Complexity on Parallel Machine Scheduling: A Review

D. K. Behera

Abstract The intended audience is scheduling practitioners and theoreticians as well as beginners in the field of scheduling. The purpose of this paper is to review the area of parallel machine scheduling (PMS) on issues of complexity. A critical review of the methods employed and applications developed in this relatively new area are presented and notable successes are highlighted. The PMS algorithms are discussed. We have given up-to-date information on polynomially type of problems based on non-preemptive criteria. It is shown that parallel machine makespan-minimization problem is NP-hard even for the two-machine problem. Moreover, the two-machine problem can be solved by the pseudo polynomial algorithm.

Keywords Parallel machine • Scheduling • Non-preemptive • Makespan • Polynomial • Complexity

1 Introduction

In the twenty-first century, the scheduling research area has made extraordinary advances in the development of techniques that enable improved solutions to practical problems [1, 2]. Notwithstanding the strengths of current techniques, the problems being addressed by current scheduling methods are generally NP-hard and solved only approximately [3]; there is scope for improvement in techniques for accommodating different classes of constraints and for optimizing under different sets of objective criteria [4–7]. The running time or time complexity of an algorithm expresses the total number of elementary operations, such as additions, multiplications, and comparisons, for each possible problem instance as a function of the size of the instance. The input size of a typical scheduling problem is bounded by the number of jobs n, the number of machines m and the number of bits to represent

D. K. Behera (✉)
Mechanical Engineering Department, Jadavpur University, Kolkata 700032, India
e-mail: dkb_igit@rediffmail.com

S. Sathiyamoorthy et al. (eds.), *Emerging Trends in Science, Engineering and Technology*, Lecture Notes in Mechanical Engineering,
DOI: 10.1007/978-81-322-1007-8_34,

the largest integer (the processing time, tardiness, the due date etc.,). An algorithm is said to be polynomial or a polynomial-time algorithm, if its running time is bounded by a polynomial in input size [3–7]. The most real-world problems are difficult to solve to optimality [3, 7–9]. So, Polynomial-time algorithms (PTA) was introduced by Cobham in year 1964 in deterministic machine models and later by Edmonds in 1965 saying that polynomial time represents efficient computation. An algorithm with rational input is said to run in polynomial time if there is an integer say k such that it runs in O (n^k) times where n is the given input size, and all numbers in intermediate computations can be stored with O (n^k) bits. We term it as a linear-time algorithm when the value of k becomes unit. PTA are persistently called "efficient" or "good". This big O notation is used to classify algorithms by how they respond (based on processing time requirements) to changes in input factor or size. Big O notation has utility when efficiency is looked into for analyzing algorithms. The number of hierarchy depends on the particulars of the machine model on which the algorithm runs, but different types of machines typically vary by only a constant factor in the number of hierarchy needed to execute an algorithm. In parallel machine scheduling (PMS), the relationships between time and space being the criteria of analysis of complexity, it is important to study for deterministic and non-deterministic problems [1–8]. Although traditional techniques such as complete enumeration, dynamic programming, integer programming, and branch and bound were used to find the optimal solutions for small- and medium-sized problems, they do not provide efficient solutions for the problems with large size [10, 11] (Table 1).

2 Notation and Classification

The use of $\alpha|\beta|\gamma$ notation given by Graham et al. [1, 2, 4, 8] for scheduling problems, where α is the machining environment, β is the set of restrictions, and γ is the objective function. Say, $\alpha = 1$ which denotes a single machine, while $\alpha = P$ is a parallel machine environment. For γ, C_j is the total completion time objective. Parallel Machines (PM):

Table 1 The time complexity of different types of problem seen in the literature

Sublinear	O(1)	Constant-time
	O(log log n)	Double logarithmic
	O(log n)	Logarithmic
	O($\log^k n$)	Polylogarithmic; *K is a constant*
	O(n^a)	a < 1 is a constant; e.g., O $\left(\sqrt{n}\right)$ for a = 1/2
	O(n/$\log^k n$)	k is constant
Linear	O(n)	
Super linear	O($n \log^k n$)	
	O(n^c)	Polynomial; $c > 1$ is a constant; e.g., O ($n\sqrt{n}$) for $c = 3/2$
	O(2^n)	Exponential
	O(2^{2^n})	Double exponential

means more than one machine is performing the same function. Table 2 gives the general notation/parameters considered in any scheduling problem. The PM can be:

- Identical: all machines have the same speed factors, and they can process all the jobs.
- Uniform: parallel machine system with different speed factor, and each job has a single operation.
- Unrelated: there is no relation between machines.

