Chapter

Approximation for Scheduling on Parallel Machines with Fixed Jobs or Unavailability Periods

Liliana Grigoriu

Abstract

We survey results that address the problem of non-preemptive scheduling on parallel machines with fixed jobs or unavailability periods with the purpose of minimizing the maximum completion time. We consider both identical and uniform processors, and also address the special case of scheduling on nonsimultaneous parallel machines, which may start processing at different times. The discussed results include polynomial-time approximation algorithms that achieve the best possible worst-case approximation bound of 1.5 in the class of polynomial algorithms unless P = NP for scheduling on identical processors with at most one fixed job on each machine and on uniform machines with at most one fixed job on each machine. The presented heuristics have similarities with the LPT algorithm or the MULTIFIT algorithm and they are fast and easy to implement. For scheduling on nonsimultaneous machines, experiments suggest that they would perform well in practice. We also include references to the relevant work in this area that contains more complex algorithms. We then discuss the main methods of argument used in the approximation bound proofs for the simple heuristics, and comment upon current challenges in this area by describing aspects of related practical problems from the automotive industry.

Keywords: multiprocessor scheduling, availability constraints, fixed jobs, uniform processors, worst-case approximation, nonsimultaneous machines, makespan, maximum completion time, unavailability

1. Introduction

The necessity to assign resources such as machines to jobs that need to be performed without interruption, where the time required for a machine to perform a certain job is known in advance, is a widely encountered problem. It can occur for example in production planning or when assigning landing and take-off stripes to planes in airports. Sometimes these resources become unavailable for predetermined periods of time, for example because of necessary maintenance. Minimizing the maximum completion time of all tasks is often considered as a goal, for example such that the workers who operate the machines can undertake other activities afterward or go home early. As a consequence, the problem of scheduling on multiple machines with predefined unavailability periods (downtimes) to minimize the maximum completion time, that is, the latest completion time of a job in a schedule, has been considered. A closely related problem, of scheduling with fixed jobs, where on each machine certain jobs have to be performed at predefined times, has also been considered. The difference between these two problems is in the meaning of the objective function: when scheduling with fixed jobs, the maximum completion time of the jobs must be at least the latest completion time of a fixed job, whereas the maximum completion time when scheduling in the presence of unavailability periods can occur before the end of an unavailability period. We consider the static nonresumable variant of the problem of scheduling with unavailability periods, where the downtimes are known for each machine before the schedule needs to be made, and where jobs that start executing before a downtime cannot resume execution after it.

In these problems, the job execution times are usually assumed to be given as an integer number of computing units such as clock cycles or of other suitable units such as time units. Similarly, the starttimes and endtimes of unavailability periods or of fixed jobs are assumed to be given as integer multiples of adequately chosen time units. We note that any problem with rational numbers as job durations and starttimes and endtimes of downtimes or of fixed jobs can be transformed into an equivalent problem where these entities are integers by multiplying them with an adequate factor, thus there is arguably no loss of generality in this assumption when considering the representation of any practical problem.

Both the problem of multiprocessor scheduling on fixed jobs and that of multiprocessor scheduling with unavailability periods are strongly NP-hard as they are more general than the strongly NP-hard multiprocessor scheduling problem (MSP), which has no downtimes or fixed jobs.

For scheduling with downtimes, it is NP-hard to find a schedule that ends within a given constant multiple of an optimal schedule even when scheduling on identical machines with at most one downtime on each machine. We discuss this in more detail in Section 4.2. To obtain approximation results for scheduling with unavailability periods in this context, assumptions about the downtimes were made such as the assumption that only a fraction of the processors can be unavailable at the same time [1, 2], or by comparing the generated schedule to the latest among the end of an optimal schedule or the latest end of a downtime, thus essentially considering scheduling with fixed jobs [3, 4].

To describe the performance of an approximation algorithm, we use the notion of a *worst-case approximation bound*. In this work, we call worst-case approximation bound of an algorithm A when applied to a scheduling problem SP the largest ratio between the maximum completion time of a schedule produced by A and the maximum completion time of an optimal schedule for a problem instance of SP.

For the problem of multiprocessor scheduling with fixed jobs to minimize the maximum completion time, even in the case where there is at most one fixed job on each machine, it has been shown in [5] that no polynomial algorithm can achieve a worst-case approximation bound that is less than 1.5 unless P = NP. Sharbrodt et al. [5] also give a polynomial-time approximation scheme (PTAS) for scheduling on a constant number of uniform processors with fixed jobs. Polynomial-time approximation algorithms for this problem that achieve the worst-case approximation bound of 1.5 were given for the general problem in [6]. For the case where there is at most one fixed job on each machine, significanlty simpler heuristics with lower time complexities resembling the largest processing time algorithm (LPT) [7] for identical processors and the MULTIFIT algorithm [8] for uniform processors also achieve this bound [3, 4].

The case where all downtimes are at the beginning of the processing time on all machines is called scheduling with nonsimultaneous machine available times, as the machines start processing jobs nonsimultaneously. For this problem, polynomial-time algorithms with constant worstcase approximation bounds exist.

For scheduling on identical nonsimultaneous parallel machines, MULTIFIT achieves a tight worst-case approximation bound of 24/19 (~1.2632) [9] and another algorithm achieves a bound of 5/4 [10], while in the case of scheduling on uniform nonsimultaneous parallel machines, a MULTIFIT variant has a worst-case approximation bound of 1.382 [11], which was shown by generalizing the bound obtained for MULTIFIT when scheduling on uniform processors in [12]. Experimental results suggest that for scheduling on nonsimultaneous uniform machines, the MULTIFIT variant from [11] is adequate for use in practice, as we discuss in Section [4].

