Solving Stochastic Parallel Machine Scheduling using a Metaheuristic Approach with Efficient Robustness Estimation.

Utrecht University

M. S. Hessey
ICA-3496724

Supervisor: J. M. Van Den Akker
J. A. Hoogeveen

Department of Computing Science
Utrecht University

MSc Thesis

March 2019

Abstract

Although robustness is often discussed when solving stochastic problems, definitions in the literature vary. We present five quantitative definitions of robustness that are close to the intuitive qualitative meaning of robustness as used by other authors. Since many scheduling problems are NP-hard, they are sometimes solved using Metaheuristic approaches. In such a case, we need a way of estimating robustness efficiently. Using Stochastic Parallel Machine Scheduling as a test problem, we analyze efficient measures for estimating these definitions.

Our results show three things. First, there are some limitations to 'slack-based' measured proposed by other authors. Second, optimizing the deterministic variant of the problem shows decent results on some problem instances, but can be a poor predictor of the expected makespan in schedules with lots of inter machine precedence arcs and no slack. In these cases, a statistical approximation approach showed good results. Third, in less extreme instances with less precedence arcs and whose solutions have more slack, the use of an efficient robustness estimator does not improve the robustness of the solutions produced by our metaheuristic approach.

Taken together, our results show that in some cases deterministic makespan minimization performed as well as any other measure, but that situations exist in which it is outperformed by a statistical approximation approach. We did not find any situations in which the statistical approximation was outperformed by deterministic makespan pr any other robustness estimation measure. It remains an open question which aspect of the schedule or problem instance determines if uncertainty must be considered. Our hypothesis is that as schedules approach the global optimum (for any robustness definition presented herein), the quality of the robustness estimation method used in the metaheuristic approach becomes more important.

Contents

Chapter 1

Project Scope

1.1 Introduction

Scheduling[1] is the process of optimally assigning tasks, often known as *jobs*, to scarce resources (e.g. machines or personnel) over time. Recently, attention in scheduling theory has been given to scheduling where the processing time of a job is stochastic. A common objective is the minimization of the expected time at which the last job completes, which is known as the *expected makespan*. However, in an application, one may prefer a solution with a slightly higher expected makespan if certain guarantees about performance under disturbance can be made. Such a solution is called *robust*. To the best of our knowledge, no universal formal definition of robustness exists. One intuitive definition is given by [GS08]: a schedule which does not significantly degrade in the face of disruption is called *robust*.

Two things may be important in a practical setting. First, that the objective function does not significantly degrade (e.g. a due date for a project is highly likely to be met, even under disruptions). Following [HL04b] and [Von$^+$05] we shall refer to such a solution as *quality robust* (other names exist, such as *objective protecting schedules*). Second, one may want to avoid reassigning personnel or machines during project execution. Again following [HL04b] and [Von$^+$05] we call a schedule that is likely to remain feasible under disruptions *solution robust* (such solutions are also referred to as *stable*).

[1] An overview of scheduling terminology and notation can be found in the appendix under Notation. A detailed introduction to scheduling theory can be found in [Pin12].

A lack of a universal definition is not due to a lack of interest in robustness. In fact, many publications discuss ways for creating a schedule with worst case guarantees (see Chapter 2 for an overview). Methods to create schedules with worst case guarantees broadly fall into one of two categories [HL04b][HL05][BKF12]: *Proactive* methods attempt to create a robust schedule that is feasible even under fluctuations due to uncertainty. *Reactive* methods attempt to update (fix) the schedule while it is being executed, known as fixing the schedule online. In this work we will focus on proactive methods.

Many deterministic scheduling problems are NP-hard in the strong sense. Even when good solutions exist for deterministic versions of a scheduling problem, the (realistic) problem where the processing times are stochastic may be difficult to solve. Due to the complexity of stochastic variants of NP-hard problems, these problems are often solved using *local search* approaches. Local search approaches rely on exploring many potential solutions, known as *states*. Consequently in order to consider robustness during the local search, the need arises to estimate the robustness of a schedule efficiently. We will compare the performance of several robustness estimation measures (RMs) and illustrate their use on a scheduling problem known as *stochastic parallel machine scheduling with precedence relations*. In Section 4.2.1) we argue that stochastic parallel machine scheduling with precedence relations is, in a sense, one of the simplest examples of commonly studied hard problems and is therefore an excellent problem to work on.

1.1.1 Stochastic Parallel Machine Scheduling with precedence relations

In the Stochastic Parallel Machine Scheduling problem (SPMS) we are given n *jobs*[2] $(J_1, \cdots, J_n)$ to be processed on m identical machines. A job may require other jobs to have been completed before it can be performed. For example, in a car manufacturing process, we cannot attach the wheels to the chassis without first having assembled the chassis. We call such a requirement a *precedence relation*. Processing job J_j requires $\mathbf{p}_j$ uninterrupted processing time on a machine, where $\mathbf{p}_j$ is a random variable drawn from some known distribution $\mathcal{D}_j$. Processing of this job may start immediately at any time after its release date r_j, as long as all its predecessors are completed (known as *0-lag finish-start* precedence constraints). Using the 3-field notation[3] due to Graham it is the $P_m|\mathbf{p}_j, \text{prec}, r_j|C_{\max}$ problem.

[2] When precedence relations exist, some authors refer to jobs as *operations* and call a set of operations connected by precedence relations a job. We do not take that approach.

[3] See Section 1.1.3 for a brief introduction and appendix A for a complete overview of this notation

Solutions are given in the form of a *schedule*: an assignment of a starting time s_j and a machine m_j to each job. We restrict ourselves to offline scheduling: schedules may not be updated during execution.

We refer to 4.2.1 for a detailed description of why SPMS is a relevant problem.

1.1.2 Representation

Machine Scheduling problems with precedence constraints can be represented as a graph, consisting of nodes representing jobs and two types of arcs. Firstly, conjunctive directed arcs between jobs define precedence relations. These are known as *precedence arcs* (PA). Secondly, for a given assignment, disjunctive, undirected edges link jobs that are assigned to the same machine. These are known as *machine arcs* (MA). The union of these arcs we denote simply by A. Making the machine arcs directed is the same as determining an order in which jobs will be performed on that machine. An order feasible assignment is one in which the resulting graph is acyclic. The earliest reference we could find for this representation is [RS64]. It has since been widely used, including among others [HL04a; Von+05; BL09; MG10]. The *earliest feasible start time* ($s(ESS)$) is determined by a parallel schedule generation (see [HK00]), specifically as described in [Pin12], in algorithm 5.1.3, page 115: whenever a job finishes, we schedule all available jobs. The *latest feasible start time* ($s(LSS)$) is similarly determined, working backwards from the makespan. More detail and an example is given in Section 2.1.2. The same technique is used to determine the *critical jobs*: The set of jobs in the graph which cannot be started at a later or earlier time without moving another job. Similarly, we define a *critical path*: a path in the graph consisting of critical jobs.

1.1.3 Deterministic scheduling problems related to parallel machine scheduling

Below we present a list of well known results in scheduling, building up towards the problem we are considering. We will use the common three field notation by Graham. The first field denotes what type of machines are available: 1 means a single machine, P_m denotes m identical, parallel machines. The second field denotes which assumptions are made on the jobs (r_j means jobs have a release date, prec means precedence relations exist). Finally the third field denotes the objective: $C_{\max}$ represents makespan minimization. See Appendix A for a complete overview of notation.

We begin by noting that $1||C_{\max}$ is trivial: any order of jobs will have the same result. On two machines, the problem is no longer trivial: $P_2||C_{\max}$ is known to be NP-hard in the weak sense (it is the PARTITION problem) [Pin12]. Next we introduce release dates (denoted r_j). Clearly $P_m|r_j|C_{\max}$ is equivalent to $P_m|r_j, d_j = 0|L_{max}$, where L_{max} is minimizing the maximum lateness of a job. $P_m|r_j|L_{max}$ has a polynomial time approximation scheme [Mas03] and thus so has $P_m|r_j|C_{\max}$. $P_m|\text{prec}|C_{\max}$ is NP-hard in the strong sense [Pin12]. However, some special cases are easily solvable: for $1|\text{prec}|C_{\max}$ it is optimal to start any available job whenever the machine becomes available. In the special case where all jobs have a processing time of 1 and $P_m|\text{prec}, p_j = 1, \text{tree}|C_{\max}$, optimal solutions are found using the Critical Path or Largest Number of Successor rules. Because $P_m|\text{prec}|C_{\max}$ is NP-hard in the strong sense, so is $P_m|\text{prec}, r_j|C_{\max}$. This problem can be solved by using column generation [AHK12].

1.1.4 Introduction to local search approaches

Local search approaches are often used when the solution space is too large to explore fully, as is the case for NP-hard problems. The aim is to minimize or maximize an objective function in the search space. This is done by taking steps through the search space, moving from solution to solution. Solutions that are reachable from one-another within a single step are called Neighbors. The *neighborhood operator* defines for any solution a set of solutions that are reachable within a single step. The search space is also commonly referred to as the *landscape* and depending on the objective function the local searcher is called a *Hill-Climbing* algorithm (maximizing an objective function) or a *Vertex Decent* (minimizing an objective function). Local search algorithms can be either *first-improvement* or *best-improvement*. First-improvement algorithms will explore neighboring solutions until any improvement is found. Best improvement will exhaustively explore all neighbors, then move to the best neighbor.

A problem for local search algorithms is that they may get stuck in local optima, or on plateaus - local optima in the weak sense. Various techniques exist to deal with this issue, such as *tabu-search*, *simulated annealing*, *Multi-Start Local Search*, *Iterated Local Search* and *Genetic Local Search*. A description of the last three is given by[Thi], summarized below:

Multi-Start Local Search (MLS)

A Multi-Start Local Search, abbreviated to MLS, repeatedly creates a random initial solution and performs Hill Climbing (vertex decent) until a local optimum is reached. It then outputs the best solution found. Although this technique is very basic, if the Hill Climbing steps are fast enough, it can explore a great many solutions quickly, increasing the chance that a good solution is found.

Iterated Local Search (ILS)

Iterated Local Search, abbreviated to ILS, start with a random solution and uses a local searcher to find a local optimum. It then performs a random mutation and uses the local searcher until it finds a (hopefully different) local optimum. If the new solution is the best found so far, it is saved. If a worse local optimum is found, the new solution is discarded. This way the ILS repeatedly uses the currently best encountered solution to find a better solution. The size of the mutation is important: If it is too large, ILS reverts to MLS. If it is too small, ILS may fail to escape a local optimum. ILS approaches can work well when the search landscape is structured: i.e. if the value of local optima approach that of the global optimum, as the solutions approach the optimal solution. If this is not the case, they offer little benefit over MLS.

Genetic Local Search (GLS)

Genetic Local Searchers mimic the concept of ’survival of the fittest’ (and are therefore sometimes known as Evolutionary Algorithms). The searcher keeps track a a list of solutions, known as the *population*. In each ’generation’, the fittest solutions of the population are combined to create new solutions (offspring). There is then some form of selection (known as competition) to reduce the total population back to the original number.

Genetic Local Searchers consist of 5 key components

- Information Structures: a way of representing a solution (often called an individual).
- A selection algorithm to determine which solutions are copied to population pool of the next generation.

- A way of creating new solutions based on two solutions, called 'parents'. This is known as recombination.
- A way of ensuring competition in the population pool.
- Some definition of what it means to be fit (a fitness function)

Recombination of two fit individuals (solutions with a high objective function value) hopefully leads to an even fitter solution. The second, fourth and fifth of the above components create selection pressure. A Genetic Algorithm may perform poorly if selection pressure is too high (eliminating genetic diversity too quickly), or too low (average fitness increase per generation is small). A Genetic Algorithm may also perform poorly if there is no variation (in which case the best possible solution is simply the best initial solution) or very low variation. Genetic algorithm approaches work well when there is a natural way of representing a solution. It is important that the recombination operator used fits with the problem structure (also noted in [AL03],p.15). Otherwise, recombination may be too disruptive.

1.2 Thesis structure

Chapter 2 lists works that are pertinent to our research questions. In Chapter 3 we discuss the experimental setup to answer our research questions. We quantify the research objectives, introducing quantitative definitions of robustness and specific robustness measures. Furthermore, we discuss practical aspects such as problem instances and schedule generation. Chapter 4 contains the results on the problem instances used in [PAH]. In Chapter 5 we expand on these results by considering the effect of inter machine dependencies. Finally in Chapter 6 we compare the results of steering the local search with deterministic makespan and with the best robustness measure identified in chapters 4 and 5.

1.3 Research objectives

The primary aim of this project is to find an efficient way of estimating the robustness of a schedule, which is suitable for use in metaheuristic approaches to finding schedules. Thereto we answer the following questions:

- What is a good quantitative definition of robustness? How does this depend on the characteristics of the stochastic problem?
- What are practicable robustness estimation measures (RMs) for local search procedures? Two questions here are key: How long does it take to calculate the RMs? How well do the RMs discern between schedules?

Given a set of practicable RMs, these questions arise:

- What is the effect of using these RMs during the local search on solution quality? We characterize this as the effect on three objectives:
  - What is the effect on expected makespan?
  - What is the effect on solution robustness?
  - What is the effect on quality robustness?

Chapter 2

Literature

In this chapter we discuss relevant earlier work. We begin by discussing robustness (Section 2.1): listing definitions of robustness (Sections 2.1.1 and 2.1.3) and works that use heuristic approaches to solve robust problems (Section 2.1.4). In order to discuss the definitions of robustness in Section 2.1.3, we first describe slack definitions in Section 2.1.2. We then discuss works that pertain to the choice of neighborhood operator and other technical details of metaheuristic search approaches (Section 2.2). Finally we build upon the presented works to argue that this thesis represents research that is interesting but not yet performed (Section 2.3).

2.1 Robustness

In this section, we summarize works discussing robust scheduling. Although in the introduction we have provided qualitative definitions of robustness, we have not yet given a quantitative definition. This is because, as far as we are aware, no universal quantitative definition of robustness exists. In the first two subsections we list (sometimes implicit) quantitative definitions of robustness. We distinguish between definitions that rely on the realization of a schedule[1] (Section 2.1.1) and those that do not (Section 2.1.3). Definitions that rely on the realization of a schedule tend to be closer to an intuitive, qualitative definition of robustness (such as those by [HL04a] and [Von^{+}05]), but are impractical for use during a local search. Conversely, definitions that do not depend on

[1]By realization of a schedule, we mean that the schedule has been executed, so that things like difference between predicted makespan and makespan of the executed schedule can be measured.

realization do not match an intuitive definition as well, but may be more useful during a local search. We will label these definitions as Estimation Measures.

2.1.1 Probability distribution dependent Definitions of Robustness

This sections lists quantitative definitions of robustness that are similar to the qualitative definitions given in the introduction. Although these measures are close to the intuitive definition of robustness, they rely on information about the realized performance of a schedule.

[Von+05] Provides an explicit, quantitative definition of quality robustness: "Quality robustness (makespan performance) is measured by the probability that a project ends within the projected deadline." In fact, they report the increase in project deadline required to ensure that a project meets a deadline with a certain probability as a means to compare algorithms. They also provide an implicit quantitative definition of solution robustness: "Stability cost [solution robustness]: weighted sum of absolute deviations between the actually realized activity starting times and the starting times indicated in the initial projected schedule as anticipated before project execution." They report this as a means to compare algorithms.

Quantitative definitions of robustness are not always provided explicitly. Often they are given implicitly as a performance measure. The following are measures used by authors and as such form implicit definitions.

[Deb+07] Focus on the resource constraint side of SRCPSP. Given baseline schedule, they find resource allocation through flow techniques. Their objective function is to minimize sum of weighted expected deviances in start time of a job in the actual schedule compared to the baseline schedule.

Earlier, [HL04b] also use minimizing the sum of weighted expected deviances in start time of a job.

[PAH] aim to minimize the expected makespan. They argue that this forms a good basis for quality robustness.

[BL09] develop an online list scheduling policy for the RCPSP with 0 lag start-finish relations using a Greedy Randomized Adaptive Search Procedure (GRASP). The technique is further discussed in Section 2.1.4. Here it is relevant to note their performance measure is the percentage distance between the expected $C_{\max}$ and the critical path length of a project with deterministic mean durations and that they also investigate how expected makespan and the probability of meeting a due date are related. For this they use results from the GA by [Bal07]. They argue that one should not investigate the correlation between the expected makespan and the probability that a project is on time (which the authors refer to as service level - a possible quality robustness measure) because many problem instances may have 0 or 1 probability of being on time given a policy. Rather, they argue that one should find for each problem instance the due date δ required for the project to have a certain (fixed) probability of being on time. Moreover, they explore the correlation between δ and the expected makespan, i.e. if we require a certain quality robustness, what is the relation between project due date and expected makespan? Their results show high correlation between expected makespan and due date required to ensure a certain robustness. They conclude that solutions that perform well for minimizing the expected makespan also perform well for minimizing the due date required to ensure a certain probability of finishing on time. An exception holds for the case where one required 99% robustness. The authors say this may be due to the difficulty in estimating the required due date (because 99% requires many rare events). They conclude: *"While our observations may not amount to irrefutable proof, they nevertheless provide considerable evidence ... that by searching for a scheduling policy with lowest expected makespan, one usually simultaneously minimizes the expected tardiness and maximizes the service level."* They leave space for improvement however: "Obviously, dedicated algorithms could perhaps obtain the same results in less time or achieve higher-quality outcomes with the same computational effort".

2.1.2 The definition of slack

To quantify robustness, slack is often used. Thus we must discuss definitions of slack used in various sources before turning to robustness measures that involve slack. Slack has various similar definitions by different authors. Before we go into these different definitions, we first repeat the definition the earliest and latest feasible starting schedules from Section 1.1.2 and give an example.

Earliest and Latest start schedules

Given an assignment of jobs to machines and an order on each machine in which they must be processed, the *earliest start schedule* (ESS) is the schedule in which each job is

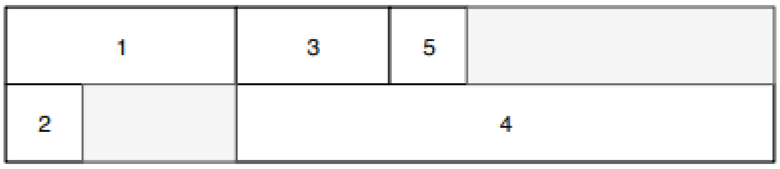


Figure 2.1: A left active schedule for the example problem. There are five jobs with processing times: $3, 1, 2, 7, 1$ respectively. The precedence relations are $(J_1, J_3), (J_1, J_4), (J_2, J_4), (J_3, J_5)$. Note that J_3 cannot be delayed without delaying J_5, but that both J_3 and J_5 can be delayed together without increasing the makespan.

started as early as possible, without changing the order in which the jobs are processed. This can be calculated using the total order on the precedence graph. That is, let the set of predecessors of job j be denoted by π_j and the machine predecessor be denoted by π_j^M. Then the earliest start time of job j is given by:

$$s_j(ESS) = \max\{r_j, \max_{i \in (\pi_j \cup \pi_j^M)} \{s_i(ESS) + p_i\}\}$$

Similarly, the *latest start schedule* (LSS) is the schedule in which each job is started as late as possible, without changing the order in which the jobs are processed or delaying the makespan. This can be calculated by first setting all latest start times to the makespan. Then going in reverse order through the precedence graph, we can calculate the latest starting time for each job. Let the set of successors of job j be denote σ_j and the machine successor be denoted σ_j^M. Then the latest start time of job j is given by:

$$s_j(LSS) = \min\{\min_{i \in (\sigma_j \cup \sigma_j^M)} \{s_i(LSS) - p_j\}, C_{\max} - p_j\}$$

This follows the recursive method described in [Pin12].

As an example, consider the problem with two machines and five jobs, with processing times: $3, 1, 2, 7, 1$ respectively. The precedence relations are $(J_1, J_3), (J_1, J_4), (J_2, J_4), (J_3, J_5)$. Assume jobs J_1, J_3 and J_5 are assigned to machine M_1 and jobs J_2 and J_4 are assigned to machine M_2. The earliest starting date for jobs J_1 and J_2 is 0, because they have no predecessors. The earliest start date for J_3 is 3, because of the (J_1, J_3) precedence relation. Similarly, the earliest start for J_4 is 3. Finally the earliest start for J_5 is $s_{J_3}(ESS) + p_{J_3} = 5$. The earliest start schedule is depicted in figure 2.1.

Job id	p	$s(ESS)$	$s(LSS)$	$s(LSS) - s(ESS)$
1	3	0	0	0
2	1	0	0	2
3	2	3	7	4
4	7	3	3	0
5	1	5	9	4

Table 2.1: The earliest and latest starting times calculated for the example schedule depicted in figure 2.1

The makespan is determined by job J_4 and is 10. Working back from the makespan, we can determine the latest feasible starting times. For jobs J_4 and J_5, that have no successors, $s_{J_4}(LSS) = C_{\max} - p_{J_4} = 3$ and $s_{J_5}(LSS) = C_{\max} - p_{J_5} = 9$. Now $s_{J_3}(LSS) = s_{J_5}(LSS) - p_{J_3} = 7$. Similarly we find latest starts for job 1 and job 2 to be 0 and 2 respectively.

The above is summarized in Table 2.1.

Slack Definitions

Slack often refers to how much the start time of a job can be delayed without influencing some aspect of the schedule, such as the makespan. Examples of this definition include:

"The start of the processing of some jobs usually can be postponed without increasing the makespan. These jobs are referred to as the slack jobs"[Pin12].

[HHE10] introduce a distinction between free slack and total slack. We will discuss the definition of total slack after the definitions of free slack: "*[Free slack is] the amount of time an activity can slip without delaying the start of any of its immediate successors.*"[HHE10; CH08] A similar formulation is: "*Define the free slack as the amount of time that an activity can slip without delaying the start of the very next activity*"[AH05].

Some of these same authors also provide a quantitative definition: "*The free slack is $LS_i - ES_i$, where $ES_i(LS_i)$ is the standard forward (backward) recursion procedure*[2] *(Hartmann and Kolisch, 2000)*"[CH08]. However, the procedure they refer to does not match their qualitative description. Indeed, the procedure in [HK00] consists of assigning each job at the "earliest possible precedence and resource feasible time". Doing the same in a backward recursion fashion is the algorithm described in [Pin12]. In this method,

[2]This is the procedure described in the previous subsection.

the slack of a job is not how much it can slip without delaying the start of any of its successors, but how much it can slip without delaying the makespan.

To make the distinction clear with a practical example, consider Job J_3 in the example from the previous subsection (figure 2.1). Note that J_3 cannot slip at all without delaying job J_5. Thus according to the qualitative definitions in [AH05; CH08; HHE10] the free slack is 0. However, we have that $s_{J_3}(LSS) - s_{J_3}(ESS) = 4$, thus by the quantitative definition, the free slack is 4.

Turning now to total slack, the definition provided is: [Total slack] is the amount of time by which the completion time of an activity can exceed its earliest completion time without delaying the project completion time.[HHE10]. They do not provide a quantitative definition of total slack.