In a Parallel Machine Environment we consider a simple case say *Pm|rj, Mj|wjTj* which denotes a system with *m* machines in parallel. Job *j* arrives at release date *rj* and has to leave by the due date *dj*. Job *j* may be processed only on

Table 2 Notation/parameters for scheduling

Data	n	Number of jobs
	p_i ($p_{i;j}$)	Processing time of job i (on machine j)
	d_i	Due date of job i
	s_i	Desired starting time of job i
	r_i	Release date of job i
Variables	C_i	completion time of job i
	E_i	Earliness of job i: $E_\mathrm{i} = \max(0; d_\mathrm{i} - C_\mathrm{i})$
	L_i	Lateness of job i: $L_\mathrm{i} = C_\mathrm{i} - d_\mathrm{i}$
	T_i	Tardiness of job i: $T_\mathrm{i} = \max(0; C_\mathrm{i} - d_\mathrm{i})$
	U_i	Flag of tardiness for job i: $U_\mathrm{i} = 1$ if i is tardy and 0 otherwise
Constraints	Permu	In a flow shop problem the job sequence is the same for each machine
	Pmtn	Jobs can be interrupted and resumed later
	Nmit	No machine idle times are allowed
	Ssd	Sequence dependent setup times occur between jobs
Criteria	$f_{\max}/C_{\max}/L_{\max}/L_{\min}/T_{\max}/E_{\max}/\overline{C}\ \overline{C}\ w)\ /\overline{T}/\overline{T}\mathrm{w}/\overline{E}/\overline{E}\mathrm{w}/\overline{U}\ \overline{U}\mathrm{w}$	General maximum function strictly increasing with the completion times/maximum completion of jobs: $C_{\max} = \max i = 1::n(C_\mathrm{i})$/ maximum lateness of jobs: $L_{\max} = \max i = 1::n(L_\mathrm{i})$/minimum lateness of jobs: $L_{\min} = \min i = 1::n(L_\mathrm{i})$/ maximum tardiness of jobs: $T_{\max} = \max i = 1::n(T\mathrm{i})$/maximum earliness of jobs: $E_{\max} = \max i = 1::n(E_\mathrm{i})$/sum of completion times: $C = \mathrm{Pn}/(C\mathrm{i})$ (weighted sum)/sum of tardiness: $T = \mathrm{Pn}/(T_\mathrm{i})$ (weighted sum)/sum of earliness: $E = \mathrm{Pn}/(E_\mathrm{i})$ (weighted sum)/number of late jobs: $U = \mathrm{Pn}(U_\mathrm{i})$ (weighted sum)

one of the machines belonging to the subset *Mj*. If job *j* is not completed in time a penalty *wjTj* is incurred.

A Complexity Hierarchy may be in following order as per nature of problem.

1. 1||*C*max,
2. *P*2||*C*max,
3. *F*2||*C*max,
4. *Jm*||*C*max,
5. *FFc*||*C*max.
6. 1||*L*max,
7. 1|*prmp*|*L*max,
8. 1|*rj*|*L*max.
9. 1|*rj*, *prmp*|*L*max,
10. *Pm* || *L*max.

One standard approach for designing polynomial time approximation algorithms for a (difficult, NP-hard) optimization problem *P* is stated as follows:

(a) Relax some of the constraints of the hard problem *P* to get an easier problem *P′* (the so-called relaxation).
(b) Work out (in polynomial time) an optimal solution S_t for this easier relaxed problem *p′*.
(c) Translate (in polynomial time) the solution S_t into an approximate solution *S* for the original problem *P*.
(d) Analyze the quality of solution *S* for *P* by comparing its cost to the cost of solution *S′* for *P′*.

3 Scheduling Algorithms

Scheduling theory is concerned with the optimal allocation of scarce resources to activities over. Time horizon [4–6].The practice of this field dates to the first time two humans contended for a shared resource and developed a plan to share it without bloodshed. Algorithm may be defined as a succession of operations producing a solution to a problem through data manipulation. These data can be constants, or variables, or both kinds which can be arranged into data structures. Algorithms can be viewed as: precise type and approximate type. Precise analysis is quite tedious and at times unattainable to perform.

Thus scheduling algorithm arises [10, 11]. It is classified based on

1. Basic
   (a) as soon as possible
   (b) as late as possible
2. Time constrained
   (a) force directed
   (b) integer linear programming

(c) iterative refinement
3. Resource constrained
   (a) List based
   (b) static lists
4. Miscellaneous
   (a) Simulated annealing (SA)
   (b) path based

Figure 3.4 gives details of type of problem in a more elaborate way. Further heuristics can be classified into three types [12]. They are

- Index-development based on dispatching rules etc.
- Solution-construction like NEH.
- Solution-improvement (metaheuristics such as tabu search, SA etc).