In Section 2, we describe the ways in which the content of this work can be used. In Section 3, we introduce the algorithms LPT and MULTIFIT. In Section 4, we consider scheduling with unavailability periods, while first focussing on scheduling with nonsimultaneous machine available times in Section 4.1, and on the more general case where downtimes do not have to occur at the beginning of the schedule in Section 4.2. In Section 5, we present results on scheduling with fixed jobs. Section 6 contains the description of main techniques used in the worst-case approximation bound proofs and Section 7 contains concluding remarks and a discussion of some of the challenges in this area.

2. Contributions of this work

We next present ways to use the content of this work.

2.1 A deeper understanding

This work aims to provide a deeper understanding of multiple related problems that involve scheduling on parallel machines with fixed jobs or unavailability periods to minimize the maximum completion time. We explain why multiprocessor scheduling with at most one unavailability period on each machine does not have a polynomial-time approximation algorithm with a constant worst-case approximation bound unless P = NP, which is the main reason why results on this topic are hard to obtain.

Furthermore, we observe that most results in this area involve variants of LPT and MULTIFIT, and comment on the other results obtained. We also hope that this work will increase awareness of these results and of how they relate to each other.

2.2 Practical use of the heuristics

The heuristics presented and referenced in this work can be used directly in practice or for research purposes to solve the problems they address. The heuristics based on LPT and MULTIFIT are fast and easy to implement and some of them have best possible worst-case approximation bounds in the class of polynomial algorithms unless P = NP for the problems they address. In addition to worst-case approximation results, this work also highlights for some cases experimental insights into how the heuristics would perform in practice based on how they perform for randomly generated instances. As expected, they perform much better for such instances than in the worst case. Also, for some cases, references to more complex methods are provided in case the user prefers to use those.

2.3 Proof techniques

This work presents the main proof techniques used to obtain worst-case approximation bounds for LPT- and MULTIFIT-like heuristics when the aim is to minimize the maximum completion time. Thus, the interested reader is provided with a concise description of the tools that can be used to develop such proofs, and he or she may not have to read hundreds of pages in order to become aware of all of them or work with an expert in the area when developing such a proof. Even for people with expertise in the area, one or more of the ideas presented may be new and helpful.

3. The algorithms LPT and MULTIFIT

The algorithms LPT and MULTIFIT are among the most studied approximation algorithms for multiprocessor scheduling with or without unavailability periods or fixed jobs. In this section, we describe the basic versions of these algorithms for MSP, which can be stated as follows: given a set of m machines P and n jobs J find a non-preemptive schedule that ends at the earliest possible time. A non-preemptime schedule is an assignment of jobs to processors, together with an order in which the jobs on each processor are processed.

The algorithm LPT [7] works as follows:

Algorithm 1 The largest processing time algorithm (LPT)

```
1: Order jobs in nonincreasing order of their processing time.
2: In this order assign each job at the earliest possible time allowed by the schedule that exists when the job is
   assigned.
```

The algorithm MULTIFIT was first introduced by Coffmann Garey and Johnson in 1978 [8]. It uses binary search for the end of its resulting schedule and receives as input an accuracy ε with which it determines this schedule end. In each iteration it assigns a deadline and attempts to create a schedule that contains all tasks that ends at or before that deadline by using the bin packing algorithm first fit decreasing (FFD). If a feasible schedule is created, it decreases the deadline and otherwise it increases the deadline. This process is repeated until the difference between the current deadline and the previously considered deadline is less than ε. More formally, the algorithm is described as Algorithm 2.

The MULTIFIT algorithm results in a schedule with a maximum completion time that is within ε of the maximum completion time of the schedule that would result if the binary search for the deadline would be continued.

An example of a LPT-schedule and a MULTIFIT schedule for the same problem instance are presented in **Figures 1** and **2** respectively.

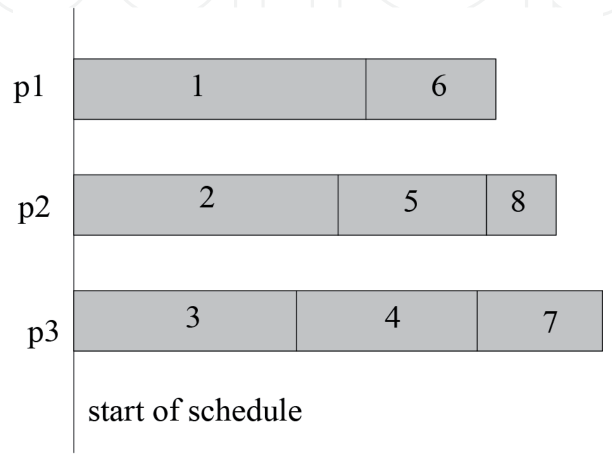

Figure 1.
A LPT schedule. The jobs are numbered according to the order in which they are considered. At start, when all processors are available at the same time, they are considered in the order p1, p2, p3 in this example.