In this work we will use (quantitative formulations of) the qualitative formulations provided above. That is, we define:

Definition 1 (Free Slack)**.** The *free slack* of job i (FS_i) is the amount of time by which it can slide without delaying any of its successors or increasing the makespan:

$$FS_i = \min\{\min_{j\in(\sigma_i\cup\sigma_i^M)}\{s_j - (s_i + p_i)\}, C_{\max} - (s_i + p_i)\} \tag{2.1}$$

Note this is always non negative in a feasible schedule.

Definition 2 (Total Slack)**.** The *total slack* of a job i (TS_i) is how much a job can slide without increasing the makespan:

$$TS_i = s_i(LSS) - s_i(ESS) \tag{2.2}$$

2.1.3 Probability distribution independent definitions of Robustness (Estimation Measures)

The following definitions differ more from the qualitative definition of robustness than those in the previous section. However, they do not require information about the distribution of job processing times and thus may be more suited to local search approaches, or problem instances in which the distribution is not known.

[AH05] qualitatively defines robustness as "We define the robustness of a schedule, as its ability to cope with 'small' increases in the time duration of some activities that may result from uncontrollable factors (i.e. with a limited effect on the completion time of the project)." Their qualitative definition corresponds to the definition of quality robustness in [HL04b],[Von+05]. Their quantitative definition however seems more a definition of solution robustness. They consider what they call Free Slack (whose quantitative definition matches our definition of Total Slack).

Chtourou et al.[CH08] introduce solution robustness measures based on what they call Free Slack (but whose quantitative definition matches our definition of Total Slack). The authors propose twelve measures based four different weightings of three measures. The three measures are sums of:

- The slack $TS_i = s_i(LSS) - s_i(ESS)$
- A binary indicator BTS_i denoting if the free slack exceeds some fraction γ of the job duration:
$$BTS_i = \begin{cases} 1, & \text{if } TS_i \geq \gamma p_i \\ 0, & \text{otherwise} \end{cases}$$
- Slack with an upperbound given by a fraction of the job duration:
$$UTS_i = \min\{TS_i, \gamma p_i\}$$

The sum of these measures without any further weighting form the first 3 measures. The authors argue that if the processing time of a job with many successors increases, it is more likely to influence $C_{\max}$. So a second set of 3 measures is obtained by using the number of successor jobs as a weight. Other weightings are based on resource requirements, which are out of scope for this paper. To utilize the Free Slack heuristics, one first needs a baseline schedule. The authors therefore adapt a two stage approach: First a schedule is created that minimizes $C_{\max}$ (heuristically), then the heuristic is re-run with the objective of maximizing the Robustness Measure subject to the constraint the $C_{\max}$ does not increase compared to the found optimum.

[HAH11] argue that a schedule in which successive jobs are frequently assigned to a single machine are more likely to be robust. Thus they argue that a good robustness measure is for a job set J and machine set M: $\sum_{j \in J} \sum_{m \in M} \gamma_{jm}$, where

$$\gamma_{jm} = \begin{cases} 1, & \text{if } \exists i \in \sigma_j \text{ that is scheduled on machine } m \\ 0, & \text{otherwise} \end{cases}$$

[RAH] similarly aim to maximize the number of successive job pairs that are executed on the same machine. That is, they measure robustness by $\sum_{(i,j)\in A} f(i,j)$ where

$$f(i,j) = \begin{cases} 1, & \text{if } i \text{ and } j \text{ are assigned to the same machine} \\ 0, & \text{otherwise} \end{cases}$$

[HHE10] propose robustness measures for the discrete time/cost trade-off problem. They refer to [CH08] and their measures are again based on slack, however now on Total Slack (TS). Many measures involve the *slack duration ratio* (SDR), which is the ratio of the total slack and the expected processing time of a job: $SDR_i = \frac{TS_i}{p_i}$. The intuition is that jobs with longer processing times are likely to have larger disruptions in the absolute sense. Thus these jobs require more buffer. New Robustness Measures include:

- Slack utility functions where the number of successors is weighted by $\sum_{j=1}^{\lceil SDR_j \rceil} e^{-j}$
- The dispersion of slacks, measured by the coefficient of variation of the SDR
- The percentage activities for which the SDR is less than some threshold (the authors use 0.25), which they call *potentially critical* jobs.
- The project buffer size as percentage of the project deadline
- The coefficient of variation of makespan: $\frac{E(C_{\max})}{Var(C_{\max})}$ The coefficient of variation of makespan also appears in [Pin12].

2.1.4 Heuristic solutions to Robust problems related to Stochastic Parallel Machine Scheduling.

Techniques for solving deterministic problems often do not solve the corresponding stochastic problem, however they frequently form a basis for a (heuristic) approach. One such frequently used method due to Goldratt is the *critical chain buffer managing* (CCBM) approach. This approach along with its advantages and drawbacks is considered in [DH02], Chapter 10.

Another example where the deterministic approach is adapted to solve a stochastic problem is [AH08]. They adapt the Moore-Hodgson algorithm for $1|d_j|U$ to exactly solve $1|\mathbf{p}_j, d_j|U$ using a chance constraint.

We refer the reader to [Pin12], Chapter 12, for more solutions to stochastic problems adaptations from deterministic problems. One noteworthy result they show is that for scheduling on two parallel machines where the jobs are distributed exponentially with rate λ_j, i.e. for $P_2|\mathbf{p}_j \backsim Exp(\lambda_j)|E(C_{\max})$, optimal solutions are found using LEPT (least expected processing time) rule, however examples are given that show LEPT is not optimal for every distribution of processing times. A similar rule does not exist for the deterministic variant. This shows that stochastic problems are not necessarily harder than deterministic ones and that better solutions than attempting to modify the deterministic approach may exist.

[HL04c] develop a heuristic for construction solution robust schedules for the SRCPSP. The 0 lag finish-start precedence relations are given by a set of arcs. The heuristic, called the *adapted float factor model*, ADFF, works as follows. Given some project due date δ each job J_i is scheduled some time after $s_i(ESS)$. Let the time to finish be defined as $TF_i(ESS) = \delta - s_i(ESS)$. Each job is assumed to have a probability of being overdue Pr_i and a cost per time unit that it is late c_i. The weight on a precedence relation $((i, j)$ is given by $p_i c_i$. Furthermore, let the float factor be $\alpha_i = \frac{\beta_i}{\beta_i + \phi_i}$, where β_i (resp. ϕ_i) is the sum all A-arcs before (resp. after) i in the network. The start time of J_i is scheduled to be $s_i = ESS_i + \alpha_i TF_i(ESS)$. This method shows good results for providing solution robustness. It may therefore be interesting as (part of) a robustness measure.

[Von$^+$05] consider the trade-off between quality and solution robustness for the SRCPSP. They compare traditional critical chain buffer managing (CCBM) with a modified version of CCBM and with the adapted float factor (ADFF) method described above. They use RanGen software by [DVH03] to generate problem instances and test their schedules using simulation. They assume job processing times are drawn from a right-skewed beta distribution with mean $= p_j$, minimum $= 0.5p_j$ and maximum $= 2.25p_j$. Their results show that as making the final deadline becomes more important ADFF is preferable to CCBM (the quality robustness difference becomes small, whereas the solution robustness difference remains large). This is surprising as ADFF was developed to provide solution robustness, whereas CCBM was developed to provide quality robustness, but clearly ADFF can provide good quality robustness as well. The authors do note however, that as the problem becomes harder (more jobs or more precedence relations), the makespan increases, so even a small relative increase in necessary due date of ADFF over CCBM may be too large to be deemed acceptable.

2.2 Metaheuristics for problems related to Stochastic Parallel Machine Scheduling

We will be solving Stochastic Parallel Machine Scheduling using a metaheuristic[3] approach. The performance of any local search based procedure is highly dependent on solution representation and the neighborhood operator. In the following we discuss in detail some neighborhood operators used in literature for SPMS and similar problems.

A general overview of basic neighborhood operators for machine scheduling is provided in [AL03]. They distinguish assignment and sequencing problems. For sequencing, they list *Transpose* - swapping two sequential jobs -, *Insert* - moving a single job to a different position in the sequence), *Swap* - swapping two non-adjacent jobs - and *Block Insert* - moving a sub sequence of jobs to a different position in the sequence.

For assignment they list *Reassign* - Remove a job from one machine and reassign it to another, *Swap* Take two jobs assigned to separate machines and reverse the assignment and *k-Reassign* Remove up to k jobs and reassign them to new machines. They observe it may be preferable to reduce neighborhood size by considering only critical jobs. Thus defining *Critical Reassign*, *Critical Swap* and *Critical k-Reassign* in which at least one of the jobs considered is critical.

They briefly note that for a problem such as $P|r_j|\sum C_j$ the above operators may be generalized appropriately. e.g. *Insert* takes a job from any machine and reinserts it in any new position (either on the same machine or not).

Finally it is worth noting that they provide an overview of earlier work on heuristic approaches to parallel machine problems.

[MG10] propose two closely related optimum connected neighborhood functions for the Flexible Job Shop Problem based on inserting a critical operation in the best way. We discuss this work in detail, as we use their operator as part of our local search procedure.

The authors consider operations where a job j is removed from a machine, then reinserted in a feasible way on to a machine (this may be the same machine). Removal of a job j from a machine induces a reduced graph by removing the corresponding machine arcs. Note that j still exists in the graph and may still be connected to other nodes via precedence arcs. In their Flexible Job Shop Problem, the processing time of a job depends on the machine it is assigned to. The authors resolve this where needed by

[3]Section 1.1.4 provides a basic introduction to metaheuristics.

assuming that the processing time of the job is equal to the processing time of that job on the last machine it was assigned to. This works nicely for parallel machine scheduling, as the processing time is machine independent, so in particular is always equal to the time on the last machine the job was assigned to.

Finally, the authors discuss two neighbourhood operators. Jobs considered for a move are all those on a particular critical path: Starting with the first job j of any critical path, take a successor job $i \in \sigma_j$ if i is part of the critical path, otherwise take the next job on the machine (σ_j^M).

In the first operator, for every such critical job j, for every machine m, the schedule resulting from inserting j on m is calculated and the best improvement is added to the neighborhood of the current solution. That is, the neighborhood of a schedule is all makespan minimizing moves of a job from the particular critical path to a machine (this may be the machine the job originally came from). The neighborhood can be determined in $\mathcal{O}(N)$, where N is the number of jobs. This operator is not optimum connected, which means that there are states from which the local searcher will never reach the optimum.

A second approach is that the neighborhood of a solution is all optimal moves of a job to another machine, and all feasible moves on the same machine. The difference with the first approach is that on the same machine, all feasible moves are considered instead of only the locally optimal moves. This neighborhood is larger than that of the first approach (it is a strict superset thereof), however finding the feasible solutions is faster than finding the optimal ones. In this case the neighborhood can be determined in $\mathcal{O}(\lg(N))$ (where again, N is the number of jobs). Note that the time taken to compute the neighborhood is not the limiting factor. A large neighborhood will take much longer to search than a small one. The reason to allow more states, is that this neighborhood operator is optimum connected.

The authors use these in a Tabu-search and report that their best results were found using the first (non-optimum connected) neighborhood operator, with the approximation explained above.

[ABH13] find robust solutions for the stochastic job shop problem. They use simulation to determine expected makespan, which they use as a fitness function for the local search procedure. Their results show their approach outperforms classical methods (methods based on using deterministic makespan as an expected makespan approximation).

[PAH] extend this idea by providing other efficient ways of estimating the makespan. They use an Iterated Local Search approach to solve Robust Parallel Machine Scheduling.

Their aim is to minimize the expected makespan (quality robustness). The key to their approach is an efficient estimation of the expected makespan, allowing expected makespan to be part of the fitness function. Their approach is faster than that of [ABH13] and finds better solutions than the result sampling approach when the number of samples is less than 100. Their work illustrates that efficient estimation in a local search can be very effective at solving robust scheduling problems. However, they only consider minimizing the expected makespan as a robustness measure. Although this measure may (or may not) be a good measure to determine quality robustness, it may not be a good measure to provide solution robustness.

[BL09] use a Greedy Randomized Adaptive Search Procedure (GRASP) to solve SRCPSP with 0 lag finish-start precedence relations. They aim to find a scheduling policy to minimize the expected makespan. They use the natural list representation for a policy. During their search, they keep track of the best n solutions. These solutions are recombined to a new list L' in one of two ways. Firstly from the set of best solutions, a random solution L is chosen. From L a set of jobs are added iteratively to L' in order of first precedence feasible job in L. The number of job addition iterations is uniform random between two parameters. This is the most likely procedure. To increase diversity there is also a small probability the next job added to L' will be the first precedence feasible job in latest feasible start schedule, or a random precedence feasible job, or the last precedence feasible job in L. After the new schedule L' is built, it is improved with a local search. Finally its expected makespan is estimated using a simulation with a small number of runs (which is inaccurate but fast). It is then compared to the worst schedule in the population and if L' is better it replaces the worst solution in the population. Although the authors do not mention it, using only a small number of simulation runs also increases diversity as it allows potentially worse solutions in to the population. Two final results are relevant to this thesis. First the authors find the best performing local search to be a two-point crossover for permutations by [Har98] that compares two schedules before and after *double justification* (described in the following paragraph). Second, the authors find a slight improvement when using descriptive[4] over random sampling, especially when the number of replications is low.

A successful technique for improving Local Search for the deterministic RCPSP is double justification described in [VBQ05]. To explain the technique, we first list some standard terminology: Schedules are divided into four classes [TB06] (although the first two definitions below are due to [RCL17]). From small to large, these are:

[4]A form of sampling where the sampled values are chosen purposefully and are thus not random. See e.g. [Sal]. We do not use this approach.

- The set of *Non-delay* schedules : Any schedule in which an activity cannot start earlier without delaying another activity (even if preemption is allowed).
- *Active* schedules: Any schedule in which, without activity preemption, no activity can start earlier without delaying another activity.
- *Semi Active* schedules: Any schedule for which, without changing the assignment of jobs to machines, no activity can start earlier without delaying another activity.
- Finally, schedules with insertion of machine idle time are those in which idle time is voluntarily inserted.

For $C_{\max}, L_{max}$ and other regular optimization criteria, Baker showed that the set of active schedules is dominant [TB06]. However, this is not the case for stochastic problems [RCL17]. We return now to [VBQ05]. Given a schedule S, they defines right (resp. left) justification of an activity j as finding a schedule S' where the start times of all other jobs are unchanged (i.e. $\forall i, (i \neq j \rightarrow s_i = s'_i)$) and job j is started as late (resp. early) as possible. Justifying activities in decreasing (increasing) order of finishing time provides a right active (left active) schedule. Double justification is the act of first right justifying then left justifying a schedule (which is therefore left active). They show that many algorithms for deterministic RCPSP benefit from this approach. Note that as active schedules are not dominant for stochastic problems, it is an open question as to whether these results are transferable to techniques for stochastic problems. However in an approach such as that by [BL09] where the Local Search is on the deterministic problem, one should probably include this technique.

2.3 Conclusion

The need for robustness is apparent to many authors. However, they may differ on their interpretations and definitions thereof. The qualitative definitions we have given in the introduction seem to be the most widespread. Several authors (such as [Bal07; BL09; MG10; ABH13; PAH]) have had good results for robust problems using local search approaches. Building upon work by [AH05; CH08; GS08; PAH] we will attempt to determine good robustness estimators for use in a local search approach. Of particular interest is [PAH], who solve Stochastic Parallel Machine Scheduling using iterated local search with good results. Their approach however considers robustness only in the sense of minimizing expected makespan. This differs from many definitions of robustness, although the results in [BL09] support this approach. We will attempt to solve Stochastic

Parallel Machine Scheduling while taking into account other common definitions of robustness. By comparing our results to those of [PAH] we will be able to determine the trade-offs involved between considering robustness and focusing on expected makespan minimization.

Chapter 3

Robustness: Definitions and estimation Measures

This chapter discusses the robustness definitions and robustness measures we use. These are all based on the literature in the previous chapter.

3.1 Quantitative robustness definitions used.

Intuitively, a robustness measure is a way of determining how robust a schedule is. This simple intuition is surprisingly hard to formalize. Indeed, robustness does not have a unique formal definition in literature. Based on other works, we present five quantitative ways to measure robustness. These five measures have been chosen in such a way that they:

1. Match or resemble a qualitative definition from the literature.
2. Can be calculated for any schedule, given enough simulations run on a single set of input parameters.

These measures are *not* required to be quick to calculate. Because we will use these measures to analyze the performance of our schedule creating algorithm, we will call them *performance measures* where ever possible. We do so to distinguish them from the robustness measures presented in Section 3.2, which we will be using during the local search.

Quality Robustness Definitions used

First we present two definitions of quality robustness. Recall that one definition of quality robustness is "Quality robustness (makespan performance) is measured by the probability that a project ends within the projected deadline."[Von+05]. That is, quality robustness is related to whether the entire schedule finishes on time. It does not matter what individual jobs do during the process, as long as all jobs are finished by some date.

One measure could be the percentage of simulation runs that complete before the deadline. Mathematically, let H be the Heaviside step function:

$$H : \mathbb{R} \to \{0,1\}, H(x) = \begin{cases} 1 \text{ for } x > 0 \\ 0 \text{ for } x \leq 0 \end{cases} \tag{3.1}$$

Let $C_{\max}(S)_i$ be the makespan in the i-th simulation of schedule S. Then the percentage of simulations out of n_{sim} runs with deadline δ, that are completed on time (denoted $\text{QR}'_1(S,\delta)$) would be given by:

$$\text{QR}'_1(S,\delta) = \frac{\sum_{i=1}^{n_{\text{sim}}} H(\delta - C_{\max}(S)_i)}{n_{\text{sim}}}$$

However this is not entirely satisfactory, as the due date is not a parameter of the SPMS problem, as [BL09] point out. They solve this by finding the due date for which in expectation a certain fraction of solutions are on time. This leads to our first quality robustness measure, the date such that the fraction of on time runs is at least π:

$$\text{QR}_1(S,\pi) = \min\{\delta | \text{QR}'_1(S,\delta) \geq \pi\} \tag{3.2}$$

Note that this is simply an elaborate way of saying we take the 100π-th percentile of the simulated makespans. In particular, we will use $\pi = 0.95$ in our experiments. For brevity, we will use the notation $\mathcal{C}^{0.95}$ as shorthand for $\text{QR}_1(S, 0.95)$.

A second measure for quality robustness that does not rely on deadlines is the *variation coefficient*, defined as the standard deviation ($\hat{\sigma}$) over the sample mean ($\hat{\mu}$) of the makespan (denoted VarCo):

$$\text{QR}_2(S) = \frac{\sigma_{\hat{C}_{\max}}}{\mu_{\hat{C}_{\max}}} = \text{VarCo} \tag{3.3}$$

This indicates how certain one is about an expected completion time and thus fits the intuitive definition of quality robustness. Furthermore, if the expected makespan is known and with additional information about the shape of the makespan distribution, this value allows one to determine (or estimate), for any probability π, $\mathrm{QR}_1(S, \pi)$.

Solution Robustness Definitions used

To measure solution robustness, we will use two definitions. Recall that solution robustness describes the likelihood of jobs starting on time, that is, that a schedule can be used without having to change the time in which resources are used. The first is again a definition similar to [BL09], namely the unweighted sum of absolute deviations between the actually realized activity starting times and the starting times indicated in the initial projected schedule as anticipated before project execution. We call this measure the sum of *linear start delays* LSD. Let s_j be the planned start time of job j in schedule S. And let $\mathbf{s_{ij}}$ be the realized time of job j in the i-th simulation run.

$$\mathrm{QR}_3(S) = \sum_{i=1}^{n_{\text{sim}}} \sum_{j \in J} |\mathbf{s_{ij}} - s_j| = \mathrm{LSD} \tag{3.4}$$

We choose this measure because it may be argued that in a practical setting, a job starting early provides as much planning difficulty as a job starting late (i.e., it is sensible that the function is symmetric). However it should be noted that in our setup, early starting is not allowed, so $\mathbf{s_{ij}} - s_j \geq 0, \forall i, j$ (so taking the absolute value is not necessary).

We also use the average percentage of jobs that start on time in a run. We will call this the *percentage of on time jobs* or *start punctuality* (SP). Again, if H is the Heaviside step function (eqn. 3.1) and n is the number of jobs, we have:

$$\mathrm{QR}_4(S) = \frac{\sum_{i=1}^{n_{\text{sim}}} \sum_{j \in J} H(\mathbf{s_{ij}} - s_j)}{n_{\text{sim}} \cdot n} = \mathrm{SP} \tag{3.5}$$

Expected Makespan

Finally, we will also consider the sample mean of the makespans, in order to determine the expected makespan. Although not intuitively a definition of robustness, there are some reasons to include this measure. First, if costs for delays are linear, a practitioner may not care what the makespan distributions of two different solutions look like. In this case he may wish to use the solution with the lowest expected makespan. By doing so, the practitioner is using some information about the uncertainty of the plans and thus could be said to have chosen the more robust schedule. Second, this measure is often used in literature so it is useful if one wishes to compare results of their algorithm. Finally, the results in [BL09] show a strong correlation between the expected makespan and quality robustness measures. So perhaps the expected makespan functions as a quality robustness measure.

We take the sample mean of the makespans to estimate the expected makespan:

$$\mathrm{QR}_5(S) = \frac{\sum_{i=1}^{n_{\mathrm{sim}}} C_{\max}(S)_i}{n_{\mathrm{sim}}} = \mu_{\hat{C_{\max}}} \tag{3.6}$$

In the next section, we turn to robustness measures that are not dependent on sampling and thus can be used in a local search approach.

3.2 Overview of Robustness Measures considered and Notation used

3.2.1 Desirable properties of Robustness Measures

We wish to determine which Robustness Measures perform well. This section will first discuss what is desired of a robustness measure.

Recall that the idea is to use a Robustness Measure during a subroutine to select the better of two different schedules. We assume that both the problem instance and optimization objective[1] are known. It is important that this decision can happen quickly whatever the problem size. Therefore, we say that:

Definition 3. A robustness measure *performs well* if, for a given problem instance and for a given objective, the robustness measure can distinguish schedules with a higher and lower objective value in linear time with relation to the number of jobs and precedence relations.

[1]In this work, the optimization objective is one of the five robustness measures presented in the previous section ($\mathcal{C}^{0.95}, \mathrm{VarCo}, \mathrm{SP}, \mathrm{LSD}, \mu_{\hat{C_{\max}}}$)

Note that it is not a requirement that the robustness measure be highly correlated with the objective. The rank based correlation matters more, as illustrated by the following example.

Consider three solutions with objective value 1,2 and 3 respectively. Consider further robustness measures RM_1 with values $1, 2.5$ and 2.4 or RM_2, with values $-1, \pi$ and 2^{10} respectively. RM_1 is closer to the actual objective values than RM_2, however the ordering is wrong. Thus RM_2 allows us to better distinguish solutions, and is the better robustness measure.

Common properties or robustness measures in literature

A general formula to create a robustness measure is to choose some property of a job in a schedule (e.g. the processing time or free slack of that job), some weighting for this property (e.g. the number of predecessors that job has) and some way of combining these properties (e.g. summing them). For example, the makespan assuming jobs have deterministic processing times is a common robustness measure[2] for estimating expected makespan. The job property used is the deterministic processing time. The weighting is multiplying by one (unweighted). The combination is the maximum over all paths of the sum of properties along that path.