Unfortunately, a simple, accurate, and time-invariant cost model for parallel machines does not exist The LPT, MULTIFIT, COMBINE, LISTFIT heuristics can also be applied in PMS for solving problems [1, 2, 13–16, 19–34] (Fig. 1).

An exact solution can be found by diverse methods of reduced enumeration, typically by a branch-and-bound algorithm. It is doubtful that an exact solution can be found by a polynomial-time algorithm. An algorithm is called an approximation algorithm if it is possible to found analytically how close the generated solution is to the optimum (either in the worst-case or on usual). The performance of a heuristic algorithm is usually analyzed experimentally, all the way through a number of runs using either generated instances or known benchmark instances.

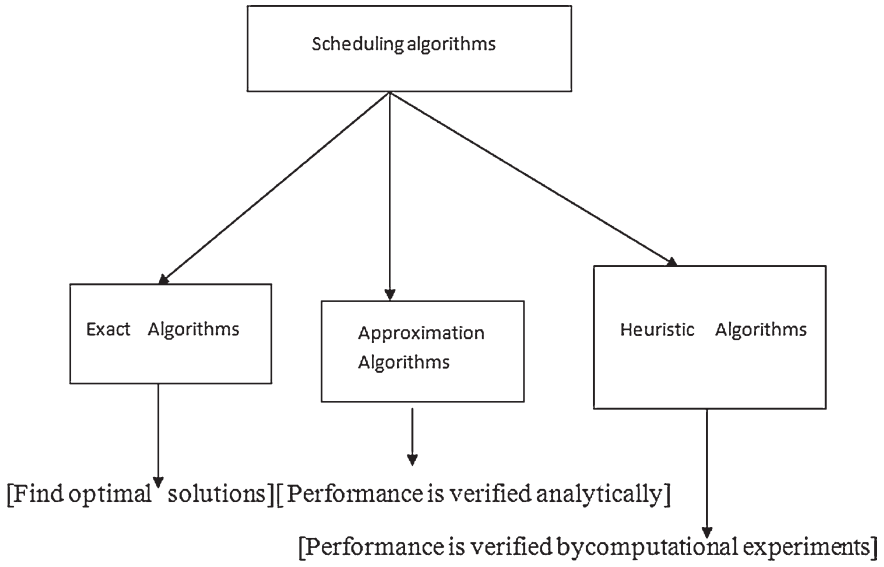

Fig. 1 The classification of scheduling algorithims

We define a ρ approximation algorithm to be an algorithm that runs in polynomial time and delivers a solution of value at most ρ times the optimum for any instance of the problem, i.e., $\frac{F(SA)}{F(SOPT)} \leq \rho$. The value of ρ is called a worst-case ratio bound. OPT stands for optimum value (Fig. 2).

In Fig. 3 *P* is polynomial time complexity problem and NP-hard belongs to non-deterministic polynomial. NP-Complete problems are the hardest problems in NP and *P* is subsets of NP.

As incase of PMS problem which is considered as hard optimization problems, finding this optimal solution is too hard because of the following reasons:

- Even with the best programming language available.
- Even with the fastest modern computer available.