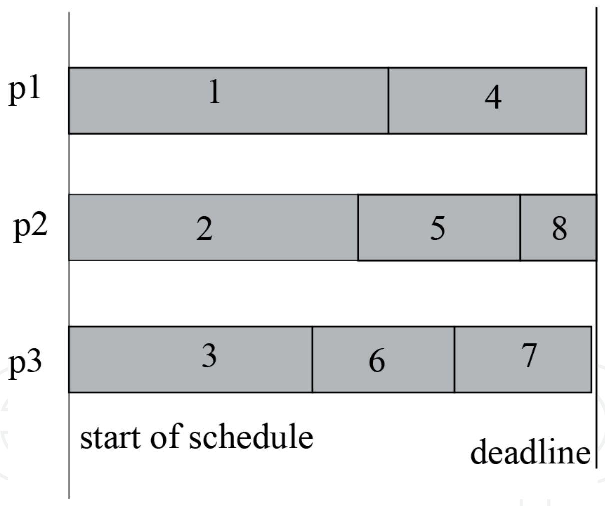

Figure 2.
A MULTIFIT schedule together with a possible deadline. The jobs are numbered according to the order in which they are considered. The processors are considered in the order p1, p2, p3 when generating the schedule.

Algorithm 2 The algorithm MULTIFIT

```
1: Order the jobs in non-increasing order of their duration.
2: Assign upper bound (ub) and lower bound (lb) for the end of schedule; (for example, lb = sum of job
   durations/number of processors, ub = sum of job durations).
3: Assign b = (ub + lb)/2 as deadline.
4: FFD: Assign tasks in non-increasing order on the first processor on which they fit while respecting the
deadline (the processors are considered in each iteration in the same order).
5: If all tasks are successfully scheduled decrease the upper bound: ub = b.
6: Else increase the lower bound: lb = b.
7: If ub − lb ≥ ε loop back to Step 3.
```

The MULTIFIT algorithm tends to produce more balanced schedules than LPT, and, as a consequence it tends to perform better when the aim is to minimize the maximum completion time. It also has a higher time complexity, as it tries to make a schedule about $\log_2(ub - lb)$ times, whereas the LPT algorithm only does that once. When the instances are prohibitively big and the schedule needs to be made in a short time, it may be indicated to use LPT or another list scheduling algorithm to schedule the jobs. This is because in practical situations, there may not be enough time to go through the list of jobs more than once while scheduling thousands of jobs and making sure that all required constraints are obeyed by the schedule. The reason why such big schedules are made is that the companies aim to estimate delivery times for their orders.

3.1 Time complexity of MULTIFIT

If the parameter ε of MULTIFIT is adequately set, for example as a computer clock cycle, the algorithm returns the best possible MULTIFIT schedule, that is, running it further would not result in a better schedule, as was commented upon in [4].

The binary search for the deadline within the MULTIFIT algorithm happens within $\log_2[(ub - lb)/\varepsilon]$ time, which is at most $\log_2(ub/\varepsilon)$, which is the size of ub in binary, assuming that the numbers for the upper bound and the lower bound do not change their representation during the execution of the algorithm and that they allow within their representation for an accuracy of ε. In Section 1, we mentioned that the job durations are usually given as integer multiples of an adequately chosen (time) unit; therefore, the end of any schedule is an integer, and thus, there is no point in making ε less than 1, in which case the MULTIFIT loop is repeated $\log_2(ub)$ times.

As a consequence, the number of times the MULTIFIT loop is called is polynomial in the size of the input, as any reasonable upper bound is at most the sum of the processing times of all jobs, which can be represented within at most the total number of bits used to represent all jobs. In [4], Grigoriu and Friesen also comment that if the upper bound is 2 years, the lower bound is 0 and the deadline is determined with an accuracy of 10^{-13} s, the MULTIFIT loop is called at most 40 times.

The time complexity of MULTIFIT is thus $O(n \log n + nm \log_2(ub))$, as the jobs need to be sorted according to their execution times in a non-increasing order in Step (1), and as in each iteration of the MULTIFIT loop, the algorithm looks for each job for the first processor on which it fits; thus, it will have to look at most at m processors. Recall that n is the number of jobs in the considered problem instance.

4. Scheduling with unavailability periods

In this section, we first present results for the case where all unavailability periods are at the beginning of the schedule. Then, we present results for the more general case where the unavailability periods can occur anywhere in the schedule.

4.1 Scheduling with nonsimultaneous machine available times

This section addresses the case where the processors may have unavailability periods at the start of their processing time. This situation is more general than the multiprocessor scheduling problem (where there are no fixed jobs or downtimes) and less general than the problems of scheduling with fixed jobs or with unavailability periods. As the less general multiprocessor scheduling problem is NP-hard, so are the problems of scheduling on identical machines with nonsimultaneous machine available times (NMSP: nonsimultaneous multiprocessor scheduling problem) and scheduling on uniform processors with nonsimultaneous machine available times (UNMSP) when minimizing the maximum completion time. Due to the NP-hardness of these problems, polynomial-time approximation algorithms like LPT and MULTIFIT and their variants have been studied for their solution. As before, we will continue to denote with the number of processors in the problem instance being considered with m.

4.1.1 Scheduling on identical nonsimultaneous processors

For NMSP, worst-case approximation bounds for LPT of $3/2 - 1/(2m)$ and for a modified version of LPT (MLPT) of 4/3 have been obtained by Lee [13]. The bound obtained by Lee in [13] was improved upon by Kellerer in [10], where a dual approximation algorithm with a worst-case approximation bound of 5/4 was presented.