In the following sections we discuss slack properties, weightings and ways to combine properties to create robustness measures.

3.2.2 Slack based measures

Slack based robustness measures are some combination of some type of slack for each job. To distinguish between the robustness measure of a schedule, and the slack measurement on a single job, we call the latter the *slack property* of a job.

For a given job j we can calculate the slack properties listed in Table 3.1.

Each property can be weighted. This work only considers two weightings, namely not using a weighting, or weighting by the number of successors.

That is for any slack property X_j, we consider X_j and $|\sigma_j| \cdot X_j$

[2] I am stretching the intuitive definition here, but it fits the formal definition in the previous section

Notation	Definition	Description
FS_j	$\min\{\min_{i\in\sigma_j}\{s_i-(s_j+p_j)\}, C_{\max}-(s_j+p_j)$	Free slack of job j.
BFS_j^γ	$\begin{cases} 1 \text{ if } & \geq \gamma \cdot p_j \\ FS_j, 0 \text{ otherwise.} \end{cases}$	Binary value indicating if the free slack exceeds a percentage of the processing time.
UFS_j^γ	$\min\{\gamma \cdot p_j, FS_j\}$	Free Slack with upperbound.
TS_j	$s_j(LSS) - s_j(ESS)$	The Total slack of job j is the difference in starting times of j in the earliest / latest start schedules.
$BTS_j\gamma$	$\begin{cases} 1 \text{ if } TS_j \geq \gamma \cdot p_j \\ 0 \text{ otherwise} \end{cases}$	Binary value indicating if the total slack exceeds a percentage of the processing time.
UFS_j^γ	$\min\{\gamma \cdot p_j, TS_j\}$	Total Slack with upperbound.
SDR_j	TS_j/p_j	Slack Duration Ratio

Table 3.1: Robustness measure list. For symbols, refer to A.1

Slack based robustness measures are some way of combining slack properties. In particular, we consider for any slack property X_j:

- The sum of all slack properties: $\sum_j X_j$
- The average of all slack properties: $\frac{\sum_j X_j}{n}$, where n is the number of jobs in the problem instance.
- The minimum of all slack properties $\min_j\{X_j\}$
- The minimum of the sum of slack properties along any path: $\min_{(0,*)\in P}\{\sum_{j\in(0,*)} X_j\}$, where P is the set of paths from the start job 0 to the completion job $*$.

3.2.3 Theoretical Analysis of RMs

Some drawbacks of robustness measures can be illustrated with thought experiments.

For example, consider a problem instance with six jobs with processing times 1, release dates 0, without precedence relations and on two machines. Let us compare the schedule S_1 with four jobs on one machine and two on the other, with the schedule S_2 where the jobs are evenly balanced between both machines. Note that the unweighted sum of free slack and total slack both have value 2 and 4 respectively in S_1 and value 0 in S_2. An approach that maximized free or total slack would thus favor S_1. However, this is clearly a poor approach when attempting to minimize the average makespan.

Second, note that if the critical path has no slack, the minimum total slack and minimum free slack both of a job and along a path are 0. The exception is if release dates force the critical path to have slack somewhere, or if slack is inserted into the critical path. Our problem instances do not force slack on the critical path and we do not consider inserting slack into the critical path. Thus the minimum properties mentioned above are no use for this work.

3.2.4 Statistical approximation in linear time.

Another estimation procedure, based on statistical rules, is presented in [PAH]. The approach works both for known and unknown job completion time distributions. In the latter case it relies upon an approximation.

It is important to note that [PAH] consider non-zero start-start precedence relations, whereas in this work we assume zero, finish-start precedence relations. This allows us to simplify their process.

When a job has two or more predecessors, its starting time is dependent on the maximum completion time of these predecessors. Given a method of calculating or estimating the maximum of two distributions, it is possible to estimate all job starting times using a dynamic programming approach. For now let us assume we have a method of calculating the maximum of two distributions[3].

Algorithmic approach

We will first describe our algorithm and argue that it is correct. Our algorithm processes the jobs in precedence order. Recall that any feasible schedule admits a complete order. We determine the approximated start time distribution of the current job j, denoted S_j. We use capital S_j to distinguish from the planned start time s_j. Similarly, in the following C_j is the estimated completion time distribution. For our problem, with 0-finish-start lag relations, we have:

$$S_j = \max\{r_j, C_{\pi_j^M}, \max_{i \in \pi_j}\{C_i\}\}$$

I.e., the start time of job j is the maximum of its release date, the completion time of its machine predecessor and all its precedence graph predecessors.

[3]This method is presented in the subsection *"Calculating the Maximum of two random Normal variables"*.

The key is to estimate the maximum of the distributions as well as possible. The maximum increases due to uncertainty in the jobs. A mathematical formulation is given in Section 30. For now, the following intuition suffices. Given random variables X_1, X_2, X_3 sampled from distribution $\mathcal{D}$, it must hold that $E(\max(X_1, X_2)) < E(\max(X_1, X_2, X_3))$. Thus we must be careful not to account for the uncertainty in a job twice. To determine which jobs should influence the approximated start time, we consider the following four cases for precedence relation (i, j):

1. i and j are assigned to the same machine, i is the machine predecessor of j.
2. i and j are assigned to the same machine, i is assigned before j, but is not its machine predecessor.
3. i and j are assigned to different machines. There is another predecessor of j, that is assigned to the same machine as i, some time after i.
4. i and j are assigned to different machines. There is no other predecessor of j, that is assigned to the same machine as i, some time after i.

An example of these cases is also given in Figure 3.1.

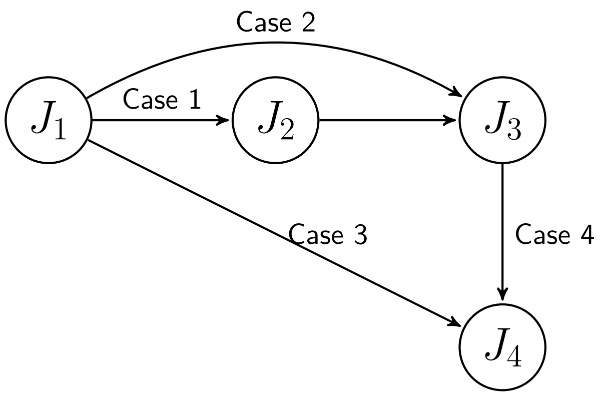

$$\begin{tikzcd}
J_1 \arrow[r, "\text{Case 1}"] \arrow[rr, bend left, "\text{Case 2}"] \arrow[rrd, "\text{Case 3}"] & J_2 \arrow[r] & J_3 \arrow[d, "\text{Case 4}"] \\
 & & J_4
\end{tikzcd}$$

Figure 3.1: An example of the cases we distinguish in the statistical approximation approach.
Case 1: J_1 and J_2 are on the same machine. J_2 is the successor of J_1.
Case 2: J_1 and J_3 are on the same machine. J_3 is not the successor of J_1.
Case 3: J_1 is a predecessor of J_4 and on a different machine. There is another predecessor of J_4 (J_3) scheduled on the same machine as J_1, sometime after J_1.
Case 4: J_3 is a predecessor of J_4 and on a different machine. There no other predecessor of J_4 scheduled on the same machine as J_3, anytime after J_3.

In the first case and in the fourth case, it is clear that j can be delayed by i and that there is no other way this information can be carried. So our algorithm should use S_i when determining S_j.

In the second and the third case, although it is clear that i influences j, it does so through an intermediate job k. Potentially even through more than one job. Note that in any total order, S_k will be determined before S_j (as k is a machine predecessor of j). Thus we should not use S_i directly when determining S_j. Note that S_i does affect S_j: S_i will be used to determine S_k, which will in turn affect S_j. If we were to use S_i, we would be considering its uncertainty twice. Thus in the second and third case we do not update S_j.

Algorithmic implementation

Now we turn to the implementation of the algorithm (given in algorithm 1. We assume that the distribution of the job processing times is unknown. Without any information on the distribution, we work with an implementation that assumes job processing times are Gaussian. In the case that the distribution is known, this approach is easily adaptable to other distributions if a way of estimating the maximum of two distributions exists (such as for the Exponential distribution). To modify the approach we discuss here, one simply has to replace the function that determines the maximum of two distributions (algorithm 2) with an appropriate function.

There is some abuse of notation involved in the algorithm. In some cases, the start time of a job is known exactly (e.g. when it can start at time 0 or at its release date r_j). In such a case, we will use the notation $S_j = N(s_j, 0)$, by which we mean that 'distribution' of the start time has zero variance. We do so because we can then take the distribution maximum between S_j and some other distribution (that may have some variance). This notation is true to the way in which the algorithm is implemented: it does not distinguish between constants and 'distributions' with zero variance. This happens in line 3 of algorithm 1.

For now, assume we can calculate the maximum of two Gaussian distributions (see Section 30). We assume (with some error) that the resulting distribution is again Normally distributed. The approach is to repeatedly use this maximum approximation to calculate all the starting times using a Dynamic Programming approach. We first present the algorithm, then discuss the assumptions made therein.

Algorithm 1: $N(\mu^{\mathrm{NA}}_{C_{\max}}, \sigma^{\mathrm{NA}}_{C_{\max}})(S)$: For a given schedule S, calculate for each job C_j, the approximated distribution of completion times for job j, for 0 lag finish-start precedence relations under the assumption that processing time of jobs are normally distributed.

```
Data: A feasible schedule.
Result: For every job $j$, the distribution $C_j = N(\mu_{C_j}, \sigma^2_{C_j})$ approximating the
        completion times of each job.
1  for Each job $j$ in precedence order do
2      if $j$ is the first job on the machine then
3          $S_j \leftarrow N(0,0)$ // Set the start time to 0.;
4      else
5          $i \leftarrow \pi^M_j$;
6          $S_j \leftarrow C_i$ // Case 1 ;
7      end
8      // Find the last predecessor of $j$ on each machine;
9      for $J_k \in \pi_j$ do
10         $\mu \leftarrow$ Machine that $J_k$ is ;
11         $\mathrm{LMP}^{\mu}_j \leftarrow$ null // Initialize the last predecessor of $j$ on machine $\mu$ ;
12         if $S_{J_k} > S_{LMP^{\mu}_j}$ then
13             $\mathrm{LMP}^{\mu}_j \leftarrow J_k$ ;
14         else
15     end
16     for Each machine $\mu$ do
17         if $j$ is on machine $\mu$ then
18             //Case 1 (handeled above, so ignore), or Case 2 (ignore) ;
19         else
20             //Different machine, update $S_j$ ;
21             if $LMP^{\mu}_j$ exists then
22                 //Case 4 ;
23                 $S_j \leftarrow \textsc{DistrMax}(S_j, S_{\mathrm{LMP}^{\mu}_j}, \text{independent})$;
24             else
25         end
26     end
27     $S_j \leftarrow \textsc{DistrMax}(S_j, r_j, \text{independent})$;
28     $C_j \leftarrow S_j + N(p_j, 0.3p_j)$ //Given the start time, we calculate the completion time
         distribution;
29 end
30 $C_{\max} \leftarrow \textsc{DistrMax}_j\{C_j\}$;
```

The algorithm runs in $\mathcal{O}(n+R)$ where n is the number of jobs and R is the number of precedence relations and machine arcs. Intuitively this is clear: each job is considered once by each of its successors and its machine successor.

Calculating the Maximum of two random Normal variables.

The maximum of two random Normal variables $D_1 \backsim N(\mu_1, \sigma_1^2), D_2 \backsim N(\mu_2, \sigma_2^2)$ may be calculated as described in algorithm 2. The independent case is due to [NK08]. Let $\theta = \sqrt{\sigma_1^2 + \sigma_2^2}$, ϕ be the standard normal probability density function and Φ be the standard normal cumulative probability function. The mean of the maximum of D_1 and D_2 is given by:

$$\mu_3 = \mu_1 \cdot \Phi(\frac{\mu_1 - \mu_2}{\theta}) + \mu_2 \cdot \Phi(\frac{\mu_2 - \mu_1}{\theta}) + \theta \cdot \phi(\frac{\mu_1 - \mu_2}{\theta})$$

The standard deviation of the maximum of D_1 and D_2 is given by:

$$\begin{aligned}\sigma_3^2 &= (\sigma_1^2 + \mu_1^2 - \mu_1) \cdot \Phi(\frac{\mu_1 - \mu_2}{\theta}) \\ &+ (\sigma_2^2 + \mu_2^2 - \mu_2) \cdot \Phi(\frac{\mu_2 - \mu_1}{\theta}) \\ &+ (\mu_1 + \mu_2 - 1)\theta \cdot \phi(\frac{\mu_1 - \mu_2}{\theta})\end{aligned}$$

For the dependent case, we follow the work described in [PAH]: Let $X = \max\{D_1, D_2\}$. For notation, let $\Delta = D_2 - D_1$ be the difference between the random variables. Second, let $E(\Delta|\Delta > 0)$ be the average difference between D_2 and D_1 when D_2 is larger than D_1. Finally, let $\mu_\Delta = E(\max\{0, \Delta\})$.

From probability theory, we have that:

$$\begin{aligned}\mu_\Delta &= Pr(\Delta \leq 0) \cdot 0 \\ &+ Pr(\Delta > 0) \cdot E(\Delta|\Delta > 0) \\ &= Pr(\Delta > 0) \cdot E(\Delta|\Delta > 0)\end{aligned}$$

We will show how to calculate $Pr(\Delta > 0)$ and $E(\Delta|\Delta > 0)$ later. Once we know these, we can calculate $\mu(X)$:

$$\mu(X) = \mu_1 + \mu_\Delta$$

The variation of X, if the distributions are unknown, is hard to calculate. [PAH] approximate by saying that the contribution to the variance of X by D_1 and D_2 is directly proportional to the probability that the respective random variable is larger than the other. That is:

$$\sigma^2(X) \approx Pr(\Delta \leq 0) \cdot \sigma_1^2 + (1 - Pr(\Delta \leq 0)) \cdot \sigma_2^2$$

Note that as D_1 and D_2 are normally distributed, so too is Δ, with $\mu_\Delta = \mu_1 - \mu_2$ and $\sigma_\Delta^2 = \sigma_1^2 + \sigma_2^2$. Let $\alpha = \frac{-\mu_\Delta}{\sigma_\Delta}$ be the transformation required to change Δ into a standard normal distribution. Then, $\Pr(\Delta \leq 0) = \Phi(\alpha)$, where Φ is the cumulative probability density function of the standard normal distribution.

Finally, from [Gre03], we have:

$$E(\Delta|\Delta > 0) = \mu_\Delta + \sigma_\Delta \cdot \frac{\phi(\alpha)}{1 - \Phi(\alpha)}$$

Here ϕ is the probability density function of the standard normal distribution.

The above allows us to create a function DISTRMAX$(N_1, N_2) \rightarrow N_3$ that creates a new normal distribution based on the approximated maximum of two normal distributions:

Algorithm 2: DISTRMAX(N_1, N_2): A function that returns the normal distribution resulting from approximating the maximum of two normal distributions.

```
Data: Two normal distributions $N_1 = N(\mu_1, \sigma_1^2)$ and $N_2 = N(\mu_2, \sigma_2^2)$
Result: A normal distribution $N_3$, which is an approximation for the distribution of
        $\max D_1, D_2$ for two random variables $D_1 \backsim N_1$, $D_2 \backsim N_2$.
if $N_1, N_2$ are independent then
    $\theta \leftarrow \sqrt{\sigma_1^2 + \sigma_2^2}$ ;
    $\mu_3 \leftarrow \mu_1 \cdot \Phi(\frac{\mu_1-\mu_2}{\theta}) + \mu_2 \cdot \Phi(\frac{\mu_2-\mu_1}{\theta}) + \theta \cdot \phi(\frac{\mu_1-\mu_2}{\theta})$;
    $\sigma_3^2 \leftarrow (\sigma_1^2 + \mu_1^2 - \mu_1) \cdot \Phi(\frac{\mu_1-\mu_2}{\theta}) + (\sigma_2^2 + \mu_2^2 - \mu_2) \cdot \Phi(\frac{\mu_2-\mu_1}{\theta}) + (\mu_1 + \mu_2 - 1)\theta \cdot \phi(\frac{\mu_1-\mu_2}{\theta})$
else
    $\mu_\Delta \leftarrow \mu_1 - \mu_2$;
    $\sigma_\Delta \leftarrow \sqrt{\sigma_1^2 + \sigma_2^2}$;
    $\alpha \leftarrow (-\mu_\Delta / \sigma_\Delta)$;
    $Pr(\Delta > 0) \leftarrow 1 - \Phi(\alpha)$;
    $E(\Delta|\Delta > 0) \leftarrow \mu_\Delta + \sigma_\Delta \cdot \frac{\phi(\alpha)}{(1-\Phi(\alpha))}$;
    $\mu_\Delta \leftarrow Pr(\Delta > 0) \cdot E(\Delta|\Delta > 0)$;
    $\mu_3 \leftarrow \mu_1 + \mu_\Delta$;
    $\sigma_3^2 \leftarrow Pr(\Delta > 0) \cdot \sigma_2^2 + (1 - Pr(\Delta > 0)) \cdot \sigma_1^2$
end
return $<\mu_3, \sigma_3^2>$;
```

Chapter 4

Robustness Measure Evaluation

In this chapter we evaluate the robustness measures presented in the previous chapter. We will first discuss our objective in Section 4.1. Then in Section 4.2 we discuss how the experiments were set up and why. In Section 4.3 we present the results. Conclusions are presented in Section 4.4.

4.1 Aim of this experiment

The goal in this thesis is to determine if it is possible to use robustness measures to steer a local search approach, in order to solve a stochastic problem in a robust way. To this end, in the previous chapter we identified five quantitative definitions of robustness as objective functions one may wish to optimize. Recall that we refer to these as *performance measures*. Furthermore, we have summarized quantitative robustness measures that can be calculated in time linear with the number of jobs and precedence relations. Finally, we determined what it means for a robustness measure to perform well:

A robustness measure performs well if, for a given problem instance and for a given objective, the robustness measure can distinguish schedules with a higher and lower objective value in linear time with relation to the number of jobs and precedence relations.

In this chapter, our aim is to answer the following questions:

- Which, if any, robustness measures perform well, for several different problem instances?

- Which, if any, robustness measures perform well, for several different performance measures?

That is, we attempt to find robustness measures that can be used to steer a local search approach. In the next section, we will discuss the experimental setup to answer these questions.

4.2 Experimental Setup

In order to answer the research questions, we need to have a variety of problem instances and performance measures. Ideally, we would also test our measures on a variety of problems. However, this would expand the scope of this work too much. So we restrict ourselves to stochastic parallel machine scheduling. Section 4.2.1 discusses why we select this problem. Section 4.2.2 discusses which problem instances we use. For each problem instance, we want a collection schedules ranging from 'good' to 'bad'. We emphasize that at this stage we do not attempt to create good schedules, only determine which robustness measures may be useful when attempting to create good schedules. How these are created is discussed in section 4.2.3. Finally, we discuss what we measure and why in section 4.2.4.

We will now go over which problem we will use and why, how we selected problem instances and how we generate schedules.

4.2.1 Stochastic Parallel Machine Scheduling: A suitable test problem

We wish to analyze the influence of using robustness measures in a local search approach. Therefore any problem we consider should be NP-hard in the strong sense as this type of problem is frequently solved using local search approaches. A commonly discussed problem in the literature is the stochastic resource constrained project scheduling problem (SRCPSP). A description of the deterministic version of this problem due to [HK00] is as follows: There are n activities together with two fictitious activities 0 and $n+1$ (the project start, respectively project end dates). Activity j takes time p_j to complete. Performing an activity requires up to K types of resources and requires a known amount ρ_{jk} of each type of resource per time unit (e.g. digging a hole may require several personnel and a machine). For each resource type, the amount of resources available for the entire project may be bounded (e.g. the amount of money for a project may be limited) or unbounded (e.g. diggers are not used-up when used). Preemption is not allowed. Precedence constraints exist between activities. For each type of resource, there is a limit to the number of resources that can be used at any given time. The objective of the RCPSP is to minimize the makespan. Heuristics for this problem are given by by the same authors.

For the stochastic variant, the objective is often minimizing the expected makespan. Examples include [CH08],[BL09] and [Von+05].

A special case of RCPSP is $P|\text{prec}|C_{\max}$. There are n jobs to be performed. Job j requires p_j uninterupted processing time to complete on any of the m identical machines. There are 0-lag finish-start precedence relations between jobs. One can see this is a special case of RCPSP as follows: Activities are now called jobs. There is one resource type ($K = 1$) called a machine. This resource is unbounded over the entire project (machines are not used up when they are used). However, a bound on the number of resources available at any given time does exist: at most all m machines can be active. Every job uses exactly one resource unit per time unit (a job occupies a single machine while it is processed). As in the RCPSP, preemption is not allowed, jobs have precedence relations and processing times.

$P|\text{prec}|C_{\max}$ is still NP-hard in the strong sense [Pin12]. Nevertheless, methods exist to solve the problem exactly. The slightly harder $P_m|\text{prec}, r_j|C_{\max}$ can be solved effectively using column generation [AHK12]. We will denote the variant of this problem with stochastic processing times (represented as $\mathbf{p}_j$) as $P_m|\mathbf{p}_j, \text{prec}, r_j|C_{\max}$ and refer to it as Stochastic Parallel Machine Scheduling (SPMS). [PAH] provides a metaheuristic approach to solve this problem. In a sense the SPMS problem is one of the simplest examples of commonly studied hard problems and is therefore an excellent problem to work on.

4.2.2 Problem Instances

In considering the design of the experiment, it is important to emphasize that robustness measures should not be used to compare the optimality of solutions for different problem instances. As a trivial example, compare two problems with two jobs of unary processing time and a single machine. There are no precedence arcs. In the first problem, let all the release dates be set to 0. In the second, let one job have a release date of 2. This job is processed last. In a left activated schedule, the expected makespan is 1 and 3 respectively and the free slack will be 0 and 1 respectively for the two problem instances. Note both these schedules have the smallest possible expected makespan for their respective problem instances (i.e. they are both optimal). Thus a difference in free slack does not mean one schedule is better than the other.

This leads to some difficulty when designing experiments. On the one hand a catch-all approach is desired, so many problem instances should be compared. On the other hand, results on different problem instances may not be directly comparable.

We adopt a practical approach. We select several test problem instances. For each of these, we attempt to determine for each of the five performance measures presented in Section 3.1 the rank based discriminatory ability of the robustness measure, by determining the Spearman correlation. We determine this based on a set of schedules including 'good' and 'bad' schedules (presented in 4.2.3).

In order to be able to compare our work, we use the same problem instances used in [PAH]. These are titled nJ-rR-mM where n is the number of jobs, r the number of relations and m the number of machines. For each job, the processing time is a natural number between 1 and 20 and the release date is a natural number between 0 and $\lfloor n/2 \rfloor$. All released jobs are assumed to be able to start as soon as all predecessors are completed. That is, we have 0-lag finish start precedence relations. Precedence relations are semi-randomly selected, such that no cycles occur. Furthermore, release dates form a partial order in the precedence graph. That is, if job j is a successor of job i, then $r_j > r_i$.