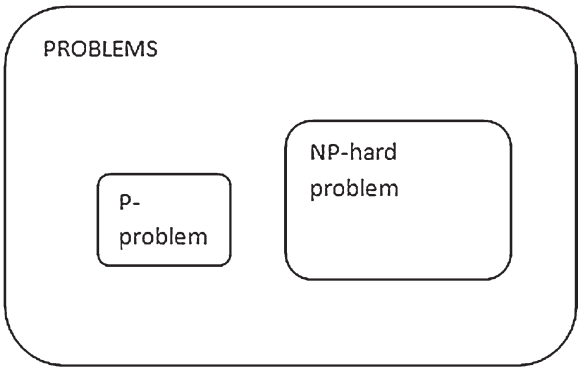

Fig. 2 Different types of problems as observed in scheduling

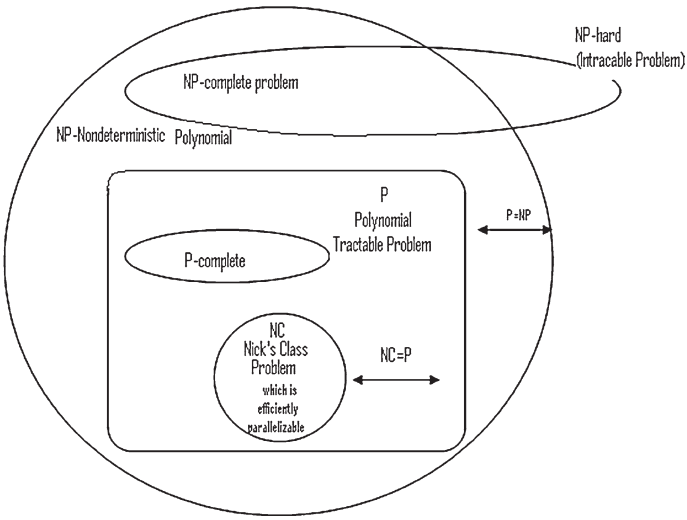

Fig. 3 A typical view of complexity classes and their relationships

- Even with the best programmer in the world.
- Even with the best and latest operating system.
- Even more years in the future.

The time complexity of an algorithm is the number of steps performed by this algorithm. For instance, our enumeration algorithm for the $P||C_{max}$ has time complexity big $O(m^n)$, since it evaluate m^n solutions. In any parallel identical machine scheduling denoted as $P||C_{max}$ the following parameters are looked into.

Given data/information	for each job, its duration
Constraint	perform all jobs
Decision	assign jobs to machines
Objective	end the last job as early as possible

4 Literature Review

Classical PMS considers a series of identical machines with a number of jobs and diverse processing times [1, 8, 10–34]. It assumes that the jobs are ready at time zero, and machines are endlessly available during the whole scheduling horizon. The simplest makespan problem arises in classical PMS when jobs are sequence independent and preemption is allowed. When preemption is permitted, the processing of a job can be interrupted and the remaining processing can be completed later, possibly on a different machine. When preemption of the jobs is permitted on all machines, the minimum makespan is obtained by: $M = \max\left[\sum_{j=1}^{n} p_j / m, \max_j \{p_j\}\right]$ where n is the number of jobs, p_j is the processing time of task j, and m is number of machines

It is shown that parallel machine makespan-minimization problem is NP-hard [1, 2, 6, 10] so far for the two-machine scheduling problem. Moreover, the two machine problem can be solved by the pseudo polynomial algorithm but solving problems with more than two machines is very tough and it becomes a Non-deterministic Polynomial-time hard problem which is NP-hard. Using some heuristics for generating one or more near-optimal individuals in the initial step can get better the last solutions obtained by meta-heuristic algorithms. Different criterion can be used for evaluating the efficiency of scheduling algorithms, the most important of which are makespan and flowtime [23]. Many researchers studied PMS problems in past. Cheng and Sin [8] and later Mokotoff [1] surveyed a PMS problem and Allahverdi et al. [13, 31] investigated a comprehensive review of setup time research for scheduling problems classifying into batch, non-batch, sequence independent, and sequence-dependent setup. Potts and Kovalyov [14] reviewed the literature on family scheduling models with single-machine, shop problems, and parallel machine. Brono et al. [17] proved that even a two-machine system for finding the weighted sum of flow times with an unequally weighted set of jobs is NP-hardness [19, 20, 27, 34]. A comparative analysis of PMS studied by Behera and Laha [18] indicates Listfit is better than all other algorithms.

5 Conclusions and Future Research

From the extensive literature review presented here, it can be concluded that interest in the area of PMS is growing. More direct search methods need to be explored for suitability to simulation optimization problems in PMS algorithms.

In this work, we consider a comprehensive survey of the PMS problems which is one of the most common and thoroughly studied problems in the scheduling literature. The papers surveyed include exact as well as heuristic techniques for many different multi-objective approaches. In numerous papers, SA is compared with Tabu search on scheduling problems and SA is observed to perform better than Tabu Search. SA forces the designer to either spend too much time or incur losses on the quality of solutions in scheduling problems. Different types of methods such as LPT, MULTIFIT, LISTFIT is studied in the literature [1, 2, 8, 13, 18, 19, 31]. Research in PMS will continue and is promising and there is scope for improvement.

