Applied Soft Computing 1 (2001) 35–52

Applied Soft Computing

www.elsevier.com/locate/asoc

Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures

Mikkel T. Jensen*

Department of Computer Science, University of Aarhus, Aarhus, Denmark

Accepted 23 March 2001

Abstract

The traditional focus of scheduling research is on finding schedules with a low implementation cost. However, in many real world scheduling applications finding a *robust* or *flexible* schedule is just as important. A robust schedule is a quality schedule expected to still be acceptable if something unforeseen happens, while a flexible schedule is a quality schedule expected to be easy to change. In this paper, the robustness and flexibility of schedules produced by minimising different robustness measures are investigated. One kind of robustness measure is the *neighbourhood-based robustness measure*, in which the basic idea is to minimise the implementation costs of a set of schedules located around a centre schedule. For tardiness problems another way of improving robustness is to increase the slack of the schedule by minimising lateness instead of tardiness. The problems used in the experiments are maximum tardiness, summed tardiness and total flow-time job shop problems.

The experiments showed that the neighbourhood-based robustness measures improves robustness for all the problem types. Flexibility is improved for maximum tardiness and loose summed tardiness problems, while it is not improved for tight summed tardiness problems and total flow-time problems. The lateness-based robustness measures are found to also improve robustness and in some cases flexibility for the same problems, but the improvement is not as substantial as with the neighbourhood-based measures.

Based on these observations, it is conjectured that neighbourhood-based robustness can be expected to improve flexibility on problems with few *critical points*. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Stochastic scheduling; Robustness; Flexibility; Job shop scheduling

1. Introduction

When solving a scheduling problem the focus traditionally is on minimising a measure of the cost of implementing the schedule. However, most real world scheduling systems operate in dynamic environments, in which unforeseen and unplanned events can happen at short notice. Such events include the breakdown of machines, employees getting sick, new jobs appearing, etc. The problem encountered when an unforeseen event and a schedule has to be changed is usually called a *rescheduling problem*. When a rescheduling problem is solved a new schedule incorporating the changes in the environment and the part of the *preschedule* (the schedule followed prior to the breakdown) already implemented is sought. This schedule should ideally have as low an implementation cost as possible. When the unforeseen event is a breakdown (the temporary unavailability of a resource), the simplest way to solve a rescheduling problem is often to keep the processing order of the preschedule, but delay processing when necessary. In the following this kind of rescheduling is called *simple rescheduling* or

* URL: www.daimi.au.dk/~mjensen/.
E-mail address: mjensen@daimi.au.dk (M.T. Jensen).

right-shifting. Right-shifting is the simplest and fastest kind of rescheduling, but in order to improve performance more complex methods searching some set of schedules can be used. In the following, this is called *rescheduling using search*.

The difficulty of the rescheduling problem depends on the nature of the breakdown as well as the preschedule. Some preschedules will generally lead to rescheduling problems with lower implementation costs than others. A preschedule which tends to perform better than ordinary schedules after a breakdown and right-shifting is termed *robust*, while a schedule which tends to perform well after a breakdown and rescheduling using search is termed *flexible*.

It is difficult to relate the terms flexibility and robustness to each other. Often a schedule which is robust can also turn out to be flexible to some degree, since robustness means that the schedule is still acceptable if small delays happen during schedule execution. The acceptability of small delays is an advantage if small changes are made to the schedule. On the other hand, the acceptability of small delays does not necessarily say anything about the possibility of making profound changes in the schedule.

The objective of this paper is to investigate two ways of achieving schedule robustness and flexibility for job shop problems. The first way is the neighbourhood-based robustness measure technique used in [10] on makespan problems, which is reformulated for maximum and summed tardiness, and total flowtime problems. The second way is a simpler idea applicable to tardiness problems; by minimising a measure of lateness instead of tardiness, the slack in the schedules can be increased, which may improve the rescheduling performance of the schedules. The slack of an operation in a schedule is the "buffer time" by which the operation can be delayed without worsening the performance of the schedule.

The work presented in this paper is an extension of the work presented in [11], in which the neighbourhood-based robustness idea was compared to ordinary scheduling for the performance measures maximum tardiness, summed tardiness and total flow time on a smaller range of problems.

The outline of the paper is as follows. Section 2 defines the job shop scheduling problem and notation. In Section 3 previous work on robust scheduling is briefly covered. Section 4 introduces the robustness measures for the maximum, summed tardiness and total flow-time job shop problems. In Section 5 the genetic algorithm used to perform the scheduling is described, while Section 6 describes how breakdowns are simulated and how rescheduling is performed in the experiments. Section 7 describes the experiments and reports the results. Section 8 contains the conclusions.

2. Notation

An $N \times M$ job shop scheduling problem consists of N jobs and M machines. A job J_j consists of a sequence of operations $\bar{O}_j = (o_{j1}, o_{j2}, \ldots, o_{jk_j})$. Each operation o_{jl} is to be processed on a specific machine and has a specific processing time τ_{jl}. Each job has at most one operation on each machine. The processing order of the operations in job J_j must be the order specified in the sequence $\bar{O}_j$. These sequences are often called *the technological constraints*. During processing each machine can process at most one operation at a time, and no preemption can take place; once processing of an operation has been started it must run until it has completed. In the following C_j will denote the end of processing time of the last operation of job J_j in a given schedule.

Some problems include a *due date* d_j for each job, a time by which the processing of the job is supposed to be finished, a *release time* r_j for each job, prior to which no processing of the job can be done, or a *initial set-up time* s_{m} for each machine, prior to which on processing can be done on the machine.

A number of different objective functions exist for job shop problems. The most extensively researched is the *makespan* $C_{\max} = \max_{j\in\{1,\ldots,N\}}(C_j)$, the time elapsed from the beginning of processing until the last operation has completed. The makespan objective is not realistic, since it is not well-suited for scheduling on a rolling time horizon-basis (jobs arriving continuously over time), and since it does not include due dates. More realistic objectives include *total flowtime* $F = \sum_{j=1}^{N} C_j - r_j$, *summed lateness* $L_{\sum} = \sum_{j=1}^{N} C_j - d_j$, *summed tardiness* $T_{\sum} = \sum_{j=1}^{N} \max(C_j - d_j, 0)$, *maximum lateness* $L_{\max} = \max_{j\in\{1,\ldots,N\}}(C_j - d_j)$ and *maximum tardiness* $T_{\max} = \max(L_{\max}, 0)$. All of these performance measures reflect schedule implementation cost and