee@fcu.edu.tw

parallel machines. The concept of a simulated annealing algorithm comes from the physical annealing of solids, and it has been successfully applied to combinatorial problems by Kirkpatrick et al. [12]. Since then, many researchers have used the simulated annealing method successfully to obtain good solutions to a variety of scheduling problems. In particular, Ruiz-Torres et al. [13], Park and Kim [14], Radhakrishnan and Ventura [15], and Kim et al. [16] applied the simulated annealing method to solve parallel machine problems. This method has the advantage that it can avoid being trapped in a local optimum by occasionally allowing "hill-climbing moves."

The remainder of this paper is organized as follows. In Sect. 2, we define the notation, formulate the problem, and describe several existing heuristic algorithms. In Sect. 3, we describe the proposed simulated annealing heuristic algorithm. Computational experiments are given in Sect. 4 and the conclusions is presented in the final section.

2 Problem description

Formally, suppose that $A=\{J_1,\ldots, J_n\}$ is the set of n jobs to be scheduled and $M=\{M_1,\ldots, M_m\}$ denotes the set of m identical parallel machines. All jobs are available for processing at time 0. Job J_i has processing time p_i, which can be processed on any machine M_j. Without loss of generality, we assume that $p_1 \geq p_2 \geq \ldots \geq p_n$. Once a job begins processing, it must be completed without interruption. Furthermore, each machine can process one job at a time, and there is no precedence relation between jobs. Let $S=(S_1,\ldots, S_m)$ denote a schedule where S_j is the subset of jobs assigned to machine M_j under S. In addition, let $C_j(S)$ denote the completion time of the last job on machine M_j under S; that is, $C_j(S) = \sum_{p_i \in S_j} p_i$. It is simplified as C_j while there is no confusion. Thus, the objective of this paper is to find an optimal schedule S^* such that:

$$\begin{aligned}&\max\{C_1(S^*),\ C_2(S^*),\ldots,\ C_m(S^*)\}\\&\quad\leq \max\{C_1(S),\ C_2(S),\ldots,\ C_m(S)\}\end{aligned}$$

Using conventional notation for describing scheduling problems, this objective is designated $P||C_{\max}$.

3 The simulated annealing approach

In this section, a simulated annealing (SA) approach is proposed to tackle the makespan problem on identical parallel machines. A brief description of the procedure is as follows. Given an initial sequence, a new sequence is created by random neighborhood generation. The new sequence is accepted if its makespan is better than that of the original sequence; otherwise, it is accepted with some probability which decreases as the process evolves. The most common functional form of the acceptance probability is $\exp(-\lambda \times \Delta L/L)$, where λ is the control parameter (temperature) and ΔL is the increase in the objective function, which is the makespan in this paper. Initially, the temperature is set to a high level so that a neighborhood exchange happens frequently in early iterations. It is gradually lowered using a predetermined cooling schedule so that, in later iterations, it becomes more difficult to exchange unless a better solution is obtained. The final solution found during the search process is an approximation to the optimal solution.

3.1 Implementation details

The implementation details for the four important elements of the SA approach in this study are described as follows:

1. *Initial sequence*: Since the performance of the LPT rule is acceptable and has been widely used as an initial sequence in other heuristics, it is chosen as the incumbent sequence in order to have a standard comparison.
2. *Neighborhood generation*: The solution space consists of all permutations of jobs. The SA algorithm attempts to minimize the objective function by examining the solution space and using moves from one permutation to another permutation. A permutation reachable by only one move is called a neighbor. The way of generating neighbors plays an important role on the efficiency of the SA method. In our implementation, we first randomly select a job from machine M_k that has the largest completion time and randomly choose another job from the remaining machines. Then, the new neighborhood sequence is generated by interchanging the positions of the two selected jobs.
3. *Acceptance probability*: In SA, solutions are accepted according to the magnitude of increase in the objective function and the temperature. After performing several pretests, the probability of acceptance is generated from an exponential distribution:

$$P(accept) = \exp\left(-\lambda \times \frac{\Delta L}{L}\right),$$

where λ is a control parameter (temperature) and ΔL is the change in the objective function. In addition, the method of changing the parameter λ at the kth iteration is given as:

$$\lambda = \frac{k}{\beta},$$

where β is an experimental constant (β=300 in this problem). If the makespan increases after the random pairwise interchange, the new sequence is accepted if:

$$P(accept) > r,$$

where r is a random number between 0 and 1.