When MULTIFIT is used for MSP, a deadline results in periods of equal duration in which jobs can be scheduled on each processor; thus the schedules resulting from using any ordering of processors in step (4) of MULTIFIT have the same maximum completion time. When considering NMSP, thus allowing for nonsimultaneous machines, the order in which processors are considered becomes relevant, as the period in which jobs can be executed on each processor corresponding to a deadline depends on the time the processor becomes available. MULTIFIT variants that address such situations usually order the processors in non-decreasing order of their periods in which jobs can be scheduled. Thus, in this case, the ordering is in non-increasing order of the times at which the processors become available.

A bound of 9/7 (about 1.2857) was obtained for MULTIFIT by Chang and Hwang [14]. In [10], Kellerer gives a problem instance of NMSP for which the approximation factor of its MULTIFIT schedule is 24/19 (about 1.2632). More recently, 24/19 was shown to be the exact worst-case approximation bound when using MULTIFIT for NMSP by Hwang and Lim [9]. By comparison, a tight worst-case approximation bound of 13/11 (about 1.18182) was shown by Yue [15] for MUTLIFIT when applied to MSP.

4.1.2 Scheduling on uniform nonsimultaneous processors

For the uniform multiprocessor scheduling problem with simultaneous machine available times (UMSP), that is, where processors execute jobs at different speeds, the amount of jobs that fit on a processor corresponding to a given MULTIFIT deadline depends on the speed of that processor. Usually, the slowest processor is considered to have a speed of 1, and for each job j, the time it would take to process it on this processor l_j is given. Thus, on a machine with speed 5, a job j needs a time of $l_j/5$ to be processed. As a consequence, the ordering in which the processors are considered in Step (4) of MULTIFIT is in most cases relevant to the maximum completion time of the resulting schedule. MULTIFIT for UMSP orders processors in each iteration before its Step (4) in non-decreasing order of the duration of the processing time on that processor times the speed of that processor [12, 16].

For UMSP, approximation bounds of 1.4 and 1.382 were obtained for MULTIFIT by Friesen and Langston [16] and by Chen [12] respectively. In [17], Burkard and He derive a worst-case approximation bound of $\sqrt{6}/2$ (about 1.2247) of MULTIFIT for UMSP with at most two processors, and show a better bound of $(\sqrt{2}+1)/2$ (about 1.2071) for the case where MULTIFIT is combined with LPT as an incumbent algorithm.

In [11], Grigoriu and Friesen show that bounds that apply to the MULTIFIT variants from previous work such as [12, 16, 17] where scheduling on two uniform processors is considered also apply to a slightly different proposed variant of MULTIFIT for UNMSP, LMULTIFIT, which was first proposed in [4] in a more general form. The difference between the MULTIFIT variants from [12, 16, 17] on the one hand and LMULTIFIT on the other hand is that in the latter, the choice of the initial upper and lower bounds is not given explicitly within the algorithm, and thus the worst-case approximation bound proofs are more general, as they work for any initial choices of upper and lower bounds for the duration of the resulting schedule. A first step in the proofs that the bounds hold in the more general case, where there are uniform nonsimultaneous parallel machines, was to show that LMULTIFIT obeys the bounds of the earlier MULTIFIT variants in the simultaneous machines case for the instances considered in those works.

Using LPT for UNMSP has been considered in [18], where worst-case approximation bound of 5/3 was shown in the general case, as well as a better bound for the case where there are only two machines.

For the case where the number of machines is constant, a PTAS exists for UNMSP [11], which was derived from a PTAS for scheduling on a constant number of uniform processors with fixed jobs from [5]. As the objective function for scheduling with fixed jobs that are at the beginning of the schedule and scheduling with unavailability periods that are at the beginning of the schedule differ, the PTAS from [5] can not be used unaltered to address UNMSP. To address UNMSP, the PTAS from [5] is first run for the transformed problem instance where the unavailability periods become fixed jobs, and then for all problem instances resulting from successively removing the machine with the latest end of a fixed job from the transformed instance [11]. This accounts for the cases where a number between 1 and $m - 1$ of processors start processing after the end of the optimal schedule.

In [19], a lower bound is derived for the end of an optimal schedule of an UNMSP instance, and using that bound approximation factors for LMULTIFIT schedules of randomly generated instances are determined. The reasonably extensive experiments described in [19] suggest that LMULTIFIT is a good option for solving UNMSP in practice, not only because of being fast and easy to implement, but also because it has very good approximation factors (less than 1.03) for the generated instances with an average of at least five jobs for each machine. In order to obtain the approximation factors, a lower bound for the end of the optimal schedule that can be calculated directly from the problem instance was used.

4.2 Multiprocessor scheduling with availability constraints

In this section, we consider the multiprocessor scheduling problem where downtimes can occur at any time during the scheduling horizon.

Surveys with focus on scheduling with availability constraints are given in [20–24]. Besides the makespan, the authors of these works survey work on various other objective functions such as total completion time, and also address additional variants of the problem, such as its resumable version.

Unless assumptions about the unavailability periods are made or unless P = NP, there is no polynomial-time approximation algorithm with a constant worst-case approximation bound for the problem of scheduling with unavailability periods to minimize the maximum completion time, since there is a polynomial-time reduction from the NP-hard problem of 3-Partition to the problem of finding a schedule that has a maximum completion time that is at most α times the end of an optimal schedule for this problem. We next outline such a reduction. Let X be an instance of 3-Partition, that is, a set of $3m$ positive integers, given with the purpose of finding out whether there is a partition of these numbers into m sets, such that the sum of the numbers in each set is the same for all sets. Let S be the sum of all numbers in X. The instance Y of scheduling with unavailability periods that we build is given as follows: there are m processors, each of which has an unavailability period of duration $(\alpha + 1)S$ that starts at time S/m, and the job durations in Y are the numbers in X. X is a yes-instance of 3-Partition if and only if in instance Y, the optimal schedule ends at time S/m. In such a situation, any approximation algorithm with worst-case approximation factor α for scheduling with availability constraints will find a schedule that ends at or before time $\alpha S/m$ which is less than $(\alpha + 1)S$. Thus, the found schedule must end before or when the unavailability periods start, at time S/m. In such a schedule, the sets of durations of jobs on each processor are a 3-Partition of X. Therefore, any polynomial-time approximation algorithm for scheduling with unavailability periods with a worst-case approximation factor α can be used to solve 3-Partition in polynomial time.

