Computers & Operations Research 64 (2015) 198–209

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Improving schedule stability in single-machine rescheduling for new operation insertion

Can Akkan *

Sabancı School of Management, Sabancı University, Orhanlı, Tuzla, Istanbul 34956, Turkey

ARTICLE INFO

Available online 12 June 2015

Keywords:
Production scheduling
Single machine
Deterministic
Heuristics
Stability
Robustness

ABSTRACT

The problem studied here entails inserting a new operation into an existing predictive schedule (preschedule) on a (non-preemptive) single machine by rescheduling its operations, so that the resultant schedule is the most stable one among schedules with minimal maximum tardiness. Stability is measured by the sum of absolute deviations of post-rescheduling start times from the pre-rescheduling start times. In addition to several simple heuristics, this study investigates a hybrid branch-and-bound/local-search algorithm. A large set of instances that include cases with inserted idle times allows for tests of the performance of the heuristics for preschedules with varying degrees of robustness. The results show that algorithms can be developed that significantly improve the stability of schedules with no degradation in T_{max}. In addition, new insights emerge into the robustness characteristics of a preschedule. Specifically, the number of gaps in the schedule, equal distribution of total slack among these gaps, and the slack introduced beyond the amount enforced by release times all have effects on schedule robustness and stability.

1. Introduction

This research addresses a scheduling problem in which a new operation is inserted into an existing predictive schedule (*preschedule*) for a single machine. We assume this is a finite-capacity schedule that has been prepared for planning, capacity reservation, and due time commitment purposes. Therefore, maintaining its stability, while making room for a new operation, is an important concern. To address this concern, the preschedule contains inserted idle times, or "gaps", between scheduled operations.

All operations have release time r_j, due time d_j and processing time p_j. The start times of operations in the preschedule are denoted by s_j^o for $j=1,\ldots,(n-1)$, and the start times after rescheduling are denoted by S_j for all j. Rescheduling is not allowed to violate the release time constraint of an operation; violations of release times are likely to be costly or even infeasible, because they depend on the arrival of parts from suppliers or other machines. We assume that the machine is operational continuously, all processes are deterministic, and operations cannot be preempted.

We define the problem as a hierarchical or lexicographical optimization problem (e.g., see [1]). The primary objective is the minimization of maximum tardiness (T_{max}) for all operations, including the new one. The secondary objective is maximizing schedule stability. Schedule stability is measured by $\overline{D}=\sum_{j=1}^{(n-1)}\left|s_j^o-S_j\right|/(n-1)$, and we attempt to find the schedule with minimum $\overline{D}$ among all schedules that minimize $T_{\max}$.

This problem is not only interesting theoretically but also relevant practically. According to recent survey research presented in De Snoo et al. [2] focused on manufacturing firms, the three most frequently cited performance criteria for a production schedule (and associated information produced by that process) are the (i) fulfillment of constraints made to external parties, (ii) fulfillment of resource utilization constraints, and (iii) schedule robustness and information completeness, in that order. Yet in the process of developing manufacturing schedules, the two most important criteria are communication quality and the flexibility of schedule adaptation. The scheduling problem defined here addresses most of these criteria: rescheduling supports the need for flexibility; the objective of accepting a new order while keeping the maximum tardiness and the deviation from a preschedule small not only supports the fulfillment of promises (constraints) made to suppliers and the rest of the supply chain but also helps increase utilization, by accepting new orders that do not at first fit into the existing schedule. As De Snoo et al. [2]

* Tel.: +90 216 4839685; fax: +90 216 4839699.
E-mail address: canakkan@sabanciuniv.edu

http://dx.doi.org/10.1016/j.cor.2015.05.015
0305-0548/

acknowledge “schedulers indicate that responsiveness in [requests for rush orders] is highly important”.

Recently, rescheduling and the more general topic of scheduling under uncertainty has attracted the interest of many researchers, as reviewed by Mehta and Uzsoy [3], Davenport and Beck [4], Vieira et al. [5], Herroelen and Leus [6] (who include project scheduling as well), Aytug et al. [7], Ouelhadj and Petrovic [8] and Sabuncuoglu and Goren [9]. These reviews highlight two main research themes: developing robust schedules that are less sensitive to disruptions and rescheduling operations by reacting to disruptions. Robustness is measured with respect to either the performance criterion of schedules (e.g., makespan, tardiness) or with respect to some characteristic(s) of the schedule itself (e.g., sequence of operations, start times of operations, assignments of operations to machines). The latter is often referred to as schedule stability, which is important because a schedule often forms the basis for planning for other activities in the production system (e.g., material delivery, tool and resource allocation), and even small changes in the schedule could ripple through the production system, creating “nervousness” [7]. We refer to the first type of robustness as simply *robustness* and the second as *stability*.

Several authors study the use of slack to improve the robustness of schedules. Despite a general consensus (e.g., [10–13]) about the need to introduce slack into a schedule to improve robustness, more work is necessary to develop methods to determine the amount of slack and its distribution within the schedule. This need is highlighted by the computational experiments conducted in this research as well.