Table 1 Summary of the computational experiments

	m	n	p
E_1	3, 4, 5	$2m$, $3m$, $5m$	$U(1, 20)$, $U(20, 50)$
E_2	2, 3, 4, 6, 8, 10	10, 30, 50, 100	$U(100, 800)$
E_3	3, 5, 8, 10	$3m+1$, $3m+2$, $4m+1$, $4m+2$, $5m+1$, $5m+2$	$U(1, 100)$, $U(100, 200)$, $U(100, 800)$
E_4	2	9	$U(1, 20)$, $U(20, 50)$, $U(50,100)$,
	3	10	$U(100, 200)$, $U(100, 800)$

4. *Stopping criterion*: The SA algorithm stops if it hits the lower bound or after $30n$ no-improvement iterations, where n is the number of jobs.

In summary, the steps of the proposed SA algorithm are as follows:

Inputs:
n, m, p_i for i=1,..., n

Step 1:
Construct an initial schedule by the LPT algorithm.

Step 2:
Compute $LB = \max\left(\left(\sum_{i=1}^{n} p_i\right)\Big/ n,\ p_1\right)$.

Step 3:
Distinguish S_a: $C_a = \max\{C_j,\ 1 \le j \le m\}$ and let $\alpha = C_a$.

Step 4:
If $\alpha = LB$, then the procedure terminates and the optimum schedule is obtained. Set K=1.

Step 5:
If $K>30n$, the procedure terminates and an approximate schedule with makespan α is obtained.

Step 6:
Select a job J_u randomly from S_a and select a job J_v from $N\backslash S_a$. Exchange the positions of these two jobs and compute the makespan α' of the new sequence.

Step 7:
Let $\Delta\alpha = \alpha' - \alpha$. If $\Delta\ \alpha<0$, accept the new sequence and go to step 4. Otherwise, go to step 8.

Step 8:
Compute $P(accept) = \exp\left(-\lambda \times \frac{\Delta\alpha}{\alpha}\right)$ and generate $r \sim U(0, 1)$. If $P(accept)>r$, accept the new sequence. Otherwise, retain the old sequence. Set K=K+1 and go to step 5.

4 Computational experiments

In this section, we present a comparison study of the LISTFIT approach [2], the pairwise interchange algorithm, denoted PI [11], and the proposed simulated annealing method, denoted SA in later sections. In order to evaluate their performances, a data-generating scheme is necessary. Fatemi Ghomi and Jolai Ghazvini [11] explored a variety of distributions and found that the PI heuristic performs poorest when processing times are uniformly distributed. Therefore, we focus on generating processing times from uniform distributions with different ranges. Gupta and Ruiz-Torres [2] provided several uniform-distribution-generating schemes, where some generating methods followed previous works and some problems were identified as difficult in their pilot experiments. In this paper, we

Table 2 Results for experiment E_1

			LISTFIT		PI		SA			
m	n	p	mean	max	mean	max	mean	max	LF/SA	PI/SA
3	6	$U(1,20)$	1.0748	1.3333	1.0753	1.3333	1.0753	1.3333	1/0	0/0
	9		1.0122	1.1111	1.0125	1.1111	1.0108	1.1111	12/17	0/5
	15		1.0007	1.0213	1.0000	1.0000	1.0000	1.0000	0/4	0/0
	6	$U(20, 50)$	1.0440	1.1552	1.0440	1.1552	1.0440	1.1552	0/0	0/0
	9		1.0158	1.0680	1.0111	1.0680	1.0105	1.0680	1/37	0/4
	15		1.0035	1.0158	1.0004	1.0059	1.0000	1.0000	0/47	0/8
4	8	$U(1,20)$	1.0844	1.4286	1.0869	1.4286	1.0854	1.4286	2/0	0/2
	12		1.0104	1.0667	1.0095	1.0667	1.0087	1.0667	7/14	0/2
	20		1.0005	1.0167	1.0000	1.0000	1.0000	1.0000	0/3	0/0
	8	$U(20, 50)$	1.0554	1.1343	1.0559	1.1379	1.0559	1.1379	2/0	0/0
	12		1.0159	1.0467	1.0072	1.0467	1.0069	1.0467	1/56	0/3
	20		1.0054	1.0214	1.0003	1.0060	1.0000	1.0000	0/70	0/6
5	10	$U(1,20)$	1.0778	1.3333	1.0802	1.3333	1.0786	1.3333	1/0	0/3
	15		1.0079	1.0741	1.0108	1.0741	1.0065	1.0741	10/15	0/14
	25		1.0019	1.0185	1.0000	1.0000	1.0000	1.0000	0/11	0/0
	10	$U(20, 50)$	1.0511	1.1818	1.0520	1.1818	1.0520	1.1818	3/0	0/0
	15		1.0204	1.0531	1.0087	1.0568	1.0073	1.0568	1/74	0/13
	25		1.0056	1.0166	1.0002	1.0062	1.0000	1.0000	0/72	0/4