4.2.1 Scheduling on identical machines with unavailability periods

For resumable scheduling, where the execution of jobs may continue after a downtime that interrupted them, but where jobs cannot be preempted by the scheduling algorithm, and where one machine does not shut down and all other machines shut down at most once, Lee shows that the makespan of LPT is in the worst case $(m + 1)/2$ times the optimal makespan [25].

In [1], Hwang and Chang make the assumption that at most half of the machines are unavailable at any time, and show for this situation that the worst-case approximation bound of LPT is 2. In [3], it is shown that no polynomial algorithm can have a better bound than 2 for this problem unless $P = NP$. The result from [1] is generalized in [2] where it is shown that if at most $\lambda \in \{1, \dots, m-1\}$ machines may be unavailable at any time, LPT has a worst-case approximation bound of $1 + \frac{1}{2}\lceil m/(m-\lambda) \rceil$, and that this bound is tight for LPT.

4.2.2 Scheduling on uniform machines with unavailability periods

In [26], scheduling with at most one unavailability period on each machine is considered and exact algorithms are given for small problem instances. The authors consider separately the case of identical jobs, and also consider total completion time beside the makespan as an objective function. For larger instances, they propose an LPT-like algorithm, which assigns jobs in nonincreasing order of their processing time to the fastest machine on which they would finish being processed at the earliest time. They also do experiments on a total number of 68 generated instances where error margins of at most 5.6% are observed. They do not compare their heuristic to the previously proposed heuristic from [4], which we discuss in Section 5.1.2, which was also proposed for this problem, even though its worst-case approximation bound was shown for the objective function of scheduling with fixed jobs.

5. Scheduling with fixed jobs

The problem of scheduling with fixed jobs is given as a number of processors, where each processor may have jobs that must be executed during certain given periods on it, together with a number of other jobs which can be executed by any processor. As noted in Section 1, job durations or required execution times are expressed for example as a number of significant units such as clock cycles. For uniform processors this number represents the time needed by a job to be executed on the slowest processor. In case there are uniform processors, each processor also has a speed factor, by which the time needed by a job on the slowest processor is divided in order to obtain the time needed for the processor to execute the job.

As noted before, the problem of scheduling with fixed jobs differs from the problem of scheduling with unavailability periods in that its maximum completion time cannot be earlier than the latest completion time of a fixed job.

In [5], Scharbrodt et al. give a polynomial-time approximation scheme for scheduling on a constant number of uniform machines with fixed jobs. They also show that it is NP-hard to obtain a schedule that ends within a factor of less than 1.5 when scheduling on identical processors with at most one fixed job on each machine. Even though they do not specify their result in this way, their proof that no polynomial-time approximation algorithm can have a better worst-case approximation bound than 1.5 for multiprocessor scheduling with fixed jobs does not use the fact that there can be more than one fixed job on each machine, which implies the previous statement.

If all fixed jobs are at the beginning of the schedule, the results presented in Section 4.1 apply, as the optimal schedule of each problem instance of scheduling on nonsimultaneous machines can only potentially get worse when scheduling with fixed jobs is considered instead, and since the resulting schedule of an approximation algorithm ends later for scheduling with fixed jobs only if its maximum completion time is the completion time of a fixed job, which the optimal schedule also needs to execute.

We next consider the case where there is at most one fixed job on each machine in Section 5.1, and the case where there can be multiple fixed jobs on each machine in Section 5.2.

5.1 Scheduling with at most one fixed job on each machine

When scheduling on multiple processors with at most one fixed job on each machine, LPT and MULTIFIT variants have been shown to achieve a worst-case approximation bound of 1.5, which is best possible for this problem unless *P* = *NP*.

5.1.1 Same-speed processors

For scheduling on identical machines with at most one fixed job on each machine, an LPT-like algorithm, LPTX, was given in [3], for which a worst-case approximation bound of 1.5 was shown. Before running LPT, LPTX creates an order of processors, which is then used by LPT to break ties in case two processors become available at the same time. The ordered list of processors created before applying LPT is built in two steps:

1. All processors that have an unavailability period that is not at the beginning of the schedule are assigned to this list in non-decreasing order of the start of their downtime.

2. All other processors (that is, those which have the downtimes at the beginning of the scheduling period or have no downtime at all) are appended to the list built in the previous step in non-decreasing order of the times at which they can start executing jobs.

5.1.2 Uniform processors

In the more general case of scheduling on uniform machines with at most one fixed job on each machine, a MULTIFIT-like algorithm, LMULTIFIT, was given in [4], which achieves the worst-case approximation bound of 1.5. For a MULTIFIT variant to work in the presence of downtimes, it must be specified how it deals with the fact that there are more than one time interval in which jobs can be scheduled on one processor.