The most commonly used measure of schedule stability, at least within the context of single machine scheduling, is the sum of absolute differences between preschedule start times and post-disruption (or post-rescheduling) start times, $\overline{D}$. To the best of our knowledge, the first article to use this measure was Wu et al. [14], followed by O’Donovan et al. [15], Mehta and Uzsoy [11], and Hall and Potts [16], among others. Another set of stability measures quantify the extent of change in the sequence. For example, Wu et al. [14] use start times for this purpose, whereas Watatani and Fujii [17] and Hall and Potts [16] use distance metrics between two sequences. In other contexts, such as parallel machines or batch processes, other measures of schedule stability might be justified.

The approach of maintaining a predictive schedule and reacting to disruptions (e.g., new job arrival, machine failure) is known as predictive–reactive scheduling (e.g., [18,19]). According to Aytug et al. [7], the purposes of a predictive schedule (preschedule) include the following: serving as a capacity check; providing visibility for the rest of the organization, external suppliers, and customers; evaluating the performance of the shop-floor personnel; and avoiding any further problems by serving as a feed-forward control tool.

An important distinction among predictive–reactive scheduling problems, which relates closely to the type of disruption, is whether the rescheduling is done on the schedule that currently is being executed on the shop floor or one that exists for planning purposes some time into the future (e.g., next day, week, or even month). This distinction has implications for the required running time of the rescheduling algorithms and the degrees of freedom in the schedule change. Plan stability (especially near-term) is as important as the stability of the execution in the shop floor due to the supply chain's interdependencies and constraints.

In summary, the rescheduling problem addressed in this paper follows a predictive–reactive scheduling strategy, with event-driven rescheduling in the planning (rather than execution) phase of the schedule, due to the arrival of a new job that serves as the triggering event (disruption). In practice, this problem occurs in finite-capacity scheduling used to provide capable-to-promise (CTP) functionality (e.g., [20]).

In terms of the main issue being addressed and the way we model stability, the previous work most related to our efforts is Wu et al. [14]. However, they use makespan as the main efficiency criterion, and the disruption they model is machine failure. Hall and Potts [16] and Hall et al. [21] are also closely related to the problem we define here, though in terms of the way the issue is modeled, there are significant differences. Hall and Potts [16] develop two classes of models: (i) the cost of disruption is included as a constraint in the model, and (ii) a total cost function is taken as the objective function, which includes the cost function (e.g., the maximum lateness, or the sum of the completion times) and the disruption cost. The disruption cost is modeled in two different ways, namely, as the sum of absolute deviations in the start times (as done here) or the number of position changes in the sequence. For several single-machine problem classes (none with the release times), they provide either an efficient solution algorithm or an intractability result. Hall et al. [21] model the rescheduling of a single machine due to a set of new orders as a problem of minimizing the maximum lateness of all jobs, subject to a limit on the maximum time change of the prescheduled jobs (jobs are all available at time zero; preschedule is not necessarily optimal and may include idle time).

With this study we make two main contributions. First, we provide evidence to support previous conclusions about stability in scheduling, in that we show that algorithms can be developed that significantly improve the stability of schedules with no degradation in an efficiency-based schedule objective. Second, we develop further insights into the robustness characteristics of a preschedule, beyond a general conclusion in prior literature that adding a slack to a schedule improves robustness. Specifically, the number of gaps in the schedule, the distribution of total slack among these gaps (measured by the Gini coefficient), and the slack introduced beyond the requirement enforced by the release times have been analyzed as factors with potential effects on schedule robustness and stability.

Section 2 offers a more detailed, formal definition of the problem. After discussing the solution approaches in Section 3, we provide a detailed description of how we generated instances in Section 4, followed by discussion of the computational results in Section 5 and concluding remarks in Section 6.

2. Problem description

As we discussed in Section 1, we assume that there is a schedule created for planning purposes on a single machine, and there are currently $n-1$ operations in this schedule. This predictive schedule is referred to as the *preschedule*, and the operations are indexed $j=1\ldots(n-1)$. Each operation j in the preschedule has a release time r_j, a start time s_j^o, a processing time p_j, and a due time d_j. The preschedule is feasible and without any tardiness ($r_j \le s_j^o$ and $s_j^o + p_j \le d_j$ for $j=1,\ldots,n-1$).

Given this preschedule, a request for scheduling a new operation arrives. This new operation is indexed by n, and its release time, due time, and processing time are denoted by r_n, d_n, and p_n, respectively. Operation n cannot be inserted into the preschedule between r_n and d_n without rescheduling at least one of the operations in the preschedule. S_j denotes the start time of operation j after the insertion of the new operation, so that $r_j \le S_j$ for $j=1,\ldots,n$. In creating this new schedule, there are two objectives: the primary objective is minimizing the maximum tardiness, and the secondary objective is maximizing the schedule stability, as defined below. The tardiness of operation j is denoted by $T_j = \max(0, S_j + p_j - d_j)$ and $T_{max} = \max_{j=1\ldots n} T_j$. Schedule stability is measured by $\overline{D} = \sum_{j=1}^{n-1} \left| s_j^0 - S_j \right| / (n-1)$. Letting T_{max}^* denote the minimum T_{max}, the overall objective is to find the schedule that