As an example, a graphical representation of 30J-75R-8M is shown in figure 4.1. It is worth noting that the maximum depth of 30J-75R-8M is 5, which is more than n/m. As the maximum depth increases, the maximum depth path is more likely to determine the makespan. Therefore, solutions become more likely to be interchangeable. This makes local search difficult in such cases. However, in this case, processing times along the maximum depth path are short.

In addition to these problem instances, we will study the effect of inter machine dependencies in artificial problem instances that are chosen such that any left active schedule has no slack. These are discussed in Chapter 5.

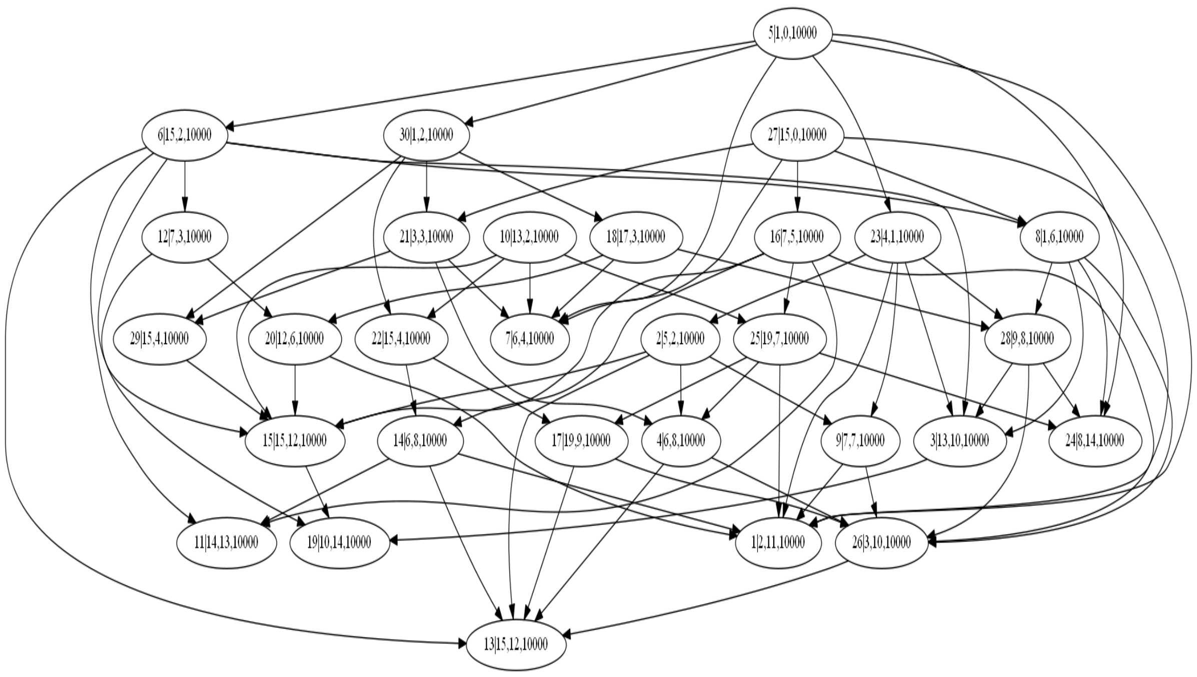


Figure 4.1: The precedence graph in the 30J-75R-8M data file. Each node represents a job j and gives its processing time, release date and due date as: $j|p_j, r_j, d_j$. Note all due dates are set so high that they do not influence the problem. This is the case for all problem instances we consider.

4.2.3 Schedule Generation

As discussed, we require a range of schedules to test our robustness measures on. Ideally, these would include schedules with both high and low values, for each of the five performance measures. However, generating such schedules is not trivial. Indeed, the main goal of this work is to determine how to create schedules with good performance measures. If we already knew how to create schedules with high performance measures, this work need not exist.

Again we adopt a pragmatic approach. To create schedules to test robustness measures on, we assume that there is no uncertainty: all jobs take exactly their mean processing time to complete. Thus the problem is deterministic. The local search procedure then attempts to minimize the makespan. We call this approach *deterministic makespan minimization*.

In fact, from the results presented in Section 4.3 it appears that only two out of five objectives do in fact correlate with the deterministic makespan.

Schedules are generated using a multi-start local search (MLS)[1] approach. MLS should effectively perform a random walk in the space of local optima. In brief, the MLS approach is as follows:

1. Create a random, feasible schedule using an assignment heuristic (algorithm 3).
2. Improve it until no further local improvements are possible using a variable neighborhood hill climb approach (algorithm 4).
3. Repeat this 1000 times and remember the best 100 found solutions.

We will now discuss each of these steps in more detail.

Assignment Heuristic

Three assignment heuristics were tested: Random Assignment (RA), Greedy Load Balancing (GLB) and Round Robin Assignment (RR). In each approach the lowest indexed job without unassigned predecessors is selected and then assigned. The indexation of jobs is arbitrary but fixed for each problem instance. This means that the order we assign the jobs in is an arbitrary (but not random) total ordering of the jobs. By assigning jobs in this order, we guarantee a feasible initial solution.

[1]see algorithm 5

The way the job gets assigned differs for each of the three heuristics. In Random Assignment, this assignment is random. In GLB, the job is assigned to the machine with the smallest load. In RR assignment, jobs are assigned to machines in turn. The first job to machine 0, the second to machine 1 etc. I.e. for m machines, the nth job is assigned to machine $n - 1 \mod m$. Note that RA is random, whereas GLB and RR are deterministic.

The assignment heuristic is also presented as an algorithm below:

Algorithm 3: Random Assignment

```
Data: A problem instance
Result: A feasible assignment of jobs to machines
1 while Unassigned jobs exist do
2     Select the lowest indexed job without unassigned predecessors;
3     Assign it to a random machine;
4     Update list of unassigned jobs;
5 end
```

In practice the choice of assignment heuristic had little effect on the quality of the solutions. All results presented use the random machine assignment heuristic (algorithm 3). The other algorithms are presented in the appendix.

Fitness Function

Local search approaches attempt to optimize some fitness function. To create schedules to test RMs on, we assume that there is no uncertainty: all jobs take exactly their mean processing time to complete. Thus the problem is deterministic. The local search procedure then attempts to minimize the makespan. We call this approach deterministic makespan minimization.

Neighborhood operators

Two neighborhood operators are used, neighborhood swaps and machine reassignments.

Given a schedule S, the neighborhood swap set $\mathrm{NO}_{\mathrm{ns}}(S)$ is the set of schedules resulting from any single feasible swap of jobs that are immediately adjacent on the same machine. Formally, for a job j, let $\mu(j, S)$ be the machine j is assigned to in S and $\iota(j, S)$ be the position on that machine. The schedule resulting from swapping job j' with its machine predecessor (if that exists) is $\mathrm{NO}_{\mathrm{ns}}(S, j')$[2]:

$$\begin{aligned}\mathrm{NO}_{\mathrm{ns}}(S, j') = \{S' | \iota(j', S') &= \iota(j'', S) = \iota(j', S) - 1 \\ &\wedge \iota(j'', S') = \iota(j', S) = \iota(j'', S) + 1 \\ &\wedge \forall j, \mu(j, S) = \mu(j, S') \\ &\wedge \forall j, j \notin \{j', j''\} \rightarrow \iota(j, S') = \iota(j, S) \\ &\wedge S' \text{ is a feasible schedule}\}\end{aligned}$$

Then $\mathrm{NO}_{\mathrm{ns}}(S) = \bigcup_{j' \in J} \mathrm{NO}_{\mathrm{ns}}(S, j')$.

Similarly, given a schedule S, the machine reassignment neighborhood $\mathrm{NO}_{\mathrm{mr}}(S, j')$ is the set of schedules resulting from reassigning a job j' to a different machine at a feasible position. $\mathrm{NO}_{\mathrm{mr}}(S)$ is the set of all schedules resulting from such a reassignment. Note that this is not a swapping of two jobs, but the removal and then reinsertion of a single job to a new position.

Both neighborhoods are explored in a semi random order: First a random job J_j is selected and the neighborhood operations involving this job are attempted. If no improvement exists, the next job is selected by simply taking the next job in the problem instance ($J_{j+1 \mod n}$). If no improvements at all exist for that neighborhood operator, the next neighborhood operator is attempted. We continue performing Hill Climb on alternating operators until a solution is found that forms a local optimum for both operators.

[2]Although this is always a set with at most one element, we chose this notation to be compatible with later neighborhood operators that may generate more than one schedule

As peusdocode, the variable neighborhood hill-climb algorithm is presented in algorithm 4.

Algorithm 4: Variable Neighborhood Hill Climb: VNHC(S). Returns a local optimum for both NS and MR.

```
Data: A set of jobs J, the neighborhood swap operator NS, the machine reassignment
      operator MR, a starting schedule S.
A local optimum for both NS and MR NSoptimum ← false;
MRoptimum ← false;
j <- random job from J;
while not (NSoptimum = true and MRoptimum = true) do
    if $NO_{ns}(S)$ contains an improvement then
        S' ← the first improvement from $\mathrm{NO}_{\mathrm{ns}}(S)$;
        S ← S';
        MRoptimum ← false ;
        j ← random job from J;
    else
        // A localy NS optimal solution has been found ;
        NSoptimum ← true;
        j <- random job from J;
        while MRoptimum = false do
            if $NO_{mr}(S)$ contains an improvement then
                S' ← the first improvement from $\mathrm{NO}_{\mathrm{mr}}(S)$;
                S ← S';
                j ← random job from J;
                NSoptimum ← false;
            else
                MRoptimum ← true;
            end
        end
    end
end
return S;
```

MLS approach

Now we are ready to present the entire MLS algorithm. In pseudocode, where S_{Nbest} is the ordered array of the Nbest best solutions found so far:

Algorithm 5: MLS$_{\text{Nbest}}$(PI,Ntries,AH,VNHC,OF): Return the best Nbest schedules out of Ntries local optima.

```
Data: A problem instance PI, a number of tries Ntries, a starting assignment heuristic
      AH, an objective function OF, a variable neighborhood hill climb VNHC
Result: The best local optimum schedule found after Ntries
trycount ← 0;
$S_{\text{Nbest}}$ ← AH(PI);
while trycount < Ntries do
    $S$ ← AH(PI);
    $S$ ← VNHC(S);
    if $S$ better than a solution in $S_{Nbest}$ then
        Insert S in the correct position;
        Shift all other solutions right by one position;
        Forget a solution if it gets shifted out of the list;
    end
    trycount ++ ;
end
return S*;
```

We use the first-improvement method as this allows for a less structured exploration of the search space.

4.2.4 Experiments performed

We start by mentioning again that the aim of this section is to discover which robustness measures from Section 3.2 are able to predict the five performance measures from Section 3.1. We attempt to determine this as follows.

We consider two job processing time distributions. Either jobs processing times are distributed normally (i.e. $p_j \backsim N(p_j, 0.3p_j)$ or job processing times are distributed exponentially with mean p_j (i.e. $p_j \backsim Exp(p_j)$). For each of the instances PI discussed in 4.2.2 we:

1. Apply $\text{MLS}_{100}(PI, 1000, RA, VNHC, \text{Deterministic Makespan Minimization})$ (algorithm 5). That is, we Generate 1000 schedules that are locally optimal with respect to the deterministic makespan and the neighborhood operators in Section 4.2.3. Of these we select the 100 best schedules (those with the shortest makespan). The assignment heuristic used is Random Machine Assignment (algorithm 3) and the hill climb is VNHC (algorithm 4).

2. Do the following once for each job processing time distribution used:

   (a) For each of the 100 schedules in MLS_{100}(PI, 1000, RA, VNHC, Deterministic Makespan Minimization):

      i. Calculate the robustness measures of the schedule.

      ii. Perform 300 simulations for each job processing time distribution.

      iii. For each simulation, we save the realized makespan, the sum of linear start delays and the percentage of on time jobs in that simulation.

      iv. Calculate the performance measures (equations 3.2 to 3.6) of the schedule.

3. Calculate the Spearmann correlation between all performance measure, robustness measure pairings. Note that as we have 100 schedules, each Spearmann correlation is based on 100 data points.

A robustness measure with an extreme (approaching 1 or -1) Spearman correlation with a performance measure may be useful to estimate the performance measure.

In total we perform 600 simulations on 1200 schedules (720,000 simulation in total). The results of these experiments are presented in the next section.

4.2.5 Definitions and notation.

Below we present an exhaustive list of all robustness measures considered in our experiments and their notation. We refer to Chapter 2 for an overview of relevant references in literature and Chapter 3 for a theoretical discussing of robustness measures. Let J denote the set of jobs to be performed. The slack property abbreviations are given in Table 3.1.

- Deterministic makespan ($C_{\max}^{\text{Det}}$): The makespan of a schedule under the assumption that all jobs take exactly their mean processing time to process. That is, let C_j be the completion time of job j, $\mathbf{p_j}$ the realized processing time of job j and p_j the mean processing time of job j. Then we have:

$$C_{\max}^{\mathrm{Det}} = \max_{j \in J} \{C_j | \mathbf{p_j} = p_j\}$$

- Normal approximation based makespan ($\mu_{C_{\max}}^{\mathrm{NA}}$): The result of Algorithm 1.
- Unweighted average of each of the slack properties from Table 3.1. To distinguish between the slack property and the slack based robustness measure, we drop the index.

  For example: FS_j is the free slack of job j as defined in Table 3.1. FS is the unweighted average of free slacks:

  $$\mathrm{FS} = \frac{1}{|J|} \sum_{j \in J} \mathrm{FS}_j$$

  The chosen value of γ in given as a superscript for binary and upperbound slacks. For example $\mathrm{BFS}^{0.3}$ is the unweighted sum of binary free slacks with $\gamma = 0.3$.
- Average of slack properties from Table 3.1 weighted by the number of successors. To indicate a weighting is used, we add the prefix 'w' for 'weighted' and subscript 'succ' for 'successor'.

  For example: $\mathrm{wFS}_{\mathrm{succ}}$ is the average of the free slacks weighted by number of successors. Let $|\sigma_j|$ be the number of successors (including the machine successor if it exists) of job j.

  $$\mathrm{wFS}_{\mathrm{succ}} = \frac{1}{|J|} \sum_{j \in J} \mathrm{FS}_j \cdot |\sigma_j|$$

4.3 Results

All robustness definition[3], robustness measure[4] combinations were measured. We have selected meaningful results to present in this section. We will discuss the results in 3 parts. First we discuss robustness measures for expected makespan, second RMs for solution robustness definitions and finally RMs for quality robustness definitions.

[3] see 3.1

[4] See end of previous section.

Experiments were performed on $N(p, 0.3p)$ and $Exp(p)$ distributions. The graphs presented in this section are all from the $N(p, 0.3p)$ runs. $Exp(p)$ runs give much the same images but with larger variations.

4.3.1 Expected Makespan

$C_{\max}^{\text{Det}}$ and the $\mu_{C_{\max}}^{\text{NA}}$ show strong rank correlation with $\mu_{\hat{C}_{\max}}$, as can be seen in tables 4.1 and 4.2. $\mu_{C_{\max}}^{\text{NA}}$ shows the best correlation for both the $N(p, 0.3p)$ and the $Exp(p)$ job processing times. It outperforms all other measures in every instance. FS also shows some correlation. For both FS and TS weighting by number of successors does not increase correlation. All RMs not presented in this table ($\text{BFS}^{0.3}$, $\text{UFS}^{0.3}$, $\text{BTS}^{0.3}$, $\text{UTS}^{0.3}$, $\text{wBFS}_{\text{succ}}^{0.3}$, $\text{wUFS}_{\text{succ}}^{0.3}$, $\text{wBTS}_{\text{succ}}^{0.3}$, $\text{wUTS}_{\text{succ}}^{0.3}$) had a small rank based correlation between -0.25 and 0.25.

It is not clear why the correlation of TS is lower than that of FS. One explanation may be that as a measure total slack is overly sensitive to the number of jobs on the same machine: If free slack exists on some non critical subpath, then this is total slack for all jobs along that subpath. This means that there are multipliers involved that are not involved in the free slack case.

Another possibility is that the higher FS correlation is due to there being large differences in machine load. In this case, there is some correlation between FS and $C_{\max}^{\text{Det}}$, which is in turn correlated with $\mu_{\hat{C}_{\max}}$. Giving an upperbound to FS_j removes this artefact. This corresponds to the drop in rank based correlation between $\text{BTS}^{0.3}$ and $\mu_{\hat{C}_{\max}}$ to 0.04 on average.

We think it unlikely that the difference between FS and TS is due to random fluctuation because for every problem instance the correlation of FS is greater than that of TS.

Figures 4.2, 4.3 show that $C_{\max}^{\text{Det}}$ and $\mu_{C_{\max}}^{\text{NA}}$ are highly correlated with the realized makespan, both given a problem instance and independent of problem instance. However, it should be noted that the schedules with highest and lowest $\mu_{\hat{C}_{\max}}$ are within one standard deviation for most problem instances. These figures also show that both measures underestimate the realized makespan by a small margin when the job processing times are $N(p, 0.3p)$.

From Figure 4.4 it is hard to see that FS has some correlation with $\mu_{\hat{C}_{\max}}$. The observed values of $\mu_{\hat{C}_{\max}}$ are generally within one standard error of each other for any given problem instance.

Problem Instance	$C_{\max}^{\text{Det}}$	$\mu_{C_{\max}}^{\text{NA}}$	FS	wFS$_{\text{succ}}$	TS	wTS$_{\text{succ}}$	SDR
30j-15r-4m.ms	0.93	0.95	0.52	0.53	0.42	0.40	0.43
30j-15r-8m.ms	0.88	0.93	0.28	0.32	0.20	0.19	0.14
30j-30r-4m.ms	0.93	0.97	0.51	0.46	0.28	0.28	0.28
30j-30r-8m.ms	0.85	0.93	0.32	0.30	0.23	0.22	0.17
30j-75r-4m.ms	0.93	0.97	0.18	0.25	0.03	0.03	0.12
30j-75r-4m.ms	0.93	0.97	0.18	0.25	0.03	0.03	0.12
100j-50r-6m.ms	0.88	0.91	0.48	0.46	0.26	0.27	0.25
100j-50r-12m.ms	0.85	0.92	0.53	0.32	0.38	0.31	0.38
100j-100r-6m.ms	0.91	0.96	0.67	0.66	0.46	0.41	0.48
100j-100r-12m.ms	0.96	0.97	0.49	0.26	0.30	0.25	0.33
100j-250r-6m.ms	0.94	0.97	0.53	0.50	0.18	0.18	0.20
100j-250r-12m.ms	0.92	0.95	0.34	0.29	0.08	0.09	0.10
Mean	0.91	0.95	0.42	0.38	0.24	0.22	0.25

Table 4.1: Spearman Correlation between $\mu_{\hat{C_{\max}}}$ and deterministic makespan, normal approximation based mean makespan, mean Free Slack, mean weighted Free Slack, mean Total Slack, mean weighted Total Slack, Total Slack duration ratio. Using $N(p, 0.3p)$ jobs, 100 Schedules and 300 simulations per schedule.

Problem Instance	$C_{\max}^{\text{Det}}$	$\mu_{C_{\max}}^{\text{NA}}$	FS	wFS$_{\text{succ}}$	TS	wTS$_{\text{succ}}$	SDR
30j-15r-4m.ms	0.58	0.62	0.21	0.33	0.06	0.03	0.19
30j-15r-8m.ms	0.58	0.63	0.05	0.13	-0.18	-0.14	-0.14
30j-30r-4m.ms	0.60	0.67	0.21	0.26	-0.13	-0.11	-0.10
30j-30r-8m.ms	0.58	0.67	0.16	0.18	-0.01	0.01	-0.00
30j-75r-4m.ms	0.75	0.81	0.00	0.11	-0.21	-0.19	-0.09
30j-75r-4m.ms	0.75	0.81	0.00	0.11	-0.21	-0.19	-0.09
100j-50r-6m.ms	0.60	0.64	0.36	0.37	-0.02	-0.02	0.05
100j-50r-12m.ms	0.44	0.53	0.24	0.18	-0.07	-0.10	0.06
100j-100r-6m.ms	0.54	0.63	0.36	0.49	-0.02	-0.06	0.05
100j-100r-12m.ms	0.66	0.70	0.22	0.06	-0.15	-0.16	-0.07
100j-250r-6m.ms	0.71	0.75	0.36	0.40	-0.09	-0.06	-0.01
100j-250r-12m.ms	0.49	0.56	0.01	0.24	-0.42	-0.35	-0.27
Mean	0.61	0.67	0.18	0.24	-0.12	-0.11	-0.04

Table 4.2: Spearman Correlation between $\mu_{\hat{C_{\max}}}$ and deterministic makespan, normal approximation based mean makespan, mean Free Slack, mean weighted Free Slack, mean Total Slack, mean weighted Total Slack, Total Slack duration ratio. Using $Exp(p)$ jobs, 100 Schedules and 300 simulations per schedule.

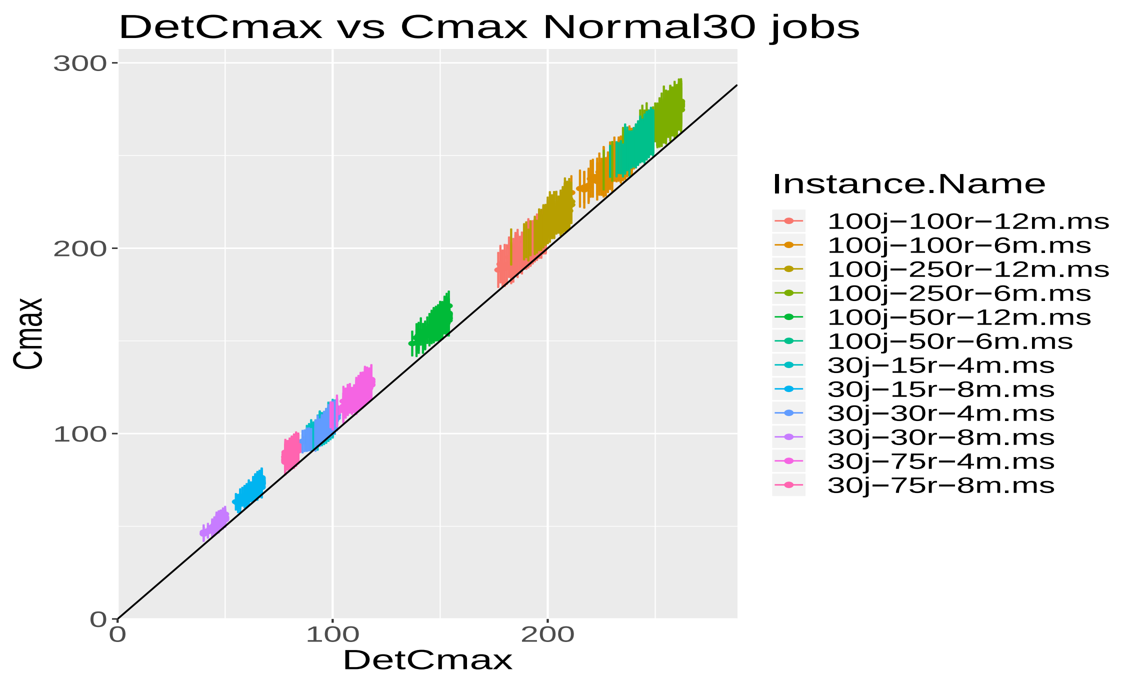


Figure 4.2: $C_{\max}^{\text{Det}}$ vs simulated makespan. The black line is the $x = y$ line. Vertical bars are the standard deviation of mean. $N(p, 0.3p)$ jobs. Top 100 MLS schedules. 300 Simulation runs per schedule.