After a MULTIFIT deadline is assigned and before applying the bin packing algorithm FFD (see Section 3), LMULTIFIT orders all time intervals in which jobs can be scheduled in non-decreasing order of their *length*. Here, the length of a time interval is the duration of the time interval multiplied by the speed factor of the processor on which the interval occurs.

5.2 Scheduling with multiple fixed jobs on each machine

For scheduling on identical processors with fixed jobs, where the number of fixed jobs on a machine can be arbitrary, approximation algorithms that are much more complex than LPT and MULTIFIT were given in [27], with a worst-case approximation bound of 1.5 + ε, where ε is the parameter of a fully polynomial time approximation scheme (FPTAS) for the multiple subset sum problem, and in [6] with a worst-case approximation bound of 1.5, where a FPTAS for the multiple knapsack problem is used as a subroutine.

In [28], a very long proof is outlined that LMULTIFIT achieves a worst-case approximation bound of 1.5 when scheduling on identical processors with at most two fixed jobs on each machine. In [29], an algorithm using two MULTIFIT-like algorithms is shown to have a worst-case approximation bound of 1.625, which likely can be improved to 1.6 without excessive effort.

The time complexity of the MULTIFIT-like and LPT-like algorithms is significanlty lower than that of the algorithms from [6, 27], and they are also significantly easier to implement; however, there is little hope in our opinion that bounds better than 1.55 can be shown in the general case for such algorithms with proofs of reasonable length. Given such problems, the user must decide what is best suited for his or her needs.

6. Proof techniques

Worst-case approximation results in this research area about LPT and MULTIFIT variants mainly use proofs by contradiction in which some proof techniques appear very often. We next describe two techniques that we consider to be the most relevant, and then comment upon some other methods that are used relatively often. In the following, we call job lengths the durations of the jobs on the slowest processor.

6.1 Minimal counterexample

A very well-known proof method is that of assuming that there exists a minimal counterexample to a theorem T, that is, a problem instance for which the algorithm's schedule does not obey that theorem, with a minimal number of processors, of jobs, of downtimes (or of fixed jobs) that do not start at the beginning of the schedule, and/or of other quantities that can be minimized which are chosen to fit the statements to prove. A minimal counterexample exists whenever there is a counterexample, and thus, showing that it does not exist proves T. In order to define minimality among counterexamples, the author of the proof first chooses a partial order among the problem instances. An instance which is minimal according to this partial order among the instances for which a theorem T does not apply is called a minimal counterexample.

This method can be very powerful, because after assuming that a minimal counterexample does not obey a theorem T, many useful properties of a minimal counterexample can be derived from the fact that no *lesser* counterexample exists, which can ultimately lead to a contradiction.

Here, a lesser counterexample is a counterexample with less processors, or less tasks, or less downtimes, or with a job with a smaller length, depending on how the order of instances was defined. Showing that a minimal counterexample does not exist is usually significantly easier than developing a direct proof for T.

The theorem to prove could be that the worst-case approximation bound holds, but it can also be an intermediary result that is later used to prove the worst-case approximation bound. One could address instances that have a certain property first, and then show that the worst-case approximation bound holds for these, for example by using a minimal counterexample among these instances, and then do the same for all other instances.

Sometimes it is enough to define the partial order only using the number of processors [11], while in most cases, it is useful to include multiple characteristics of problem instances, such as all or a part of the characteristics enumerated above. In one situation, a minimal counterexample was defined to also have minimal job lengths, meaning that if in a minimal counterexample the length of one job is reduced, the resulting instance is not a counterexample [28].

6.2 Weighing arguments

For a problem instance that does not obey a worst-case approximation bound, there is a job (X) that is scheduled such that it crosses the bound to prove times the end of an optimal schedule (B) for LPT-like algorithms, or that can not be scheduled when the deadline is at B for MULTIFIT-like algorithms. If the order defined on instances includes the number of jobs, there is only one such job in a minimal counterexample to (for example) a theorem that the analyzed algorithm (A) generates only schedules that obey the bound. Otherwise, all jobs that would be scheduled afterward can be removed and a lesser counterexample could be obtained, resulting in a contradiction. As a consequence of how LPT and MULTIFIT work, X is the smallest job of the minimal counterexample in both cases.

The schedule S_A generated by an LPT-like algorithm A until a job would cross B and the schedule S_A generated by a MULTIFIT-like algorithm A if the MULTIFIT deadline is at B do not contain the job X. An optimal schedule, however, contains all jobs, including X.

Therefore, the optimal schedule has more total execution time than S_A. Also, if nonzero weights are assigned to each job, the optimal schedule has a total weight of all its jobs that is greater than that of S_A, the difference being the weight of X. An adequate assignment of weights to jobs can lead to the conclusion that the sum of weights of all jobs contained in the schedule S_A is at least the sum of the weights of all jobs in the considered optimal schedule, a contradiction.

There are infinite ways of assigning weights, and there is no unique strategy that leads to success. Usually, the weight function is monotonic with regard to job lengths, and, as X has the smallest job length, its weight can be set to 1. In the following, we denote both the job X and the time X would need to be executed on the slowest processor by X. The other weights can be assigned for intervals of job lengths, for example, a job with length within $[X,1.5X)$ could be assigned weight 1. Weights can also be assigned otherwise: for example, a job with a certain property can be chosen to be the end of a weight interval. Jobs that have a certain property can also be assigned a specific weight that corresponds to that property. Of course, while proving different theorems that lead to the proof of a worst-case approximation bound, a new weight function can be chosen for each statement to prove.