Table 3 Results for experiment E_2

		LISTFIT		PI		SA			
m	*n*	mean	max	mean	max	mean	max	LF/SA	PI/SA
2	10	1.0030	1.0117	1.0021	1.0223	1.0015	1.0223	9/55	4/28
3		1.0157	1.0721	1.0111	1.0721	1.0090	1.0721	6/69	0/33
2	30	1.0001	1.0008	1.0000	1.0003	1.0000	1.0000	0/48	0/12
3		1.0010	1.0048	1.0002	1.0011	1.0000	1.0000	0/95	0/56
4		1.0029	1.0080	1.0005	1.0016	1.0000	1.0003	0/100	1/90
6		1.0081	1.0210	1.0019	1.0059	1.0004	1.0012	0/100	0/95
8		1.0136	1.0319	1.0048	1.0150	1.0015	1.0030	0/100	1/97
10		1.0190	1.0528	1.0127	1.0475	1.0071	1.0475	5/92	0/83
2	50	1.0000	1.0001	1.0000	1.0000	1.0000	1.0000	0/12	0/0
3		1.0002	1.0008	1.0000	1.0001	1.0000	1.0000	0/82	0/13
4		1.0009	1.0023	1.0001	1.0004	1.0000	1.0000	0/99	0/50
6		1.0027	1.0061	1.0004	1.0014	1.0000	1.0003	0/100	2/73
8		1.0052	1.0141	1.0009	1.0024	1.0002	1.0008	0/100	0/91
10		1.0072	1.0139	1.0016	1.0041	1.0005	1.0010	0/100	0/90
2	100	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0/0	0/0
3		1.0000	1.0001	1.0000	1.0000	1.0000	1.0000	0/51	0/0
4		1.0001	1.0004	1.0000	1.0001	1.0000	1.0000	0/83	0/1
6		1.0004	1.0019	1.0000	1.0001	1.0000	1.0000	0/100	0/11
8		1.0011	1.0030	1.0001	1.0004	1.0000	1.0002	0/100	0/31
10		1.0020	1.0043	1.0001	1.0007	1.0000	1.0002	0/100	0/47

Table 4 Results for experiment E_3: p~U (1, 100)