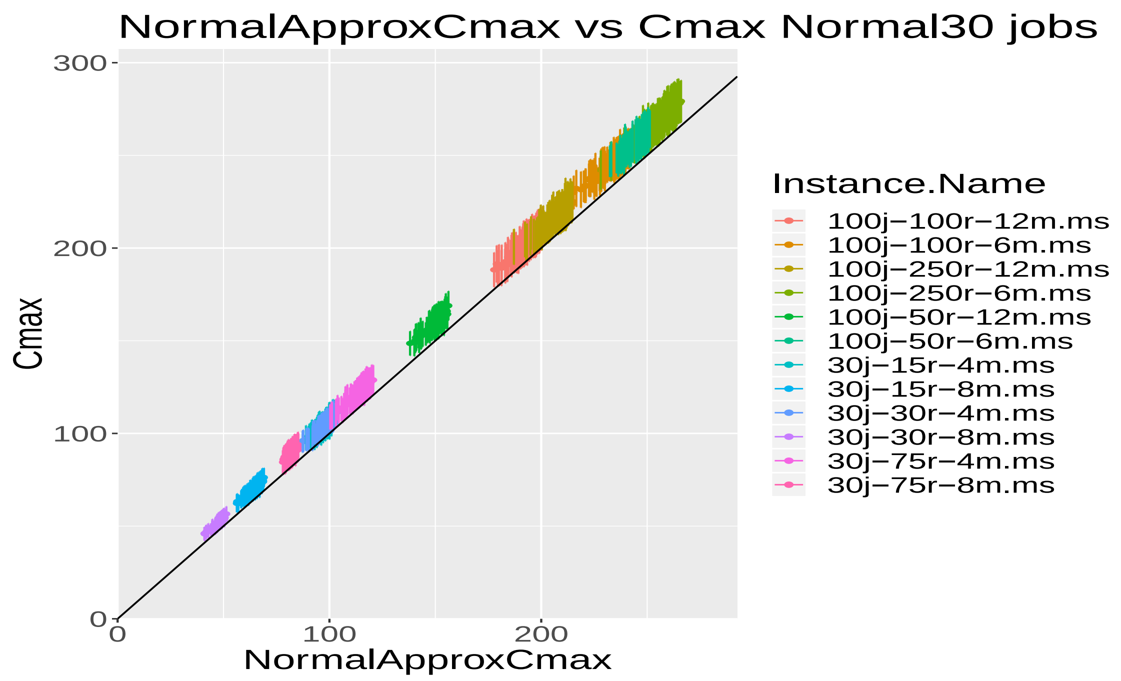


Figure 4.3: $\mu_{C_{\max}}^{\text{NA}}$ vs simulated makespan. The black line is the $x = y$ line. Vertical bars are the standard deviation of mean. $N(p, 0.3p)$ jobs. Top 100 MLS schedules. 300 Simulation runs per schedule.

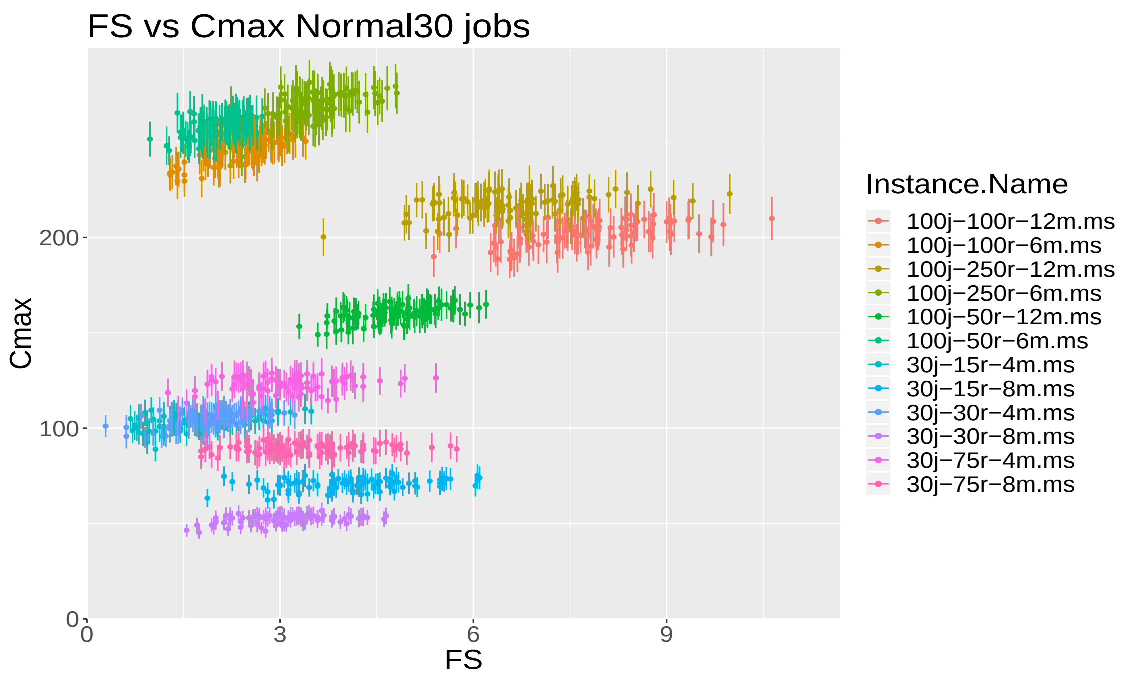


Figure 4.4: FS vs simulated makespan. Vertical bars are the standard deviation of mean. $N(p, 0.3p)$ jobs. Top 100 MLS schedules. 300 Simulation runs per schedule.

4.3.2 Solution Robustness

This section compares measures to estimate solution robustness definitions. The definitions considered are the percentage of jobs that start on time SP and the sum of linear start delays LSD. See Section 3.1 for the quantitative definition.

Linear Start Delay

Tables 4.3 and 4.4 show the Spearman correlations between the average linear start delay and the robustness measures for $N(p, 0.30p)$ and $Exp(p)$ jobs respectively. They show that there is no strong correlation between any robustness measure and LSD. The best measure would appear to be wTS$_{\text{succ}}$ with a mean correlation of -0.33 in the Normal and -0.41 in the Exponential case. As before, all omited measures (i.e. $\mu^{\text{NA}}_{C_{\max}}$, wFS$_{\text{succ}}$, wBFS$^{0.3}_{\text{succ}}$, wUFS$^{0.3}_{\text{succ}}$, BTS$^{0.3}_{,}UTS^{0.3}$) have values between -0.25 and 0.25.

Problem Instance	$C^{\text{Det}}_{\max}$	FS	BFS$^{0.3}$	UFS$^{0.3}$	TS	wTS$_{\text{succ}}$	SDR
30j-15r-4m.ms	0.04	0.05	-0.10	-0.07	-0.19	-0.25	-0.07
30j-15r-8m.ms	0.31	-0.21	-0.27	-0.12	-0.33	-0.33	-0.26
30j-30r-4m.ms	0.10	-0.21	-0.20	-0.20	-0.35	-0.35	-0.37
30j-30r-8m.ms	0.30	0.06	-0.22	-0.29	-0.14	-0.18	-0.22
30j-75r-4m.ms	0.22	-0.19	-0.36	-0.46	-0.54	-0.56	-0.34
30j-75r-4m.ms	0.22	-0.19	-0.36	-0.46	-0.54	-0.56	-0.34
100j-50r-6m.ms	0.20	-0.08	-0.14	-0.26	-0.17	-0.20	-0.13
100j-50r-12m.ms	0.11	-0.07	-0.09	-0.25	-0.18	-0.25	-0.11
100j-100r-6m.ms	0.00	-0.12	0.07	0.08	-0.20	-0.21	-0.17
100j-100r-12m.ms	0.05	-0.06	-0.02	-0.22	-0.37	-0.36	-0.29
100j-250r-6m.ms	0.08	-0.05	-0.26	-0.20	-0.27	-0.26	-0.21
100j-250r-12m.ms	0.10	-0.08	-0.16	-0.12	-0.41	-0.42	-0.31
Mean	0.14	-0.10	-0.18	-0.21	-0.31	-0.33	-0.23

Table 4.3: Spearman Correlation between LSD and deterministic makespan, mean Free Slack, mean Binary Free Slack (0,30 cutoff), mean Upperbound Free Slack (0,30 cutoff), mean Total Slack, mean weighted Total Slack, Total Slack duration ratio. Using $N(p, 0.3p)$ jobs, 100 Schedules and 300 simulations per schedule.

Figures 4.6 and 4.5) show that the variation on LSD is large. This may mean that the observed correlation is only statistical fluctuation.

Problem Instance	$C_{\max}^{\text{Det}}$	FS	BFS$^{0.3}$	UFS$^{0.3}$	TS	wTS$_{\text{succ}}$	SDR
30j-15r-4m.ms	-0.08	-0.13	-0.14	-0.09	-0.30	-0.35	-0.18
30j-15r-8m.ms	0.22	-0.26	-0.28	-0.09	-0.37	-0.40	-0.33
30j-30r-4m.ms	-0.12	-0.35	-0.11	-0.16	-0.46	-0.45	-0.45
30j-30r-8m.ms	0.26	-0.00	-0.10	-0.11	-0.12	-0.12	-0.24
30j-75r-4m.ms	0.17	-0.34	-0.47	-0.47	-0.63	-0.63	-0.44
30j-75r-4m.ms	0.17	-0.34	-0.47	-0.47	-0.63	-0.63	-0.44
100j-50r-6m.ms	0.08	-0.14	-0.02	-0.12	-0.19	-0.20	-0.14
100j-50r-12m.ms	0.04	-0.06	0.01	-0.06	-0.27	-0.32	-0.23
100j-100r-6m.ms	-0.14	-0.30	0.07	0.01	-0.40	-0.45	-0.36
100j-100r-12m.ms	0.03	-0.02	0.04	-0.04	-0.38	-0.42	-0.31
100j-250r-6m.ms	-0.10	-0.09	-0.11	0.04	-0.32	-0.36	-0.30
100j-250r-12m.ms	-0.06	-0.29	-0.33	-0.26	-0.57	-0.58	-0.45
Mean	0.04	-0.19	-0.16	-0.15	-0.39	-0.41	-0.32

Table 4.4: Spearman Correlation between LSD and deterministic makespan, mean Free Slack, mean Binary Free Slack (0,30 cutoff), mean Upperbound Free Slack (0,30 cutoff), mean Total Slack, mean weighted Total Slack, Total Slack duration ratio. Using $Exp(p)$ jobs, 100 Schedules and 300 simulations per schedule.

However, the fact that wTS$_{\text{succ}}$ and TS perform similarly, and both perform reasonably in both the $N(p, 0.3p)$ and $Exp(p)$ case indicates that there is some pattern here. Another reason to suspect a pattern is that the results make sense: As slack decreases, one expects delays to increase. Since the Exponential case has larger disturbances, slack is more important. Thus the correlation is larger in the $Exp(p)$ case.

Nevertheless, it is clear that LSD varies greatly in each simulation run. So it is not surprising that any robustness measure struggles to predict it well.

Finally, Figure **??** highlights a problem with BFS$^{0.3}$ an BTS$^{0.3}$. In both these measures, many schedules have the same robustness measure value. This makes it impossible to distinguish between schedules.

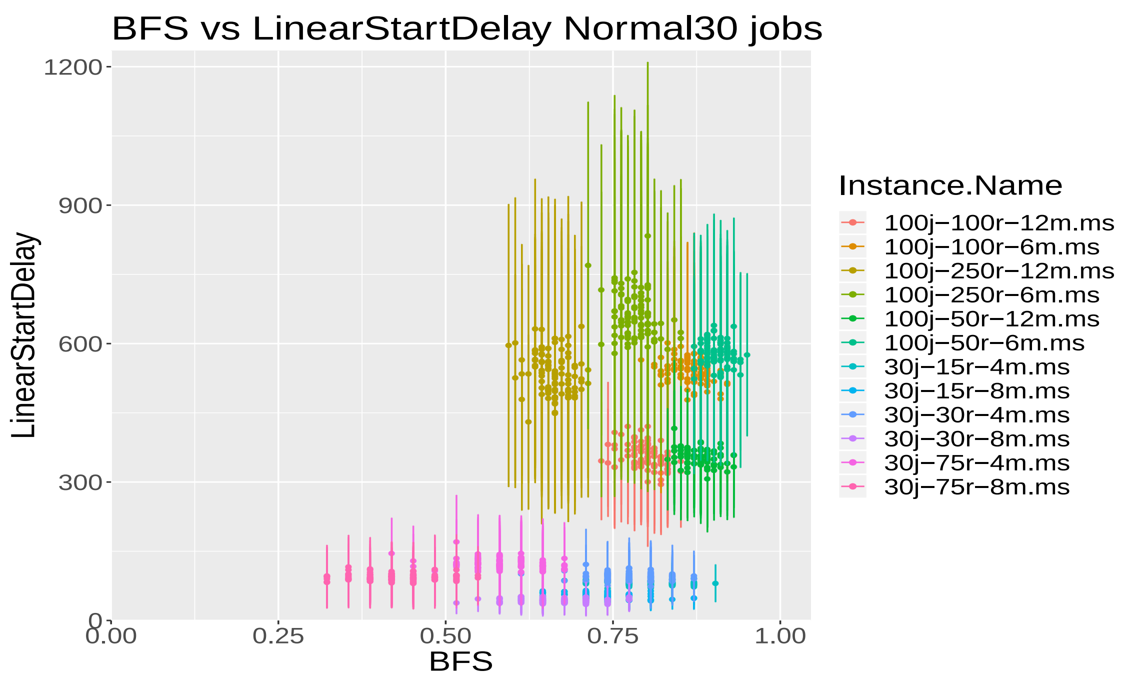


Figure 4.5: Binary free slack (0.30 cutoff) vs average start delay in schedule. $N(p, 0.3p)$ jobs. 300 Simulation runs per schedule.

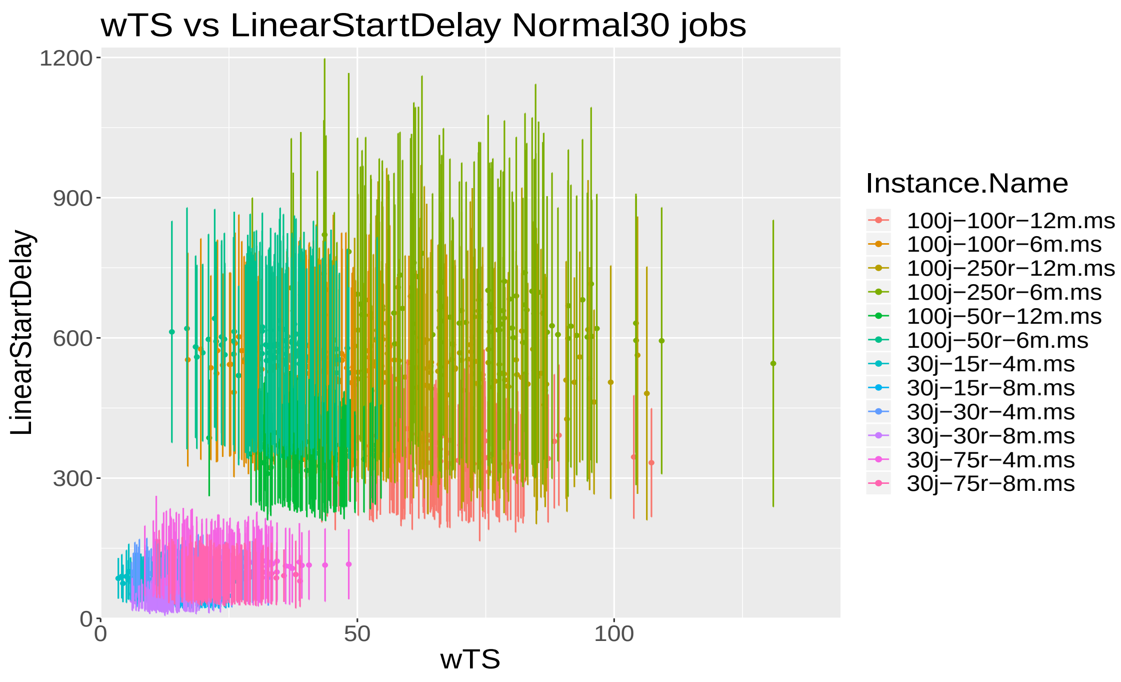


Figure 4.6: $\mathrm{wTS_{succ}}$ vs LSD. $N(p, 0.3p)$ jobs. 300 Simulation runs per schedule. Vertical bars are the standard deviation of the LSD for each schedule.

Start Punctuality

Tables 4.5 and 4.6 show the Spearman correlations between the average percentage of on time jobs and the robustness measures for $N(p, 0.30p)$ and $Exp(p)$ jobs respectively. As with the case of LSD, SP appears to be a difficult measure to predict. The best measure now appears to be $\text{BFS}^{0.3}$, with a correlation of 0.41 for the Normal and 0.33 for the Exponential case. Both the free slack and total slack based measures presented perform around the 0.3 region. That is, no measure predicts particularly well. Nevertheless, that may be an improvement over using $C_{\max}^{\text{Det}}$.

The low correlation values are unsurprising given the large variance in SP. Figure 4.7 shows the large variance in SP and the lack of distinct values of $\text{BFS}^{0.3}$ for the generated schedules.

Problem Instance	$C_{\max}^{\text{Det}}$	FS	$\text{BFS}^{0.3}$	wFS_{succ}	TS	wTS_{succ}	SDR
30j-15r-4m.ms	-0.08	-0.13	0.48	-0.06	0.07	0.12	0.09
30j-15r-8m.ms	-0.17	0.36	0.40	0.36	0.39	0.40	0.44
30j-30r-4m.ms	-0.09	0.14	0.47	0.31	0.23	0.26	0.29
30j-30r-8m.ms	-0.17	0.30	0.55	0.37	0.28	0.30	0.32
30j-75r-4m.ms	-0.10	0.33	0.50	0.44	0.62	0.65	0.48
30j-75r-4m.ms	-0.10	0.33	0.50	0.44	0.62	0.65	0.48
100j-50r-6m.ms	-0.09	0.28	0.47	0.28	0.12	0.14	0.14
100j-50r-12m.ms	0.04	0.18	0.32	0.30	0.14	0.17	0.15
100j-100r-6m.ms	0.07	0.24	0.22	0.31	0.07	0.10	0.11
100j-100r-12m.ms	0.04	0.29	0.48	0.42	0.27	0.26	0.32
100j-250r-6m.ms	-0.04	0.21	0.30	0.07	0.24	0.19	0.30
100j-250r-12m.ms	-0.08	0.17	0.27	0.25	0.32	0.35	0.31
Mean	-0.06	0.22	0.41	0.29	0.28	0.30	0.29

Table 4.5: Spearman Correlation between SP and deterministic makespan, mean Free Slack, mean Binary Free Slack (0,30 cutoff), mean weighted Free Slack, mean Total Slack, mean weighted Total Slack, Total Slack duration ratio. Using $N(p, 0.3p)$ jobs, 100 Schedules and 300 simulations per schedule.

Problem Instance	$C_{\max}^{\text{Det}}$	FS	BFS$^{0.3}$	wFS$_{\text{succ}}$	TS	wTS$_{\text{succ}}$	SDR
30j-15r-4m.ms	0.08	0.13	0.37	0.11	0.30	0.30	0.30
30j-15r-8m.ms	-0.15	0.39	0.38	0.41	0.41	0.43	0.48
30j-30r-4m.ms	0.07	0.32	0.32	0.33	0.45	0.46	0.47
30j-30r-8m.ms	-0.05	0.41	0.50	0.46	0.30	0.30	0.35
30j-75r-4m.ms	-0.14	0.37	0.56	0.48	0.57	0.58	0.43
30j-75r-4m.ms	-0.14	0.37	0.56	0.48	0.57	0.58	0.43
100j-50r-6m.ms	0.04	0.26	0.22	0.23	0.06	0.05	0.06
100j-50r-12m.ms	-0.02	0.20	0.23	0.39	0.16	0.22	0.20
100j-100r-6m.ms	0.13	0.30	0.03	0.38	0.23	0.26	0.22
100j-100r-12m.ms	0.02	0.14	0.29	0.36	0.26	0.30	0.30
100j-250r-6m.ms	0.10	0.20	0.13	0.16	0.26	0.26	0.34
100j-250r-12m.ms	0.05	0.26	0.33	0.35	0.36	0.38	0.34
Mean	-0.00	0.28	0.33	0.34	0.33	0.34	0.33

Table 4.6: Spearman Correlation between SP and deterministic makespan, mean Free Slack, mean Binary Free Slack (0,30 cutoff), mean weighted Free Slack, mean Total Slack, mean weighted Total Slack, Total Slack duration ratio. Using $Exp(p)$ jobs, 100 Schedules and 300 simulations per schedule.

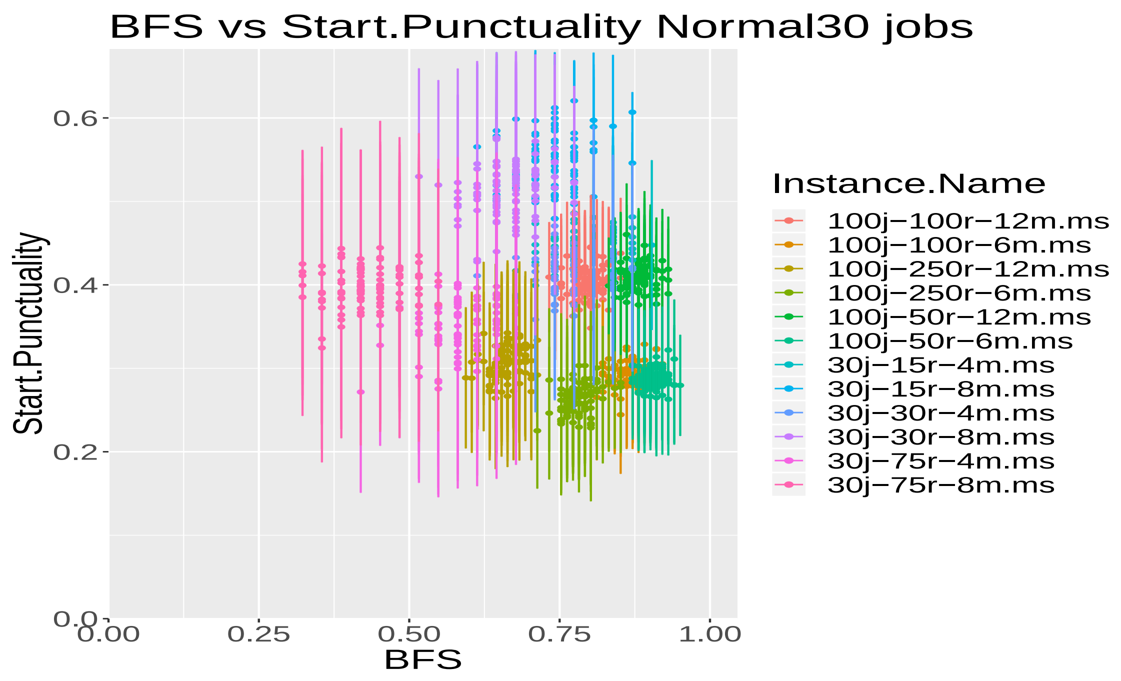


Figure 4.7: Binary free slack vs start punctuality. $N(p, 0.3p)$ jobs. 300 Simulation runs per schedule.