References

1. Mokotoff E (2001) Parallel machine scheduling problems: a survey. Asia-Pacific J Oper Res 18:193–242
2. Baker KR, Trietsch D (2009) Principles of sequencing and scheduling. Wiley, New york
3. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP completeness. Freeman, San Francisco
4. Graham RL, Lawler EL, Lenstra JK, Kan AHGR (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math 5:287–326
5. Karp RM (1972) Reducibility among combinatorial problems: complexity of computer computations. Plenum Press, New York, pp 85–103
6. Lawler EL, Lenstra JK, Kan AHGR, Shmoys DB (1993) Sequencing and scheduling: algorithms and complexity. Handbooks in operations research and management science 4, logistics of production and inventory, North Holland, Amsterdam, pp 445–524
7. Papadimitriou CH (1994) Computational complexity, Addison-Wesley, Boston
8. Cheng T, Sin C (1990) A state-of-the-art review of parallel-machine scheduling research. Eur J Oper Res 47:271–292
9. Filippi C, Romanin-Jacur G (2009) Exact and approximate algorithms for high-multiplicity parallel machine scheduling. J Sched 12:529–541
10. Levner E (2007) Multiprocessor scheduling: theory and applications. Itech Education and Publishing, Vienna, p 436. ISBN 978-3-902613-02-8
11. Koulamas C (1993) Total tardiness problem: review and extensions. Oper Res 42:1025–1041
12. Chen CL (2009) A bottleneck-based heuristic for minimizing makespan in a flexible flow line with unrelated parallel machines. Comput Oper Res 35:3073–3081
13. Allahverdi A, Gupta JND, Aldowaisan T (1999) A review of scheduling research involving setup considerations. Omega 27:219–239
14. Potts CN, Kovalyov MY (2000) Scheduling with batching: a review. Eur J Oper Res 120:228–249
15. Lee H, Guignard M (1996) Hybrid bounding procedure for the workload allocation problem on parallel unrelated machines with setups. J Oper Res Soc 47:1247–1261
16. Weng MX, Lu J, Ren H (2001) Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. Int J Prod Econ 70:215–226

17. Bruno J, Sethi R (1978) Task sequencing in a batch environment with setup times. Found Control Eng 3:105–117
18. Behera DK, Laha D (2012) Comparison of heuristics for identical parallel machine scheduling. Adv Mater Res 488–489:1708–1712
19. Radhakrishnan S, Ventura JA (2000) Simulated annealing for parallel machine scheduling with earliness/tardiness penalties and sequence-dependent set-up times. Int J Prod Res 38:2233–2252
20. McNaughton R (1959) Scheduling with deadlines and loss function. Manage Sci 6:1–12
21. Kim DW, Kim KH, Jang W, Frank Chen F (2002) Unrelated parallel machine scheduling with setup times using simulated annealing. Rob Comput Integr Manufact 18:223–231
22. Koulamas C (1997) Decomposition and hybrid simulated annealing heuristics for the parallel-machine total tardiness problem. Nav Res Logistics 44:105–125
23. Izakian H, Abraham A, Snášel V Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments, collected from internet
24. Armentano VA, Yamashita DS (2000) Tabu search for scheduling on identical parallel machines to minimize mean tardiness. J Intell Manufact 11:453–460
25. 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
26. Li X, Yalaoui F, Amodeo L, Chehade H (2002) Metaheuristics and exact methods to solve a multiobjective parallel machines scheduling problem. J Intell Manuf 23(4):1179–1194
27. Brucker P, Sotskov YN (2007) Complexity of shop-scheduling problems with fixed number of jobs: a survey. Math Meth Oper Res 65:461–481
28. Johnson DS, Aragon CR, Mageoch LA, Schevon C (1989) Optimization by simulated annealing: an experimental evaluation; part 1, graph partitioning. Oper Res 37:865–892
29. Ghirardi M, Potts CN (2005) Makespan minimization for scheduling unrelated parallel machines: a recovering beam search approach. Eur J Oper Res 165(2):457–467
30. Martello S, Soumis F, Toth P (1997) Exact and approximation algorithms for makespan minimization on unrelated parallel machines. Discrete Appl Math 75:169–188
31. Allahverdi A, Ng CT, Cheng TCE, Kovalyov MY (2007) A survey of scheduling problems with setup times or costs, Eur J Oper Res
32. Chen CL, Chen CL (2009) Hybrid metaheuristics for unrelated parallel machine scheduling with sequence-dependent setup times. Int J Adv Manufact Technol 43(1-2):161–169
33. Tavakkoli-Moghaddam R, Mehdizadeh E (2007) A new ILP model for identical parallel-machine scheduling with family setup times minimizing the total weighted flow time by a genetic algorithm, IJE Transactions a: basics, vol 20(2)
34. Pinedo ML (2008) Scheduling—theory, algorithms, and systems. Prentice–Hall, Englewood Cliffs