		LISTFIT		PI		SA			
m	*n*	mean	max	mean	max	mean	max	LF/SA	PI/SA
3	10	1.0111	1.0462	1.0113	1.0526	1.0075	1.0452	8/35	0/29
	11	1.0112	1.0468	1.0082	1.0414	1.0049	1.0252	3/60	2/38
	13	1.0062	1.0184	1.0038	1.0155	1.0008	1.0078	1/74	1/48
	14	1.0053	1.0201	1.0022	1.0101	1.0003	1.0094	1/72	0/40
	16	1.0032	1.0111	1.0015	1.0148	1.0000	1.0034	1/63	0/30
	17	1.0033	1.0163	1.0010	1.0071	1.0000	1.0000	0/67	0/28
5	16	1.0158	1.0448	1.0134	1.0769	1.0067	1.0388	6/70	0/52
	17	1.0137	1.0354	1.0074	1.0321	1.0020	1.0112	0/88	1/55
	21	1.0081	1.0333	1.0032	1.0151	1.0003	1.0057	2/90	1/51
	22	1.0068	1.0295	1.0031	1.0135	1.0000	1.0038	0/88	0/56
	26	1.0044	1.0233	1.0015	1.0084	1.0000	1.0000	0/78	0/37
	27	1.0035	1.0135	1.0008	1.0045	1.0000	1.0000	0/70	0/22
8	25	1.0136	1.0591	1.0110	1.0470	1.0038	1.0215	3/78	1/71
	26	1.0131	1.0497	1.0090	1.0429	1.0028	1.0311	0/85	0/68
	33	1.0072	1.0193	1.0030	1.0121	1.0001	1.0049	0/92	0/56
	34	1.0056	1.0134	1.0029	1.0135	1.0000	1.0050	0/84	0/55
	41	1.0042	1.0117	1.0017	1.0085	1.0000	1.0042	0/75	0/42
	42	1.0036	1.0101	1.0012	1.0045	1.0000	1.0000	0/75	0/31
10	31	1.0118	1.0513	1.0100	1.0513	1.0036	1.0513	1/74	0/73
	32	1.0111	1.0469	1.0076	1.0309	1.0018	1.0185	2/84	1/69
	41	1.0058	1.0205	1.0030	1.0105	1.0005	1.0057	0/81	0/49
	42	1.0051	1.0154	1.0031	1.0105	1.0002	1.0052	0/79	0/55
	51	1.0035	1.0116	1.0013	1.0051	1.0000	1.0000	0/74	0/33
	52	1.0035	1.0135	1.0012	1.0082	1.0000	1.0000	0/72	0/31

Table 5 Results for experiment E$_3$: p~U (100, 200)

		LISTFIT		PI		SA			
m	n	mean	max	mean	max	mean	max	LF/SA	PI/SA
3	10	1.0171	1.0704	1.0133	1.0704	1.0130	1.0704	0/41	0/8
	11	1.0199	1.0575	1.0110	1.0460	1.0104	1.0460	0/82	0/12
	13	1.0079	1.0329	1.0022	1.0204	1.0015	1.0204	0/81	0/31
	14	1.0073	1.0412	1.0021	1.0284	1.0012	1.0284	0/94	0/45
	16	1.0058	1.0235	1.0005	1.0038	1.0000	1.0000	0/94	0/31
	17	1.0050	1.0197	1.0005	1.0048	1.0000	1.0012	0/93	0/32
5	16	1.0267	1.0772	1.0080	1.0772	1.0067	1.0772	0/89	0/34
	17	1.0244	1.0608	1.0098	1.0589	1.0089	1.0589	0/95	0/27
	21	1.0279	1.0477	1.0012	1.0062	1.0000	1.0017	0/100	0/57
	22	1.0231	1.0493	1.0010	1.0032	1.0000	1.0016	0/100	0/55
	26	1.0238	1.0401	1.0004	1.0014	1.0000	1.0000	0/100	0/35
	27	1.0211	1.0346	1.0005	1.0013	1.0000	1.0011	0/100	0/36
8	25	1.0386	1.0751	1.0045	1.0508	1.0022	1.0508	0/99	0/68
	26	1.0310	1.0852	1.0049	1.0741	1.0031	1.0741	0/100	0/61
	33	1.0426	1.0648	1.0013	1.0050	1.0001	1.0017	0/100	0/64
	34	1.0376	1.0526	1.0010	1.0048	1.0001	1.0048	0/100	0/53
	41	1.0330	1.0489	1.0005	1.0014	1.0000	1.0000	0/100	0/39
	42	1.0312	1.0461	1.0004	1.0026	1.0000	1.0000	0/100	0/29
10	31	1.0445	1.0861	1.0040	1.0447	1.0016	1.0447	0/100	0/73
	32	1.0343	1.0711	1.0040	1.0621	1.0020	1.0621	0/100	0/72
	41	1.0485	1.0686	1.0010	1.0033	1.0000	1.0016	0/100	0/52
	42	1.0431	1.0683	1.0010	1.0033	1.0000	1.0016	0/100	0/54
	51	1.0335	1.0480	1.0005	1.0014	1.0000	1.0013	0/100	0/39
	52	1.0334	1.0445	1.0003	1.0013	1.0000	1.0000	0/100	0/23

Table 6 Results for experiment E$_3$: p~U (100, 800)