4.3.3 Quality Robustness

In this section theh quality robustness measures, i.e. the 95th percentile ($\mathcal{C}^{0.95}$) and the standard deviation over mean (VarCo), are used. See Section 3.1 for more information on $\mathcal{C}^{0.95}$ and VarCo. First we discuss the $\mathcal{C}^{0.95}$, then VarCo.

95th percentile makespan

$C_{\max}^{\mathrm{Det}}$ and $\mu_{C_{\max}}^{\mathrm{NA}}$ both show strong correlation with $\mathcal{C}^{0.95}$. $\mu_{C_{\max}}^{\mathrm{NA}}$ is most highly correlated (see tables 4.7 and 4.8). In the same tables, we see a much lower correlation for slack based measures. The quality of all robustness measures is worse when using $Exp(p)$ jobs. The unreported robustness measures ($\mathrm{BTS}^{0.3}_{,}\mathrm{UTS}^{0.3}_{,}\mathrm{wTS}_{\mathrm{succ}}, \mathrm{BFS}^{0.3}, \mathrm{UFS}^{0.3}$) have rank based correlation coefficients between -0.25 and 0.25 with $\mathcal{C}^{0.95}$. In the Normal distribution case, FS performs better than TS for every problem instance.

Problem Instance	$C_{\max}^{\mathrm{Det}}$	$\mu_{C_{\max}}^{\mathrm{NA}}$	TS	SDR	FS	$\mathrm{wFS}_{\mathrm{succ}}$
30j-15r-4m.ms	0.89	0.91	0.44	0.46	0.51	0.51
30j-15r-8m.ms	0.83	0.87	0.22	0.15	0.27	0.28
30j-30r-4m.ms	0.88	0.92	0.30	0.30	0.46	0.39
30j-30r-8m.ms	0.79	0.84	0.30	0.18	0.37	0.36
30j-75r-4m.ms	0.90	0.94	0.03	0.12	0.21	0.24
30j-75r-4m.ms	0.90	0.94	0.03	0.12	0.21	0.24
100j-50r-6m.ms	0.85	0.87	0.27	0.26	0.45	0.42
100j-50r-12m.ms	0.83	0.85	0.41	0.42	0.48	0.26
100j-100r-6m.ms	0.90	0.93	0.50	0.51	0.69	0.66
100j-100r-12m.ms	0.93	0.94	0.28	0.31	0.53	0.29
100j-250r-6m.ms	0.93	0.94	0.23	0.23	0.56	0.52
100j-250r-12m.ms	0.89	0.93	0.09	0.09	0.34	0.29
Mean	0.88	0.91	0.26	0.26	0.42	0.37

Table 4.7: Spearman Correlation between $\mathcal{C}^{0.95}$ and deterministic makespan, normal approximation based mean makespan, mean Total Slack, Total Slack duration ratio, mean Free Slack, mean weighted Free Slack. Using $N(p, 0.3p)$ jobs, 100 Schedules and 300 simulations per schedule.

Figure 4.8 shows the high correlation between $\mathcal{C}^{0.95}$ and $C_{\max}^{\mathrm{Det}}$ in the $N(p, 0.3p)$ case. The figure for $\mu_{C_{\max}}^{\mathrm{NA}}$ looks very similar.

Figure 4.9 shows that in the Exponential case the values of $\mathcal{C}^{0.95}$ are more scattered.

Problem Instance	$C_{\max}^{\text{Det}}$	$\mu_{C_{\max}}^{\text{NA}}$	TS	SDR	FS	wFS_{succ}
30j-15r-4m.ms	0.34	0.40	0.08	0.19	0.12	0.12
30j-15r-8m.ms	0.38	0.43	-0.15	-0.06	0.02	0.09
30j-30r-4m.ms	0.37	0.44	-0.18	-0.14	0.12	0.22
30j-30r-8m.ms	0.50	0.53	0.13	0.08	0.14	0.13
30j-75r-4m.ms	0.45	0.52	-0.25	-0.13	-0.03	-0.05
30j-75r-4m.ms	0.45	0.52	-0.25	-0.13	-0.03	-0.05
100j-50r-6m.ms	0.39	0.42	0.05	0.12	0.26	0.29
100j-50r-12m.ms	0.32	0.38	-0.02	0.00	0.19	0.11
100j-100r-6m.ms	0.35	0.41	-0.04	0.03	0.23	0.30
100j-100r-12m.ms	0.51	0.55	-0.06	-0.02	0.15	-0.09
100j-250r-6m.ms	0.48	0.50	-0.04	0.01	0.42	0.35
100j-250r-12m.ms	0.39	0.44	-0.30	-0.19	0.01	0.24
Mean	0.41	0.46	-0.09	-0.02	0.13	0.14

Table 4.8: Spearman Correlation between $\mathcal{C}^{0.95}$ and deterministic makespan, normal approximation based mean makespan, mean Total Slack, Total Slack duration ratio, mean Free Slack, mean weighted Free Slack. Using *Exp*(p) jobs, 100 Schedules and 300 simulations per schedule.

There are no error bars on these figures because to provide these would require repeated sets of simulations.

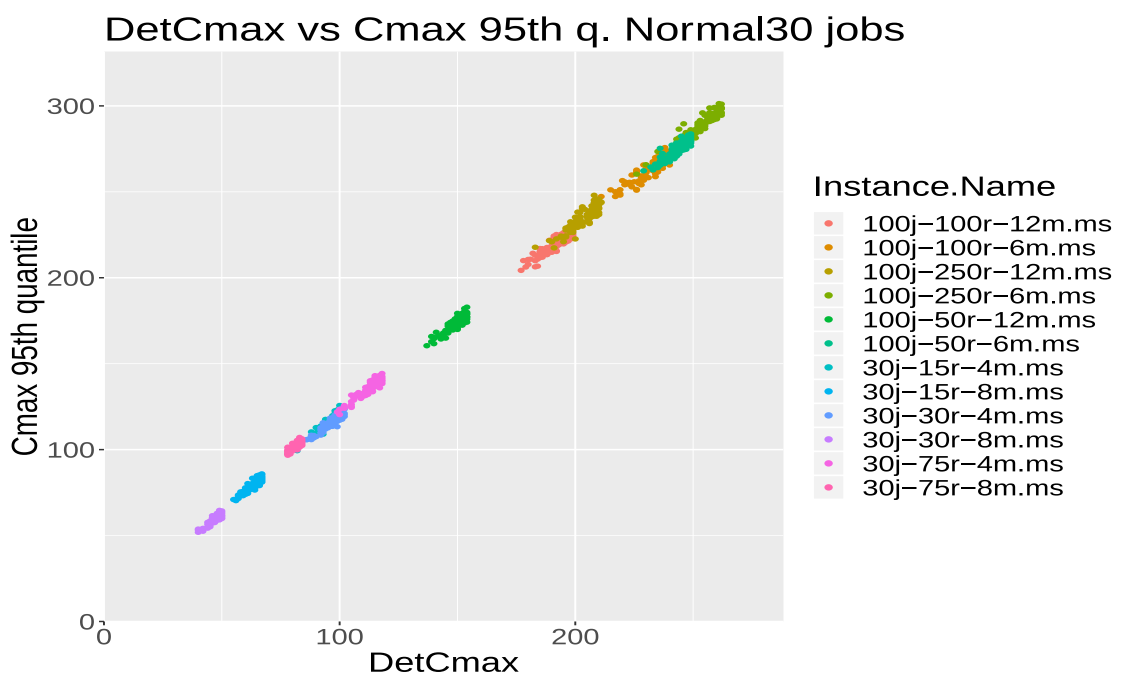


Figure 4.8: $C_{\max}^{\text{Det}}$ makespan vs $\mathcal{C}^{0.95}$. $N(p, 0.3p)$ jobs. 300 Simulation runs per schedule.

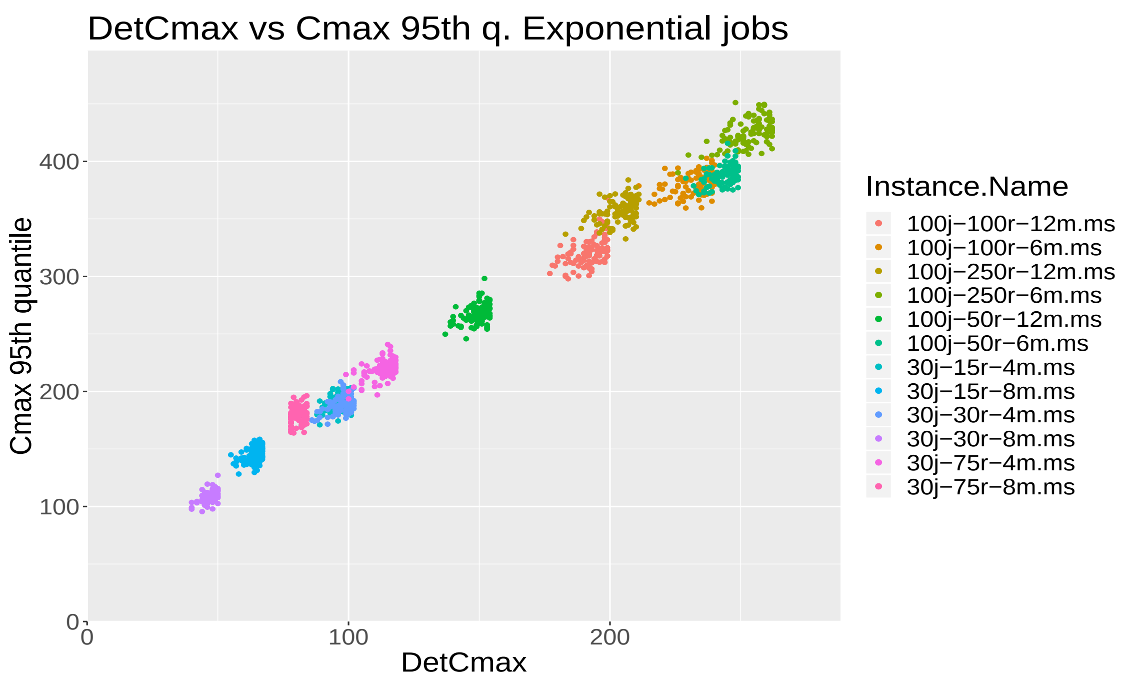


Figure 4.9: $C_{\max}^{\text{Det}}$ makespan vs $\mathcal{C}^{0.95}$. $Exp(p)$ jobs. 300 Simulation runs per schedule.

Coefficient of Variation

As we can see from Tables 4.9 and 4.10 none of the robustness measures are good predictors for the coefficient of variation of the makespan. With the sole execption of $\mathrm{wFS}_{\mathrm{succ}}$ in the case of $N(p, 0.3p)$ jobs every robustness measure (including $\mathrm{BTS}^{0.3}_{,}\mathrm{UTS}^{0.3}_{,}\mathrm{wTS}_{\mathrm{succ}}$, $\mathrm{wBTS}^{0.3}_{\mathrm{succ},\mathrm{wUTS}^{0.3}_{\mathrm{succ}}}$ and SDR not shown in the table) has a correlation of between -0.25 and 0.25.

Figure 4.10 illustrates the lack of correlation. No error bars are included because doing so would require multiple repeated sets of simulation runs.

Problem Instance	$C_{\max}^{\mathrm{Det}}$	$\mu^{\mathrm{NA}}_{C_{\max}}$	FS	$\mathrm{BFS}^{0.3}$	$\mathrm{UFS}^{0.3}$	$\mathrm{wFS}_{\mathrm{succ}}$	TS
30j-15r-4m.ms	-0.15	-0.15	-0.11	0.10	0.03	0.51	-0.05
30j-15r-8m.ms	-0.14	-0.17	-0.08	-0.16	-0.09	0.28	-0.00
30j-30r-4m.ms	-0.08	-0.08	-0.13	0.18	0.16	0.39	0.02
30j-30r-8m.ms	0.06	0.05	0.09	-0.01	-0.06	0.36	0.08
30j-75r-4m.ms	-0.25	-0.21	-0.04	-0.02	-0.09	0.24	-0.17
30j-75r-4m.ms	-0.25	-0.21	-0.04	-0.02	-0.09	0.24	-0.17
100j-50r-6m.ms	0.12	0.11	-0.03	0.19	-0.01	0.42	-0.04
100j-50r-12m.ms	0.03	-0.07	0.01	0.20	0.05	0.26	0.15
100j-100r-6m.ms	0.02	0.01	0.02	0.33	0.22	0.66	-0.03
100j-100r-12m.ms	-0.06	-0.04	0.01	0.19	0.07	0.29	-0.17
100j-250r-6m.ms	-0.08	-0.12	0.03	0.18	-0.01	0.52	-0.08
100j-250r-12m.ms	-0.34	-0.27	-0.17	0.18	0.20	0.29	-0.41
Mean	-0.09	-0.10	-0.04	0.11	0.03	0.37	-0.07

Table 4.9: Spearman Correlation between VarCo and deterministic makespan, normal approximation based mean makespan, mean Free Slack, mean Binary Free Slack (0,30 cutoff), mean Upperbound Free Slack (0,30 cutoff), mean weighted Free Slack, mean Total Slack. Using $N(p, 0.3p)$ jobs, 100 Schedules and 300 simulations per schedule.

Problem Instance	$C_{\max}^{\mathrm{Det}}$	$\mu_{C_{\max}}^{\mathrm{NA}}$	FS	$\mathrm{BFS}^{0.3}$	$\mathrm{UFS}^{0.3}$	$\mathrm{wFS}_{\mathrm{succ}}$	TS
30j-15r-4m.ms	-0.08	-0.04	-0.10	-0.05	-0.09	0.12	-0.02
30j-15r-8m.ms	-0.26	-0.28	-0.12	-0.22	-0.10	0.09	-0.05
30j-30r-4m.ms	-0.24	-0.22	-0.20	0.06	-0.11	0.22	-0.11
30j-30r-8m.ms	-0.02	-0.07	0.03	-0.04	-0.13	0.13	0.16
30j-75r-4m.ms	-0.19	-0.17	-0.11	-0.19	-0.25	-0.05	-0.10
30j-75r-4m.ms	-0.19	-0.17	-0.11	-0.19	-0.25	-0.05	-0.10
100j-50r-6m.ms	-0.07	-0.06	0.04	0.21	0.05	0.29	0.12
100j-50r-12m.ms	0.06	0.06	-0.01	0.07	-0.01	0.11	0.04
100j-100r-6m.ms	-0.37	-0.37	-0.23	0.21	0.07	0.30	-0.27
100j-100r-12m.ms	-0.02	-0.00	-0.05	-0.03	-0.08	-0.09	-0.06
100j-250r-6m.ms	-0.17	-0.18	0.12	0.15	0.04	0.35	-0.05
100j-250r-12m.ms	-0.09	-0.07	-0.05	0.05	0.06	0.24	-0.03
Mean	-0.14	-0.13	-0.07	0.00	-0.07	0.14	-0.04

Table 4.10: Spearman Correlation between VarCo and deterministic makespan, normal approximation based mean makespan, mean Free Slack, mean Binary Free Slack (0,30 cutoff), mean Upperbound Free Slack (0,30 cutoff), mean weighted Free Slack, mean Total Slack. Using $Exp(p)$ jobs, 100 Schedules and 300 simulations per schedule.

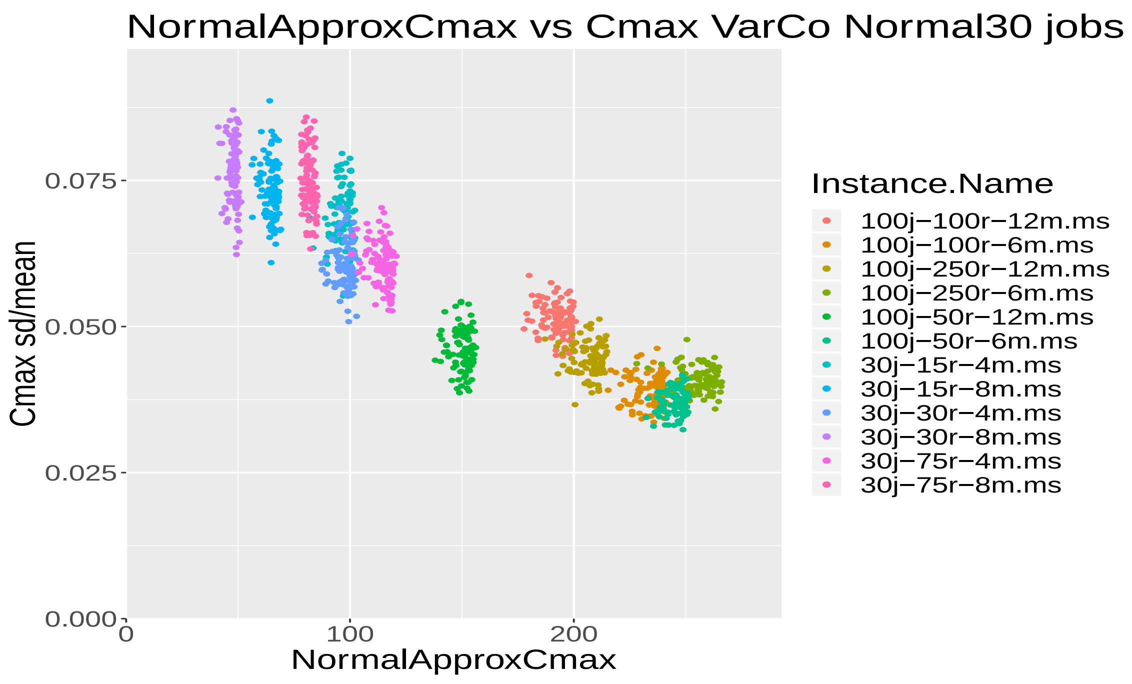


Figure 4.10: $\mu_{C_{\max}}^{\mathrm{NA}}$ vs coefficient of variation of makespan. 100 schedules per problem instance. $N(p, 0.3p)$ jobs. 300 Simulation runs per schedule.

4.4 Conclusion

We present the conclusions we draw from our results below.

Linear Start Delay and Start Punctuality are hard to predict

Our results show that the solution robustness measures we consider (LSD and SP) have a large coefficient of variation. This means that they are hard to predict even using multiple simulation runs. Although does not mean they cannot be used as definitions, which can take a long time (many simulation runs), it does make them hard to estimate.

One problem with these definitions may the following. Consider a simple schedule containing two jobs J_1, J_2 on one machine without any slack between them. Then the start punctuality distribution of J_2 is of the same shape as p_{J_1}, which may be quite volatile. To some extent a problem instance with a large number of jobs reduces this effect.

Perhaps a measure such as the number of precedence arcs (i, j) in which job i and job j are both delayed might of more practical use as a definition. The hope being that it is large only if there is some knocking on of delay and small due to random effect, thus making it less sensitive to noise.

Coefficient of Variation of Makespan may also be hard to predict

The variance in VarCo observed between different schedules might be mostly statistical fluctuation. To check that this is indeed what is behind the results in the previous section, one could use a single schedule. Then perform 100 separate sets of 300 simulations. Each simulation run gives a coefficient of variation value, 100 values in total. The distribution of these VarCo values can then be compared with the distribution of the values of the 100 different schedules. If they are similar, then it cannot be ruled out that the variation in VarCo values is purely random.

In that case, one needs to come up with a different method of generating schedules.

After this part of the research was concluded we noted the following: If a way of generating schedules with different coefficients of variation is found, then one additional robustness measure worth considering may be to apply Algorithm 1, which gives both an estimate for the mean μ and standard deviation σ. Using both these, $\frac{\sigma}{\mu}$ may be a good robustness measure for VarCo.

Slack based measures may be usefull in a local search only with additional requirements.

Slack based measures perform poorly in this setup, in the sense that $C_{\max}^{\mathrm{Det}}$ has a higher rank based correlation for both $\mu_{\hat{C_{\max}}}$ and $\mathcal{C}^{0.95}$. One possible explanation is that our problem does not have a deadline. Therefore, we are not considering ways of delaying jobs along the critical path: All our schedules are earliest start schedules.

In these schedules, free slack along the critical path is always due to release dates of jobs. Therefore, in most cases, the free slack along the critical path is zero (so there is no useful free slack). Similarly, excluding release dates, total slack along the critical path is zero. If a deadline is given, and one is considering different methods of distributing jobs given a machine assignment, then slack based measures may be more effective.

Furthermore, note that in any setup without a deadline, we can optimize slack by delaying the final job indefinitely. Binary and upperbound variants can also be optimized trivially be delaying each job by at least the cutoff. That is, we can always create a schedule with very large slack. However, clearly in most instances such a schedule is undesirable. This means that without bounds on the makespan (such as a deadline), slack based measures should not be used *by themselves* to optimize on.

Nevertheless, the results on SP and LSD indicate that given a set of schedules with a similar $C_{\max}^{\mathrm{Det}}$, using slack based measures to guide the local search might improve the solution robustness. This supports their use as the second phase of a (repeated) two-phase approach.

Best Robustness Estimation Measure to guide a local search.

Based on the results of this chapter, the best robustness estimation measure is the normal approximation approach for both $\mu_{\hat{C_{\max}}}$ and $\mathcal{C}^{0.95}$ and for both the $N(p, 0.3p)$ and $Exp(p)$ distributions. It outperforms all other robustness measures considered, not just on average, but for each problem instance. It is interesting to see that it still outperforms the deterministic approach, even when the job distribution is exponential rather than Normal. We emphasize that our results do not show it can be used to improve SP, LSD or VarCo.

Chapter 5

The effect of inter machine dependencies on robustness measures in problem instances without any slack.

The previous chapter would seem to indicate that when the objective is expected makespan ($\mu_{\hat{C_{\max}}}$) minimization or 95th percentile makespan ($\mathcal{C}^{0.95}$) minimization the deterministic makespan is a reasonable measure to optimize on, although the normal approximation based approach ($\mu^{\text{NA}}_{C_{\max}}$) is better. The results also imply that robustness measures that correlate with $\mu^{\text{NA}}_{C_{\max}}$ also correlate with $\mu_{\hat{C_{\max}}}$.