6.3 Normalizing time intervals and job execution times

In order to reason easier about time, one can divide all durations of time intervals in which jobs can be scheduled by X if scheduling on identical processors. For uniform processors, such intervals can be divided by the time it would take X to execute on the processor on which the interval occurs, in order to derive the *length* of the interval [4]. Also, all job durations can be divided by X and these normalized durations can be used in the proofs. For example, a theorem could be proved that there are no jobs that have a longer duration than 5, or it can be stated that the unused time intervals on processors before the MULTIFIT deadline B all have length less than 1, as otherwise X would have been scheduled in one of those intervals.

6.4 Task density

In the case of uniform processors, the time intervals can have unbounded lengths because the speed factors may be arbitrarily high. A way to describe the amount of jobs assigned within a schedule in such an interval is to use *task densities*, which can be defined for each task as being the ratio between its weight and its length. Also, a *task type density* can be defined as a lower bound for all possible task densities of tasks of that type. The concept of task density can be used in order to reason about time intervals that may be very long. For example, the total weight of all tasks in an interval of length t is at most t times the maximum task type density among all task types represented within that interval.

6.5 Processor with more execution time in the optimal schedule

We use the notations from the previous subsection. Since X is not contained in S_A, there must be a processor p^* in the optimal schedule that has a total execution time that is greater than that of S_A on p^*. Such a processor can be analyzed in detail and may be shown to have certain properties that, in conjunction with other methods, result in proofs of useful theorems. For example, the existence of p^* may imply a certain minimum duration of the optimal schedule, in conjunction with the observation that X could not be fitted by A on p^* without causing the maximum completion time of S_A to exceed B.

7. Conclusions and future challenges

In this work, we considered worst-case approximation for scheduling on multiple machines with availability constraints or with fixed jobs in order to minimize the maximum completion time. We surveyed results obtained in this research area and commented upon the algorithms used.

Prominent among these algorithms are LPT and MULTIFIT and their variants, whereas for multiprocessor scheduling with fixed jobs, more complicated algorithms were used to achieve best possible worst-case approximation bounds in the class of polynomial algorithms assuming that $P \neq NP$ in the general case where there can be any number of fixed jobs on each machine.

The problem of scheduling with availability constraints cannot be approximated by a polynomial-time algorithm with a constant worst-case approximation bound, even if there is at most one downtime on each machine, unless assumptions about the downtimes are made. The results we presented in this area address the problem of scheduling on identical processors with at most one downtime on each machine, with various assumptions.

Due to its different objective function, the problem of scheduling on identical parallel machines with fixed jobs allowed for the development of a polynomial-time approximation algorithm with a worst-case approximation bound of 1.5, and the development of a PTAS for scheduling on a constant number of uniform machines with fixed jobs was also possible.

The MULTIFIT and LPT variants developed for the discussed variants of these problems could be useful in practice, as their time complexity is low and thus they should be able to address very large problem instances, as they are easy to implement, and because in some cases their worst-case approximation bounds could be considered to be good enough. In the case of scheduling on uniform nonsimultaneous machines, the average performance of a MULTIFIT variant was studied, and shown to be very good, as the experiments suggest that in general, for instances that can be relevant in practice and for which exhaustive search is not an option, the algorithm returns schedules with a maximum completion time that is within 3% of that of an optimal schedule.

We also elaborated on the most encountered proof techniques in worst-case approximation bound proofs for LPT and MULTIFIT variants.

The limitations of the presented works result mainly from the difficulty of the problem of scheduling with unavailability periods when considering the subject of approximation. To assess how well the proposed heuristics for this problem perform under such conditions is difficult, as it is hard to have a good estimate of the optimal schedule unless it is computed by an exact algorithm as was done in [26]. This problem has therefore been addressed only by considering the special case where there is at most one downtime on each machine.

In the future, it may be interesting to compare the heuristics proposed for the same problems experimentally.

Another limitation of the discussed works and of many other research works on scheduling results from the fact they attempt to understand problems with one, two, or at most three aspects at one time, whereas in many practical problems such as some production planning problems, many aspects occur at once. For example, availability constraints can appear alongside a multitude of other constraints that have to be considered simultaneously. These can be precedence constraints, that certain jobs have to be assigned to certain machines, or preferences of the manufacturer that the machines should not have more than a 60% or another predefined load for example in order to leave room for unexpected events. Furthermore, orders often come online, and if an urgent order from an important client needs to be given priority, this can alter the delivery times of other orders. Also, delivery times and delays have a big relevance in practice, as not delivering on time can cause fines. Such practical problems can also have sequence-dependent setup times, the necessity for setup operators to be present to perform setups, the preference that setup times are kept low by putting jobs from the same family of types of jobs consecutively on machines whenever possible, the necessity for workers to attend to certain machines while production takes place, and worker breaks and holidays. The preexisting schedule also has to be kept unchanged for a predefined time period since materials are brought to the production place in preparation for the production process. In addition, orders may have priorities and deadlines. For such problems, given the time constraints in which the schedule needs to be generated and that there can be thousands of jobs, usually a heuristic is employed that first orders the jobs for example by using priorities assigned to them and/or their deadlines and then schedules them on the machines in that order while also obeying all constraints and attempting to fulfill all preferences. Difficulties in researching such problems include that probably for different sets of orders different scheduling strategies may be better, and that an optimal schedule may be very hard to find and thus it is hard to quantify how well a heuristic performs.

Acknowledgements

This publication was supported by the Open Access Publication Fund of Technische Universität Berlin.