		LISTFIT		PI		SA			
m	n	mean	max	mean	max	mean	max	LF/SA	PI/SA
3	10	1.0145	1.0416	1.0114	1.0504	1.0082	1.0504	8/74	0/41
	11	1.0113	1.0403	1.0081	1.0494	1.0049	1.0304	8/77	0/56
	13	1.0082	1.0260	1.0046	1.0178	1.0015	1.0105	1/94	0/79
	14	1.0068	1.0172	1.0030	1.0134	1.0005	1.0028	1/98	0/84
	16	1.0051	1.0189	1.0016	1.0061	1.0001	1.0012	0/99	0/85
	17	1.0045	1.0143	1.0015	1.0095	1.0001	1.0004	0/100	0/88
5	16	1.0212	1.0560	1.0122	1.0441	1.0069	1.0261	5/91	0/71
	17	1.0174	1.0505	1.0088	1.0254	1.0039	1.0166	0/99	1/88
	21	1.0099	1.0250	1.0042	1.0220	1.0008	1.0023	0/100	1/94
	22	1.0110	1.0282	1.0033	1.0095	1.0006	1.0017	0/99	2/95
	26	1.0069	1.0249	1.0018	1.0055	1.0003	1.0014	0/100	0/92
	27	1.0067	1.0214	1.0016	1.0062	1.0003	1.0009	0/100	1/91
8	25	1.0197	1.0409	1.0105	1.0520	1.0047	1.0245	2/97	3/85
	26	1.0195	1.0428	1.0085	1.0277	1.0035	1.0277	3/97	2/92
	33	1.0112	1.0228	1.0028	1.0069	1.0009	1.0020	0/100	1/91
	34	1.0098	1.0216	1.0032	1.0102	1.0008	1.0017	0/100	1/98
	41	1.0074	1.0175	1.0015	1.0046	1.0004	1.0010	0/100	0/91
	42	1.0070	1.0213	1.0014	1.0031	1.0004	1.0009	0/100	1/89
10	31	1.0189	1.0410	1.0094	1.0217	1.0037	1.0107	0/100	0/94
	32	1.0192	1.0417	1.0077	1.0279	1.0028	1.0068	0/100	0/97
	41	1.0096	1.0216	1.0033	1.0103	1.0009	1.0021	0/100	0/95
	42	1.0097	1.0170	1.0031	1.0087	1.0008	1.0017	0/100	0/95
	51	1.0074	1.0141	1.0016	1.0039	1.0004	1.0013	0/100	1/96
	52	1.0076	1.0156	1.0014	1.0037	1.0004	1.0009	0/100	1/89

Table 7 Results for experiment E_4

			LISTFIT		PI		SA	
m	*n*	*p*	mean	max	mean	max	mean	max
2	9	U(1,20)	1.0006	1.0217	1.0002	1.0192	1.0000	1.0000
		U(20,50)	1.0019	1.0196	1.0003	1.0067	1.0000	1.0000
		U(1,100)	1.0015	1.0132	1.0024	1.0255	1.0004	1.0067
		U(50,100)	1.0008	1.0092	1.0002	1.0088	1.0000	1.0000
		U(100,200)	1.0010	1.0129	1.0003	1.0093	1.0000	1.0000
		U(100,800)	1.0020	1.0167	1.0012	1.0076	1.0003	1.0067
3	10	U(1,20)	1.0060	1.0370	1.0031	1.0417	1.0025	1.0417
		U(20,50)	1.0058	1.0261	1.0017	1.0254	1.0000	1.0000
		U(1,100)	1.0065	1.0323	1.0048	1.0638	1.0015	1.0160
		U(50,100)	1.0036	1.0251	1.0004	1.0155	1.0000	1.0000
		U(100,200)	1.0033	1.0246	1.0002	1.0098	1.0000	1.0000
		U(100,800)	1.0062	1.0312	1.0028	1.0273	1.0007	1.0154

replicate their generating schemes and add a few new ones based on our findings.