In this chapter we explore the effects of inter machine precedence arcs on these robustness measures in an extreme case: schedules in which there is no slack.

5.1 Experimental Setup

We have created a set of problem instances with the following characteristics. Each problem instance has an optimal deterministic makespan of 40 and four machines. All release dates are 0. Half the problem instances consist of 16 jobs of processing time 10, and the other half of 160 jobs of processing time 1.

The precedence relations consist of repeating patterns of 16 jobs (4 per machine), which we will call *blocks*. In each such block we compare what we expect to be good and bad ways of assigning a schedule. We have the following block patterns (see figure 5.1).

(a) No inter machine precedence relations. This represents a schedule in which jobs between which precedence relations exist were all assigned to the same machine. Therefore the precedence relations can be forgotten as they are imposed by the machine.

(b) A single cycle, in which there is a dependency path from the top left to the bottom right. This represents a schedule of intermediate quality.

(c) Cyclic precedence relations, in which each job is a successor of the job in the previous column on the previous machine. This represents very bad scheduling. Indeed, clearly it is possible to assign each job along a path to the same machine, resulting in no inter machine dependencies.

(d) Dependencies in a *diamond* pattern, where each job on the first machine in an odd column is dependent on all jobs in the even column.

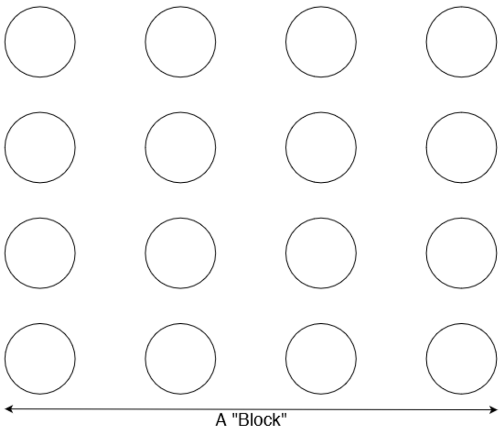
(e) Dependencies in a *rolling diamond* pattern, where each job on the $k(mod 4)$th machine in an odd ($2k + 1$th) column is dependent on all jobs in the even column. That is, this is the same pattern as the diamond pattern, but the job that forms the 'bottleneck' rotates between each of the four machines.

(f) Full dependency, where each job is dependent on all jobs in the previous column.

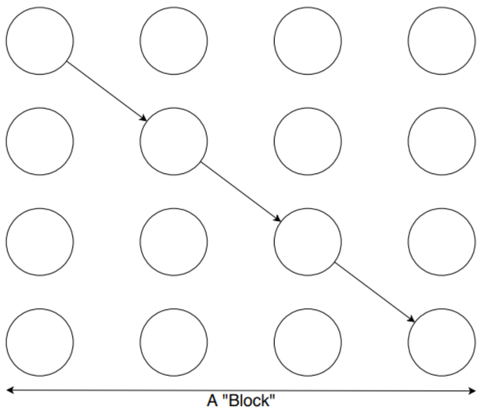
The schedules consist of either one of 16 jobs with processing time 10 (16 jobs in total), or of 10 *blocks* of 16 jobs with processing time 1 (160 jobs total). So we have six patterns, with jobs of time 10 and jobs of time 1. That makes 12 schedules in total.

For each of these 12 schedules, we perform simulations for the following job processing time distributions: $N(p, 0.1p), N(p, 0.3p), LN(p, 0.1p), LN(p, 0.3p), Exp(p)$. Note that the second argument is the standard deviation (not the variance). Simulations are run 300 times.

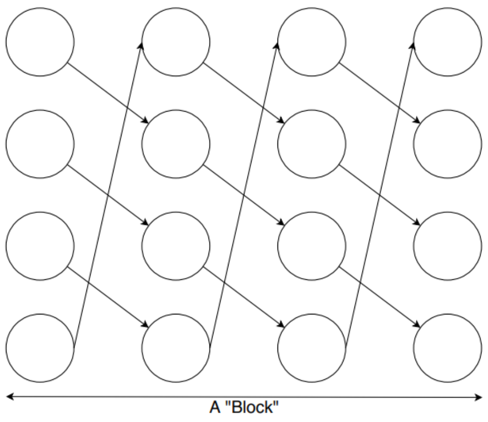
We measure the absolute difference between the simulated makespan and the deterministic makespan. In contrast to the previous chapter, we do not measure the rank correlation. This is because in this case each problem instance has only a single schedule (compared to the 100 schedules in the previous chapter).



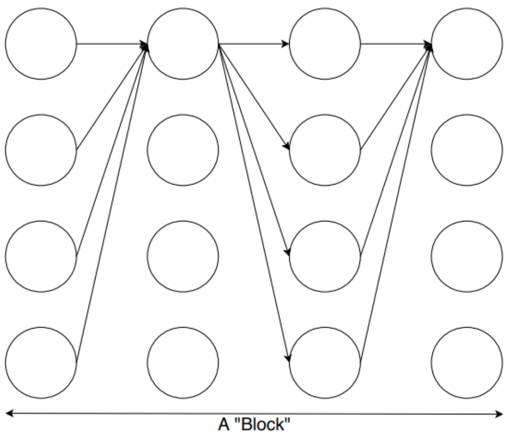
(a) No inter machine dependencies



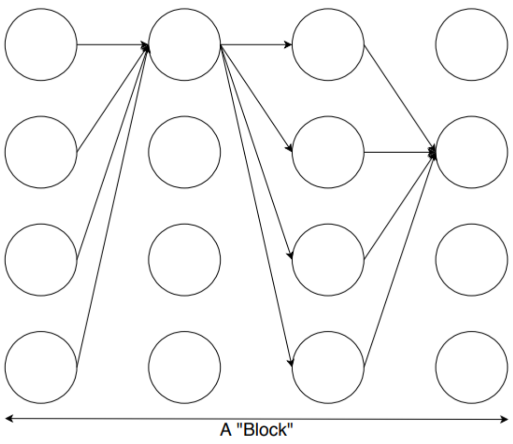
(b) Single Cycle



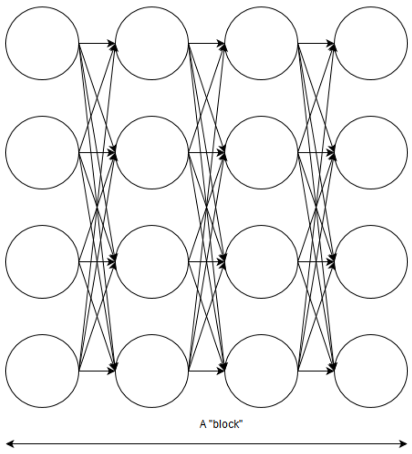
(c) Four cyclic precedence relations



(d) Diamond pattern



(e) Rolling Diamond pattern



(f) Full Dependency

Figure 5.1: The different block patterns. Jobs are represented as nodes, inter machine dependencies as arcs. Jobs on the same row are assigned to the same machine.

5.2 Results

Deterministic Makespan and the Normal Distribution

We start by discussing the results on the $N(p, 0.3p)$ distribution (figures 5.2 and 5.3).

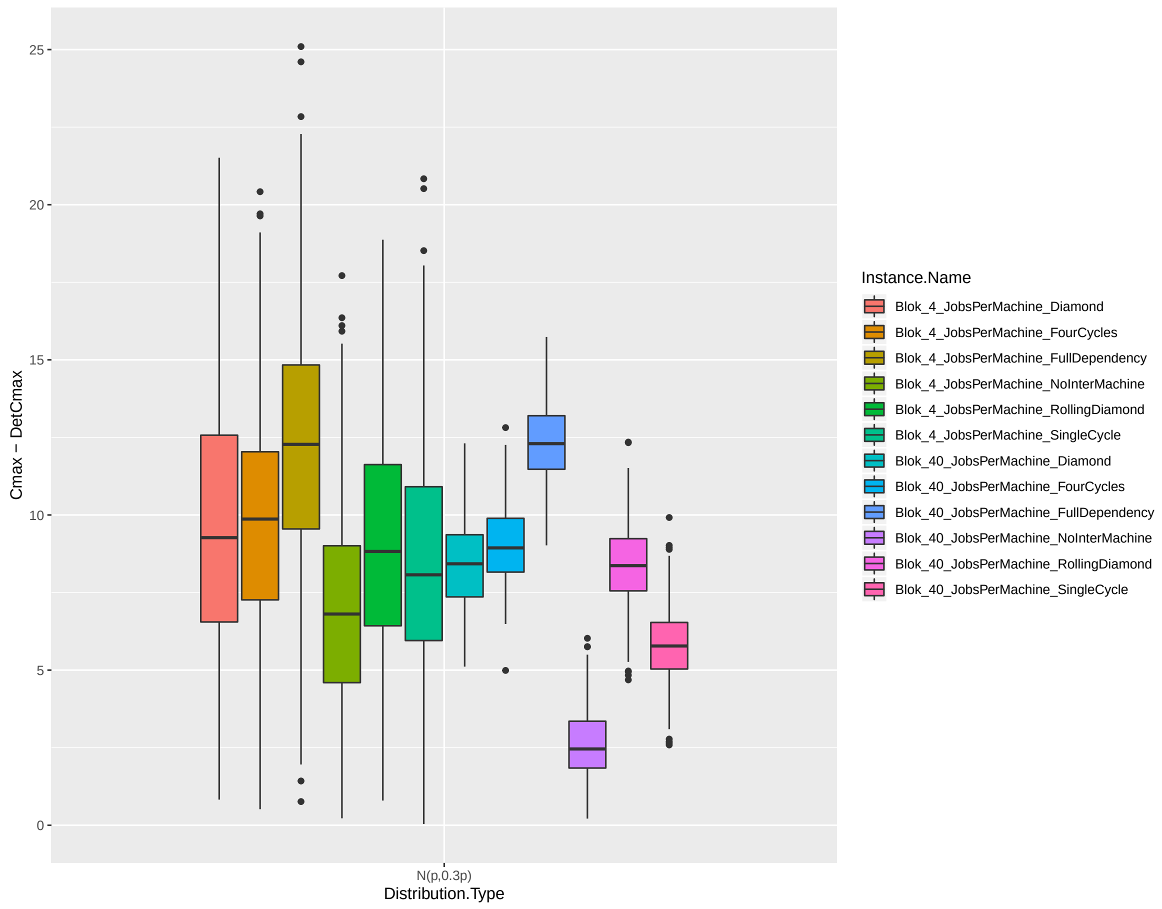


Figure 5.2: Absolute difference between the simulated makespan and the deterministic makespan, for each block pattern and distribution. Box and whisker chart: The box represents the 25th,50th and 75th percentile. The black line (whisker) represents all other values except the outliers, which are shown as black dots.

First, note that the difference between simulated and $C_{\text{max}}^{\text{Det}}$ is frequently more than 4 (10% of $C_{\text{max}}^{\text{Det}}$), for all but one problem instance.

Second, more inter machine arcs results in a higher makespan (and thus a greater difference between the makespan and $C_{\text{max}}^{\text{Det}}$). This is because every time there is an inter machine arc, we must take a maximum between the completion times of the incoming paths. Deterministic makespan in no way accounts for taking the maximum of two distributions. More inter machine arcs therefore imply that the deterministic makespan is a worse estimate.

Furthermore, for the 160 job instances, the diamond pattern has a larger makespan than the rolling diamond pattern. This is likely because the diamond pattern creates a delay on the first machine, whereas the rolling diamond pattern spreads the delay more evenly. Both patterns have the same number of inter machine arcs, they differ only in how the inter machine arcs are distributed. Thus it is not only the number of inter-machine arcs, but also the position in which they occur that is important. Note that to distinguish between these schedules using a number of predecessor based weighting requires including at least two levels of predecessors (i.e. the number of nodes that can be reached following two arcs in the reverse direction - including the not drawn machine arcs).

Third, many small jobs results in both smaller expected makespan and smaller variance in makespan than a few large jobs. This is due to the law of large numbers: significant outliers in the completion time of a single machine are less likely with more jobs per machine. The makespan is determined by the maximum of the completion times of each machine. This maximum is likely to be less extreme when the completion times are nearer their expected values.

Fourth, the pattern of the schedule has a greater effect on the makespan for the 160 job than the 16 job instances. The instances with more jobs repeat the patterns more often, so their effects are more readily observable.

Normal approximation and Normal Distribution

Now we compare the realized makespan with the normal approximation for the block instances (Figure 5.3).

For the single block instances, the normal approximation provides results similar to those from the deterministic makespan. One important difference is that the pattern has a smaller influence on the size of the approximation error when using the normal approximation compared to when using the deterministic makespan.

Interestingly, the normal approximation underestimates the makespan in the single block instances and overestimates the makespan in almost all the instances consisting of 10 blocks. This indicates that the approximation overestimates the uncertainty in the maximum of two distributions. This is to be expected, as the approach only uses information about a jobs direct predecessors. Consider a schedule with four jobs, and precedence arcs $(1,2),(1,3),(2,4),(3,4)$. The start time of job 4 is the maximum of the completion times of jobs 2 and 3. We assume these are independent, but the completion times of jobs 2 and 3 both depend on the completion time of job 1, so this assumption is clearly false.

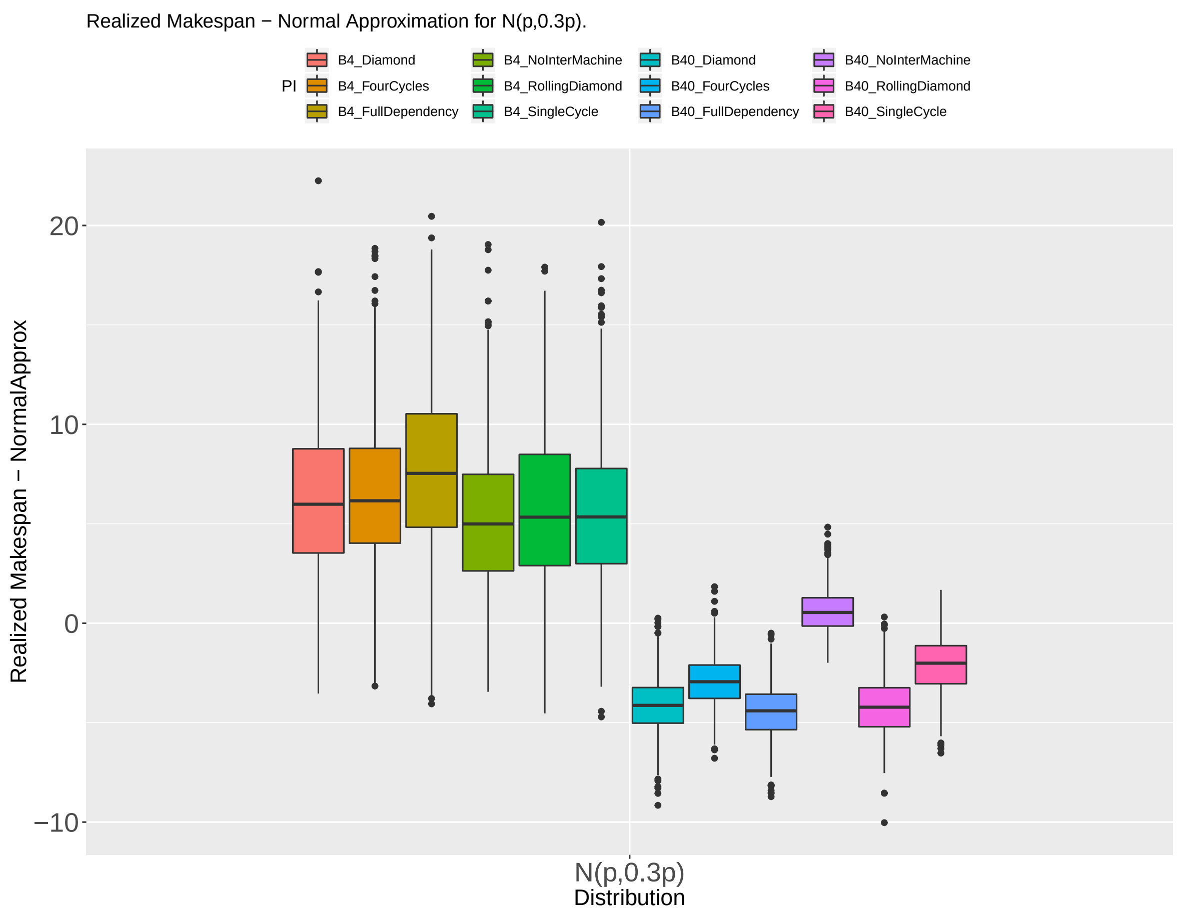


Figure 5.3: Absolute difference between the simulated makespan and the normal approximation, for each block pattern for the $N(p, 0.3p)$ distribution.

Other distributions

Now we consider the effect of different distributions on the makespan. Figures 5.4 and 5.5 show that the observations made above also hold for the other simulated distributions. Other than that, we can see that as expected, a larger variance has a larger impact on the difference between expected makespan and deterministic makespan. The exponential distribution has the largest differences of all. This is because the Exponential distribution has the largest variance.

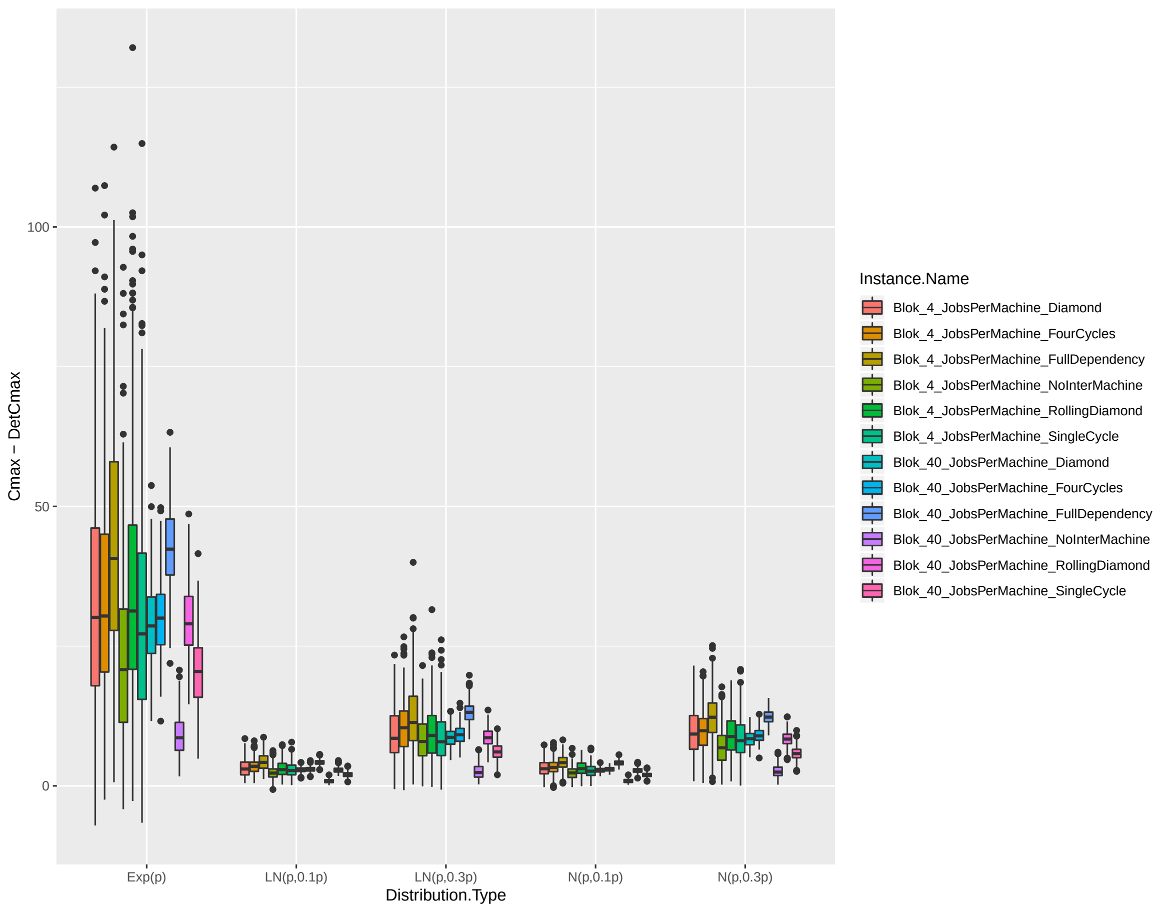


Figure 5.4: Absolute difference between the simulated makespan and the deterministic makespan, for each of the block patterns and distributions.

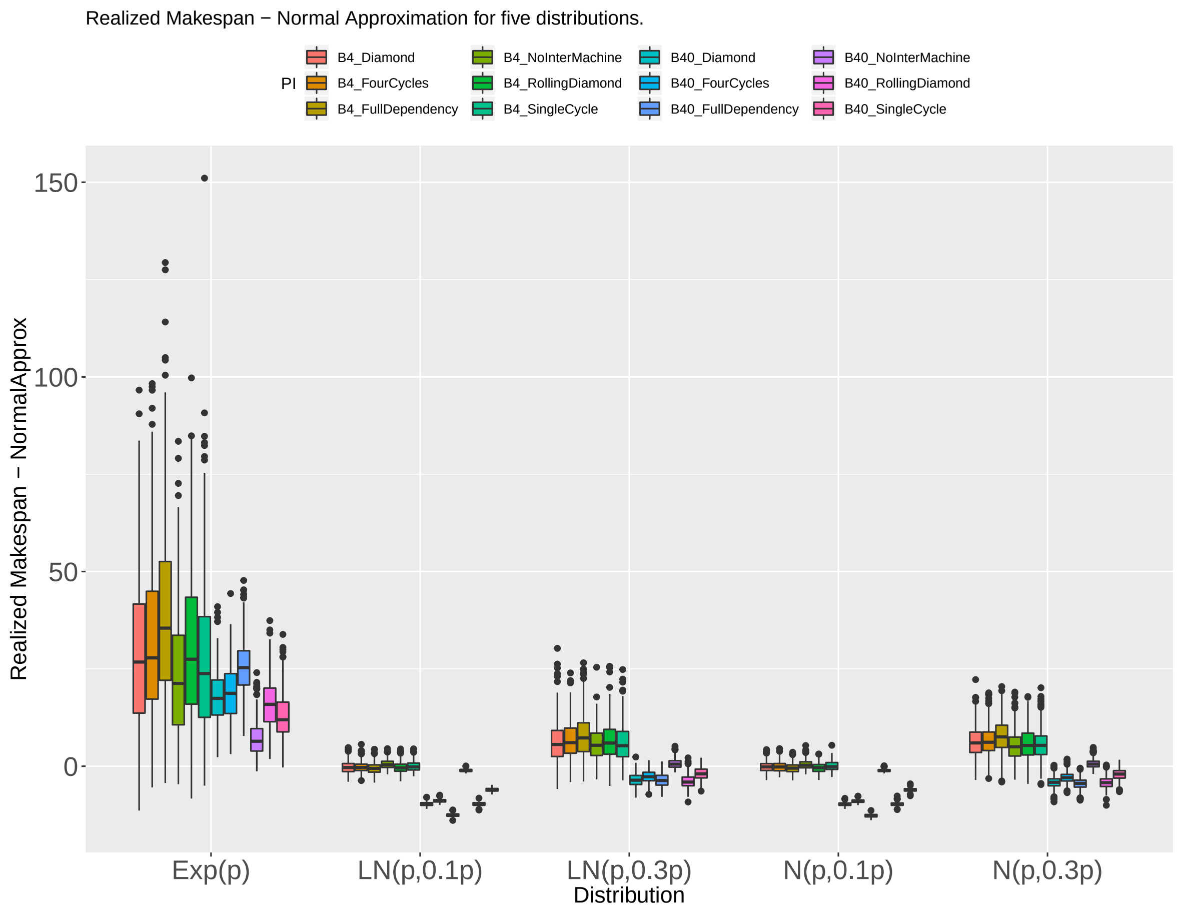


Figure 5.5: Absolute difference between the simulated makespan and the normal approximation, for each of the block patterns and distributions.