Conflict of interest

The author declares no conflict of interest.

Abbreviations

MSP	multiprocessor scheduling problem
NMSP	nonsimultaneous multiprocessor scheduling problem
UNMSP	uniform nonsimultaneous multiprocessor scheduling problem
LPT	largest processing time
FFD	first fit decreasing

Author details

Liliana Grigoriu[1,2]

1 Department of Computer Science and Engineering, Faculty of Control and Computers, Politehnica University Buchares, Bucharest, Romania

2 Department of Mathematics, Technical University of Berlin, Berlin, Germany

*Address all correspondence to: liliana.grigoriu@cs.pub.ro

References

[1] Hwang H-C, Chang SY. Parallel machines scheduling with machine shutdowns. Computers and Mathematics with Applications. June 1998;**36**:21-31

[2] Hwang H-C, Lee K, Chang SY. The effect of machine availability on the worst-case performance of LPT. Discrete Applied Mathematics. April 2005;**148**:49-61

[3] Grigoriu L, Friesen DK. Scheduling on same-speed processors with at most one downtime on each machine. Discrete Optimization. November 2010;**7**:212-221

[4] Grigoriu L, Friesen DK. Scheduling on uniform processors with at most one downtime on each machine. Discrete Optimization. November 2015;**17**:14-24

[5] Scharbrodt M, Steger A, Weisser H. Approximability of scheduling with fixed jobs. Journal of Scheduling. November 1999;**2**(6):267-284

[6] Jansen K, Pradel L, Schwarz UM, Svensson O. Faster approximation algorithms for scheduling with fixed jobs. In: 17th Conference of Computing: The Australasian Theory Symposium (CATS 2011), Perth, Australia, January. 2011

[7] Graham RL. Bounds on multiprocessing timing anomalies. SIAM Journal of Applied Mathematics. March 1969;**17**:416-429

[8] Coffman EG Jr, Garey MR, Johnson DS. An application of bin-packing to multiprocessor scheduling. SIAM Journal on Computing. February 1978;**7**:1-17

[9] Hwang HC, Lim K. Exact performance of MULTIFIT for nonsimultaneous machines. Discrete Applied Mathematics. 2014;**167**:172-187

[10] Kellerer H. Algorithms for multiprocessor scheduling with machine release times. IIE Transactions. 1998;**30**:991-999

[11] Grigoriu L, Friesen DK. Approximation for scheduling on uniform nonsimultaneous parallel machines. Journal of Scheduling. December 2017;**20**:593-600

[12] Chen B. Tighter bound for multifit scheduling on uniform processors. Discrete Applied Mathematics. May 1991;**31**:227-260

[13] Lee CY. Parallel machine scheduling with nonsimultaneous machine available time. Discrete Applied Mathematics. January 1991;**30**:53-61

[14] Chang SY, Hwang HC. The worst-case analysis of the MULTIFIT algorithm for scheduling nonsimultaneous parallel machines. Discrete Applied Mathematics. June 1999;**92**:135-147

[15] Yue M. On the exact upper bound of the MULTIFIT processor scheduling algorithm. Annals of Operations Research. December 1990;**24**:233-259

[16] Friesen DK, Langston MA. Bounds for multifit scheduling on uniform processors. SIAM Journal on Computing. February 1983;**12**:60-69

[17] Burkard RE, He Y. A note on MULTIFIT scheduling for uniform machines. Computing. 1998;**61**:277-283

[18] He Y. Uniform machine scheduling with machine available *constraints*. Acta Matematicae Applicatae Sinica (English Series). 2000;**16**:122-129

[19] Grigoriu L, Friesen DK. Scheduling on uniform nonsimultaneous parallel machines. In: Fink A, Fiigenschuh A, Geiger M, editors. Operations Research

Proceedings 2016 Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), Hannover, August 30–September 2. 2016. pp. 467-473

[20] Lee CY, Lei L, Pinedo M. Current trends in deterministic scheduling. Annals of Operations Research. April 1997;**70**:1-41

[21] Sanlaville E, Schmidt G. Machine scheduling with availability constraints. Acta Informatica. September 1998;**35**:795-811

[22] Schmidt G. Scheduling with limited machine availability. European Journal of Operational Research. February 2000;**121**:1-15

[23] Lee C-Y. Machine scheduling with availability constraints. In: Leung JY-T, editor. Handbook of Scheduling: Algorithms, Models and Performance Analysis. London: Chapman & Hall/CRC; 2004. pp. 22-1-22-13

[24] Ma Y, Chu C, Zuo C. A survey of scheduling with deterministic machine availability constraints. Computers & Industrial Engineering. 2010;**58**:199-211

[25] Lee CY. Machine scheduling with an availability constraint. Journal of Global Optimization. December 1996;**9**:395-416

[26] Kaabi J, Harrath Y. Scheduling on uniform parallel machines with periodic unavailability constraints. International Journal of Production Research. 2019;**57**(1):216-227

[27] Diedrich F, Jansen K. Improved approximation algorithms for scheduling with fixed jobs. Proceedings of 20th ACM-SIAM Symposium on Discrete Algorithms (SODA). 2009:675-684

[28] Grigoriu L. Multiprocessor scheduling with availability constraints [PhD thesis]. College Station, TX, USA: Texas A&M University; 2010

[29] Grigoriu L. Scheduling on parallel machines with variable availability patterns [PhD thesis]. Bucharest, Romania: Politehnica University Bucharest; 2012