Four computational experiments were studied and each experiment investigated three variables; namely, the number of machines, the number of jobs, and the minimum and maximum values of the uniform distribution used to generate processing times. A summary of the four experiments is presented in Table 1. In E_1, the number of machines is set at three levels, namely, 3, 4, and 5. The number of jobs is also set at three levels, namely, $2m$, $3m$, and $5m$. The processing times are generated from the discrete uniform distribution at two levels, namely, $U(1, 20)$ and $U(20, 50)$. In E_2, the number of machines is set at six levels, namely, 2, 3, 4, 6, 8, and 10. The number of jobs is set at four levels, namely, 10, 30, 50, and 100. However, the number of machines is set at 2 and 3 when the number of jobs is 10. The processing times are generated from the discrete uniform distribution $U(100, 800)$. In E_3, the number of machines is set at four levels, namely, 3, 5, 8, and 10. The number of jobs is also set at six levels, namely, $3m+1$, $3m+2$, $4m+1$, $4m+2$, $5m+1$, and $5m+2$. The processing times are generated from the discrete uniform distribution at three levels, namely, $U(1, 100)$, $U(100, 200)$, and $U(100, 800)$. In E_4, the number of machines and the number of jobs are fixed at two-machine, nine-job problems and three-machine, ten-job problems, and the processing times are generated in the five levels considered in the three previous experiments. As a result, 120 experimental conditions were conducted. One hundred problems were generated for each condition, which yielded a total count of 12,000 problems.

All the heuristics were coded in Fortran and run on a Pentium 4 personal computer. In Tables 2, 3, 4, 5, 6, and 7, the average and the maximum values of the ratio of the makespan over the lower bound (LB) in step 2 above are given for all three heuristics, which are shown in columns 4–9. The last two columns present the relative performances of the SA heuristic with respect to the LISTFIT and the PI algorithms. For instance, a value of *c*/*d* in column LF/SA means that, out of 100 problems, there are *c* problems for which LISTFIT yields a better solution than SA, *d* problems for which SA performs better, and 100–*c*–*d* problems for which LISTFIT and SA yield the same solution. A similar interpretation applies to column PI/SA, which contains comparisons between PI and SA. Execution times are not given since they were less than 1 s, even for problems with 100 jobs. In Table 8, the number of times that the heuristics yielded the optimal solution is given for the three heuristics, respectively.

The results of experiment E_1 are given in Table 2 The performance of the LISTFIT algorithm is similar to that reported by Gupta and Ruiz-Torres [2]. In this experiment, it is found that the means and the maximum values of the ratios are quite similar for all three heuristics. There are no significant differences between the performances of PI and SA, although the solution of the SA heuristic is always slightly better than that of the PI method.

The results for experiment E_2 are given in Table 3. It is observed that the superiority of SA over the other two heuristics is more pronounced in E_2 than in E_1. When SA is compared to LISTFIT, it is found that SA yields a solution better than LISTFIT in 1,586 out of 2,000 tested problems, while SA yields a solution worse than LISTFIT only 20 times in this experiment. In other words, SA performs better 79.3% of the time, LISTFIT performs better only 1% of the time, and the two methods yield the same results

Table 8 Results for experiment E_4

m	*n*	p	LISTFIT	PI	SA
2	9	U(001,020)	97	99	100
		U(020,050)	78	95	100
		U(001,100)	73	70	93
		U(050,100)	83	97	100
		U(100,200)	80	92	100
		U(100,800)	45	58	88
3	10	U(001,020)	78	90	93
		U(020,050)	52	85	100
		U(001,100)	47	73	83
		U(050,100)	63	94	100
		U(100,200)	67	97	100
		U(100,800)	36	60	85

19.7% of the time. When it is compared to the PI algorithm, it is observed that SA performs better 45.05% (901/2,000), worse 0.4% (8/2,000), and ties 55.01% (1,091/2,000) of the time. In addition, it is noticed that SA always outperforms LISTFIT when m is 6 or more and n is 30 or more.

The results of experiment E_3 are given in Tables 4, 5, and 6. It is seen that the mean and the maximum ratios for all three heuristics are larger than those in E_1 and E_2, which was expected, since these problems are classified as difficult by Gupta and Ruiz-Torres [2]. However, the increments of the mean and maximum ratios for SA are relatively small compared to the other two heuristics. In addition, the superiority of SA over the other two heuristics is even more significant than in E_1 and E_2. In Table 4, when the processing times are generated from a discrete uniform distribution between 1 and 100, SA performs better 75.33% (1,808/2,400), worse 1.17% (28/2,400), and ties 23.5% (564/2,400) of the time compared to LISTFIT. On the other hand, SA performs better 40.17% (964/2,400), worse 0.29% (7/2,400), and ties 59.54% (1,429/2,400) of the time compared to PI. In Table 5, when the processing times are generated from a discrete uniform distribution between 100 and 200, SA performs better 94.5% (2,268/2,400) and ties 5.5% (132/2,400) of the time compared to LISTFIT. On the other hand, SA performs better 43.33% (1,040/2,400) and ties 56.67% (1,360/2,400) of the time compared to PI. In particular, it is noticed that SA never yields an inferior solution in these cases. In Table 6, when the processing times are generated from a discrete uniform distribution between 100 and 800, SA performs better 96.9% (2,325/2,400), worse 1.17% (28/2,400), and ties 2% (47/2,400) of the time compared to LISTFIT. On the other hand, SA performs better 86.5% (2,076/2,400), worse 0.63% (15/2,400), and ties 12.87% (309/2,400) of the time compared to PI. Overall, SA has the best performance among the three heuristics. In particular, its superiority is more pronounced for larger processing times.

The results for experiment E_4 are given in Tables 7 and 8. In Table 7, SA performs better than the other two heuristics as usual. In addition, it is observed that SA finds the optimal solution more often than LISTFIT and PI. It is noticed that problems become harder if the ranges and the values get larger. When the processing times are from U (100, 800), the performances of LISTFIT and PI drop dramatically, while the performance of SA is quite stable. Overall, SA reaches the optimal solution for 95.17% of the problems compared to 66.58% for LISTFIT and 84.17% for PI.

5 Conclusion

In this paper, a simulated annealing (SA) approach is presented to solve the identical parallel machine makespan problem. As the computational results show, the SA heuristic outperforms the LISTFIT and pairwise interchange (PI) algorithms for all experimental frameworks. In particular, the margin of superiority becomes much more significant for the problems classified as difficult in the Gupta and Ruiz-Torres [2] pilot study. Compared to LISTFIT and PI methods, it is also noted that the performance of SA is relatively stable for a variety of numbers of machines, jobs, and ranges of uniform distributions. Most importantly, the SA algorithm is very efficient even for large-sized problems, since the execution times are less than 1 s for all problems generated. Thus, the SA method is worth considering when faced with difficult scheduling problems.

References

1. Pinedo M (2002) Scheduling: theory, algorithms, and systems. Prentice-Hall, Upper Saddle River, New Jersey
2. Gupta JND, Ruiz-Torres AJ (2001) A LISTFIT heuristic for minimizing makespan on identical parallel machines. Prod Plan Control 12:28–36
3. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP completeness. Freeman, San Francisco, California
4. Graham RL (1969) Bounds on multiprocessor timing anomalies. SIAM J Appl Math 17:416–429
5. Coffman EG, Garey MR, Johnson DS (1978) An application of bin-packing to multiprocessor scheduling. SIAM J Comput 7:1–17
6. Friesen DK (1984) Tighter bounds for the multifit processor scheduling algorithm. SIAM J Comput 13:170–181
7. Yue M (1990) On the exact upper bound for the MULTIFIT processor algorithm. Ann Oper Res 24:233–259
8. Lee CY, Massey JD (1988) Multiprocessor scheduling: combining LPT and MULTIFIT. Discrete Appl Math 20:233–242
9. Blazewicz J, Ecker K, Pesch E, Schmidt G, Weglarz J (1996) Scheduling computer and manufacturing systems. Springer, Berlin Heidelberg New York
10. Riera J, Alcaide D, Sicilia J (1996) Approximate algorithms for the PC_{max} problem. Topology 4:345–359
11. Fatemi Ghomi SMT, Jolai Ghazvini F (1998) A pairwise interchange algorithm for parallel machine scheduling. Prod Plan Control 9:685–689
12. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
13. Ruiz-Torres AJ, Enscore EE, Barton RR (1997) Simulated annealing heuristics for the average flow-time and the number of tardy jobs bi-criteria identical parallel machine problem. Comput Ind Eng 33:257–260
14. Park MW, Kim YD (1997) Search heuristics for a parallel machine scheduling problem with ready times and due dates. Comput Ind Eng 33:793–796
15. Radhakrishnan S, Ventura JA (2000) Simulated annealing for parallel machine scheduling problem with earliness–tardiness penalties and sequence-dependent set-up times. Int J Prod Res 38:2233–2252
16. Kim DW, Na DG, Chen FF (2003) Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. Robot Comput-Int Manuf 19:173–181