5.3 Conclusions

The results from Section 4.3 indicate that the normal approximation based approach ($\mu^{\text{NA}}_{C_{\max}}$) has the best rank correlation with expected makespan ($\mu_{\hat{C_{\max}}}$) and 95th percentile makespan ($\mathcal{C}^{0.95}$). However, since deterministic makespan ($C^{\text{Det}}_{\max}$) also has a good rank correlation with these measures, one may wonder in a practical setting if it is necessary to consider uncertainty in job processing times.

The results form this section provide two arguments for using robustness measures that consider problem structure and uncertainty in job processing times.

First, the problem structure has a significant effect on the realized makespan ($C_{\max}$). Thus robustness measures that attempt to predict $\mu_{\hat{C_{\max}}}$ or $\mathcal{C}^{0.95}$ should consider problem structure.

Second, the normal approximation uses more information about the schedule structure than the deterministic approach, by considering extra uncertainty in the start time of a job with several predecessors. Indeed, deterministic makespan cannot distinguish any of the schedules presented in this chapter. As a result, we see that the $\mu^{\text{NA}}_{C_{\max}}$ is closer to the realized makespan than $C^{\text{Det}}_{\max}$. Furthermore, the difference between $\mu^{\text{NA}}_{C_{\max}}$ and $C_{\max}$ is less dependent on inter machine precedence relations than the difference between $C^{\text{Det}}_{\max}$ and $C_{\max}$.

Thus we conclude robustness measures should consider problem structure. We hypothesize that this becomes more important as the amount of slack in a schedule decreases and as the number of precedence relations increases.

As a consequence, we also conclude that although there is some merit to creating a schedule based on deterministic makespan minimization, given two schedules with (almost) equal deterministic makespan these should not be considered equal. Further distinction based on schedule structure is desirable. In particular, since both the normal approximation and the deterministic makespan run in time and memory linear with the number of precedence relations and jobs, we think it preferable to use the normal approximation over the deterministic makespan approach.

Figure 5.5 shows that there is still some considerable difference between the $\mu^{\text{NA}}_{C_{\max}}$ and the realized makespan. Perhaps robustness measures exist that better consider problem structure and therefore better predict the realized makespan. If so, these measures may also have a better rank correlation with $\mu_{\hat{C_{\max}}}$ or $\mathcal{C}^{0.95}$.

The instances considered here are a special case: There is no slack at all in them. To further our understanding of robustness measures and the effect of inter machine relations, it may be interesting to study similarly structured schedules with fixed $C_{\max}^{\text{Det}}$ where slack is introduced.

Chapter 6

Comparing deterministic makespan and statistical approximation minimization SPMS solutions.

The results in Chapters 4 and 5 indicate that the normal approximation method [1] may be useful as a fitness function, outperforming the deterministic makespan approach, in a local search when the objective is to minimize the expected makespan or the ninety fifth percentile of the makespan.

In this chapter we compare the schedules generated by a multi-start local search (MLS) algorithm[2] that uses deterministic makespan as a fitness function and an MLS algorithm that uses the normal approximation as a fitness function.

6.1 Setup

For the problem instances discussed in 4.2.2 we run our MLS algorithm twice. Once in which the fitness function is deterministic makespan and once in which the fitness function is the normal approximation. Recall that each MLS application gives us the 100 best schedules out of 1000 locally optimal schedules. So for each problem instance we have 100 schedules based on deterministic makespan minimization and 100 schedules based on normal approximation minimization. Every schedule is simulated 300 times

[1]see Section 3.2 for a full description

[2]Algorithm 5 from Section 4.2

with $N(p, 0.3p)$ job processing times and 300 times with $Exp(p)$ job processing times. For each schedule we calculate the sample mean (Equation 3.6) and the ninety fifth makespan percentile (Equation 3.2). We use these quantitative definitions of robustness because the results in Section 4.3 indicate that the other suggested definitions are not predicted by any of the robustness measures we have considered.

6.2 Results

We present the results in two parts. First we discuss the expected makespan results and then the ninety-fifth makespan percentile results.

Expected makespan minimization

Figures 6.1 and 6.2 show very little difference between the mean makespan of schedules created by minimizing $C_{\max}^{\text{Det}}$ and created by minimizing $\mu_{C_{\max}}^{\text{NA}}$. Indeed a two sided Kolmogorov-Smirnov test for any given problem instance and job distribution fails to reject the possibility that the $C_{\max}^{\text{Det}}$ and $\mu_{C_{\max}}^{\text{NA}}$ results are drawn from the same underlying distribution.

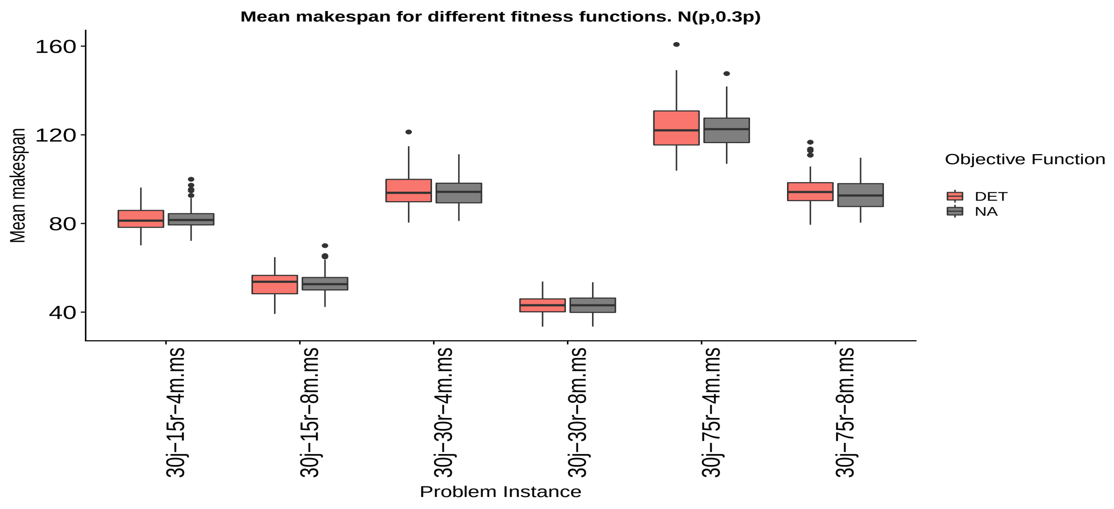


Figure 6.1: For each problem instance, a comparison of the mean makespan of schedules found by the MLS algorithm when minimizing deterministic makespan (DET) and when minimizing the normal approximation (NA). $N(p, 0.3p)$ jobs. 300 simulation runs per schedule.

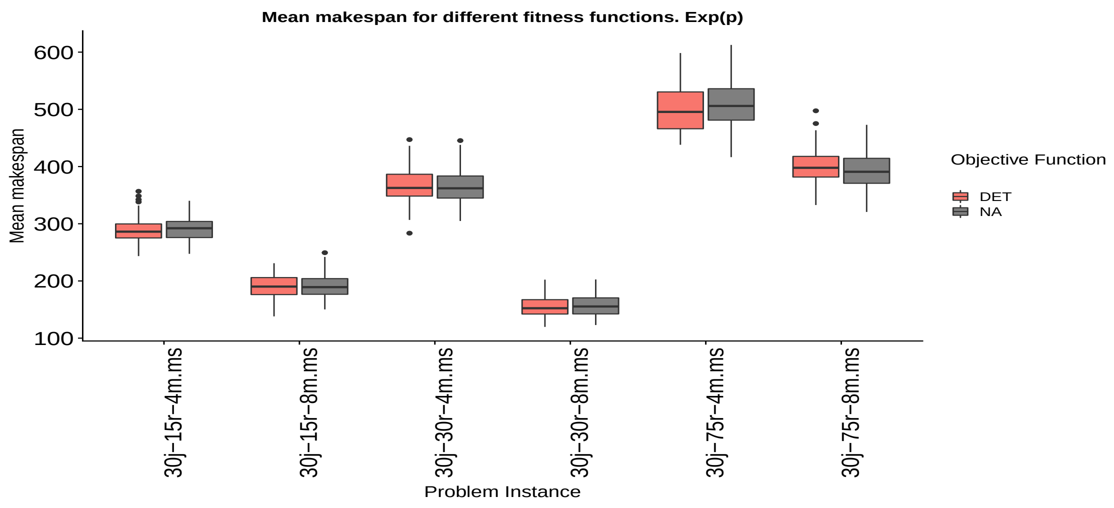


Figure 6.2: For each problem instance, a comparison of the mean makespan of schedules found by the MLS algorithm when minimizing deterministic makespan (DET) and when minimizing the normal approximation (NA). *Exp*(p) jobs. 300 simulation runs per schedule.

Ninety-fifth makespan percentile minimization

As was the case for expected makespan, both Figure 6.3 and Figure 6.4 show very little difference between the ninety-fifth percentile of the makespan of schedules created by minimizing $C^{\text{Det}}_{\text{max}}$ and by minimizing $\mu^{\text{NA}}_{C_{\text{max}}}$. Again, a two sided Kolmogorov-Smirnov fails to reject the possiblity that for any given problem instance and job distribution the $C^{\text{Det}}_{\text{max}}$ and $\mu^{\text{NA}}_{C_{\text{max}}}$ results are drawn from the same underlying distribution.

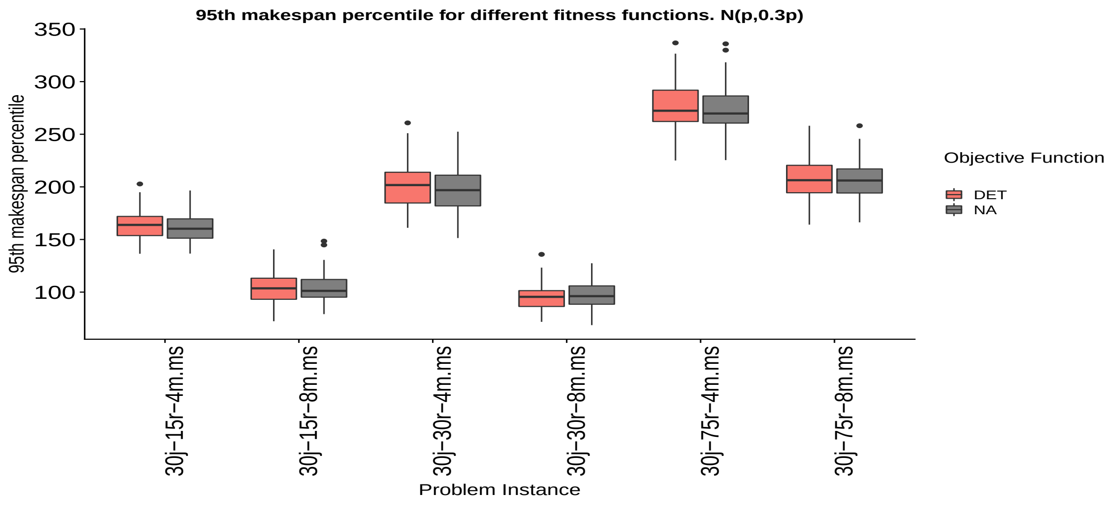


Figure 6.3: For each problem instance, a comparison of the 95th makespan percentile of schedules found by the MLS algorithm when minimizing deterministic makespan (DET) and when minimizing the normal approximation (NA). $N(p, 0.3p)$ jobs. 300 simulation runs per schedule.

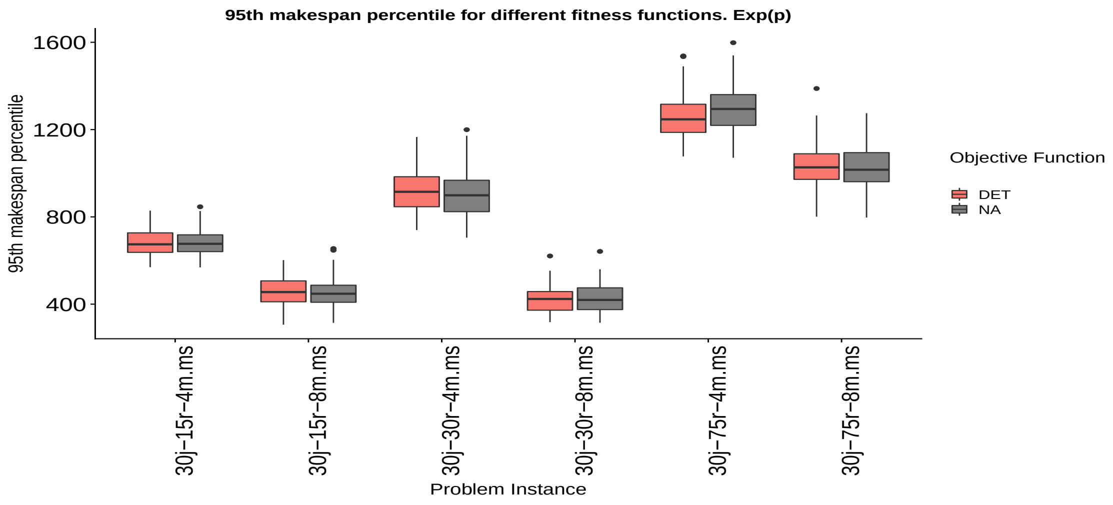


Figure 6.4: For each problem instance, a comparison of the 95th makespan percentile of schedules found by the MLS algorithm when minimizing deterministic makespan (DET) and when minimizing the normal approximation (NA). $Exp(p)$ jobs. 300 simulation runs per schedule.

6.3 Conclusion and Discussion

The results in this chapter show no improvement to either $\mu_{\hat{C_{\max}}}$ or $\mathcal{C}^{0.95}$ by using the normal approximation approach over the deterministic makespan approach. This contradicts the expectation based on the results from the previous two chapters.

One possible explanation for this combination of results may be that the quality of schedules produced by the MLS algorithm is generally far from the optimum and thus have a lot of slack. That means that the conclusions from Chapter 5 do not apply to these schedules. Furthermore, if the MLS algorithm is indeed the bottleneck on solution quality, then the slight difference in correlation observed in Chapter 4 may not be exploited. That is, if the quality of the MLS solutions remains in a region where the difference between the quality of locally optimal schedules is still large, then given two schedules both $C_{\max}^{\text{Det}}$ and $\mu_{C_{\max}}^{\text{NA}}$ can predict which is better. In such a region of the solution space there is no real advantage to using either method.

It would be interesting further research to see if this is indeed the underlying cause and if so, if there is some predictable relation between difference in solution quality between metaheuristics guided by different robustness measures with comparable global optima and these global optima.

Chapter 7

Conclusion

7.1 Summary of Conclusions

In this section we will summarize the conclusions drawn at the end of Chapters 4,5 and 6.

In Chapter 4 we conclude that Linear Start Delay (LSD) and Start Punctuality (SP) are hard to predict because of their large variation. Slack based measures may help to predict them. No robustness measure considered predicts the variation of coefficient (VarCo) well. Because LSD, SP and VarCo have high variation, we focus on $\mu_{\hat{C_{\max}}}$ and $\mathcal{C}^{0.95}$. $\mu_{\hat{C_{\max}}}$ and $\mathcal{C}^{0.95}$ have the highest rank based correlation with $\mu^{\text{NA}}_{C_{\max}}$ and a high rank based correlation with $C^{\text{Det}}_{\max}$. Slack based measures have low rank based correlations with $\mu_{\hat{C_{\max}}}$ and $\mathcal{C}^{0.95}$. We thus expect $\mu^{\text{NA}}_{C_{\max}}$ to be the best robustness measure to guide a local search when the aim is to minimize $\mu_{\hat{C_{\max}}}$ or $\mathcal{C}^{0.95}$.

In Chapter 5 we conclude that the number and position of inter machine dependencies influences the realized makespan $C_{\max}$. Thus robustness measures that attempt to predict $\mu_{\hat{C_{\max}}}$ or $\mathcal{C}^{0.95}$ should consider problem structure. Again, this suggests $\mu^{\text{NA}}_{C_{\max}}$ is a better robustness measure than $C^{\text{Det}}_{\max}$.

In Chapter 6 we conclude that there is no significant difference in $\mu_{\hat{C_{\max}}}$ or $\mathcal{C}^{0.95}$ between schedules resulting from Algorithm 5 using $C^{\text{Det}}_{\max}$ as a robustness measure and the same algorithm using $\mu^{\text{NA}}_{C_{\max}}$ as a robustness measure.

7.2 Comparison to similar work.

[PAH] use a statistical approximation approach similar to Algorithm 1. They find that using this approach does help to minimize $\mu_{\hat{C_{\max}}}$. In this section we list similarities and differences between this work and [PAH]. Our aim is to shed light on why the Normal Approximation approach improves their solutions but does not help to improve ours.

First, both works consider the problem, where uncertainty is due to job processing times. The distributions of job processing times considered are the same. So are the number of machines and the pairs of jobs between which we introduce precedence relations. The problem instances differ in only one respect: They use start-start precedence relations that may include lag, whereas we use 0-lag finish-start precedence relations. This difference means that some jobs that must be performed one at a time in our instances can be performed simultaneously on different machines in theirs. This would mean that in their instances the expected makespan of schedules may be lower (effecting $\mu_{\hat{C_{\max}}}$). Conversely, the introduction of lag in precedence relations may introduce slack in a schedule that is not required in our schedules (increasing $\mu_{\hat{C_{\max}}}$). The observed values of $\mu_{\hat{C_{\max}}}$ are smaller in [PAH] than in this work. This may be due to the fact that they use and iterated local search approach or as stated earlier, simultaneous performance of jobs. Finally and importantly, although Algorithm 1 is based on their statistical approach, it differs due the different precedence relation types used. This has no effect on $\mu_{\hat{C_{\max}}}$, but may clearly influence the correlation between $\mu_{\hat{C_{\max}}}$ and the robustness measure.

This gives us several hypotheses as to the difference in conclusions drawn. First, it may be that the schedules generated in their work are closer to the possible optimum, i.e. slack is minimized, in which case considering robustness becomes more important. The results in Chapter 5 support this hypothesis. Second, it may be that robustness is more important in start-start precedence relation problems if they are somehow 'harder' to solve robustly. The fact that the adaptation of their statistical approach involved a simplification step supports this hypothesis. Finally, it may simply be that the adaptation to the statistical approach makes it less accurate in some undetermined way. We have no indication that this is the case.

7.3 General Conclusion

In addition to the conclusions drawn in Chapters 4,5 and 6 we can draw the following general conclusion: The choice of problem instance and definition of robustness may have a large effect on the conclusions drawn. This highlights the need for clear and universal definitions of robustness, so that works can be more readily compared and patterns recognized.

7.4 Further Research

We propose further research to determine what factors in a problem make it so that robustness needs to be considered. In Chapter 5 we considered the effect of inter machine dependencies. It would be interesting to perform a similar study with artificial schedules, where the amount of slack is varied ceteris paribus. In such a project one may also consider the effect of allowing or disallowing jobs to start before their planned start date.

Bibliography

[ABH13] J.M. van den Akker, Kevin van Blokland, and J.A. Hoogeveen. “Finding robust solutions for the stochastic job shop scheduling problem by including simulation in local search”. In: *Experimental Algorithms - SEA* 7933 (2013), pp. 402–413.

[AH05] M. Al-Fazwan and M. Haouari. “A bi-objective model for robust resource-constrained project scheduling”. In: *Int. J. Production Economics* 96 (2005), pp. 175–187.

[AH08] J.M. van den Akker and J.A. Hoogeveen. “Minimizing the number of late jobs in a stochastic setting using a chance constraint”. In: *Journal of Scheduling* 11 (2008), pp. 59–69.

[AHK12] J.M. van den Akker, J.A. Hoogeveen, and J.W. van Kempen. “Using column generation to solve parallel machine scheduling problems with minmax objective functions”. In: *Journal of Scheduling* 15 (Aug. 2012), pp. 801–810.

[AL03] E.H.L. Aarts and J.K. Lenstra. *Local Search in Combinatorial Optimization.* Princeton University Press, 2003.

[Bal07] Francisco Ballestin. “When it is worthwile to work with the stochastic RCPSP”. In: *Journal of Scheduling* 10.10 (2007), pp. 153–166.

[BKF12] M. Brcić, D. Kalpić, and K. Fertalj. *Resource Constrained Project Scheduling under Uncertainty: A Survey.* Tech. rep. University of Zagreb, 2012.

[BL09] Francisco Ballestín and Roel Leus. “Resource-Constrained Project Scheduling for Timely Project Completion with Stochastic Activity Durations”. In: *Production and operations management.* 18.4 (July 2009), pp. 459–474.

[CH08] H. Chtourou and M. Haouari. “A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling”. In: *Computers Industrial Engineering* 55.55 (Jan. 2008), pp. 183–194.

[Deb$^+$07] Filip Deblaere et al. “A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling”. In: *Decision Sciences* 38.1 (Feb. 2007).

[DH02] Eric Demeulenmeester and Willy Herroelen. *Project Scheduling: A research handbook*. Kluver Academic Publishers, 2002.

[DVH03] Erik Demeulenmeester, Mario Vanhoucke, and Willy Herroelen. “RanGen: A random network generator for activity-on-the-node networks”. In: *Journal of Scheduling* 6 (Jan. 2003), pp. 17–38. URL: https://doi.org/10.1023/A:1022283403119.

[Gre03] W.H. Greene. *Econometric analysis*. Pearson Education India., 2003.

[GS08] Seluk Goren and Ihsan Sabuncuoglu. “Robustness and stability measures for scheduling: single-machine enviroment”. In: *IIE Transactions* 40.1 (2008), pp. 66–83.

[HAH11] D.J. Hoppenbrouwer, Marjan van den Akker, and Han Hoogeveen. “Robust parallel machine scheduling with relations between jobs”. MSc thesis. Utrecht University, Nov. 2011.

[Har98] S Hartmann. “A competitive genetic algorithm for resorce-constrained project scheduling.” In: *Naval Research Logistics* 45.7 (1998), pp. 733–750.

[HHE10] O. Hazir, M. Haouari, and E. Erel. “Robust Scheduling and robustness measures for the discrete time/cost trade-off problem.” In: *European Journal of Operational Research* (2010).

[HK00] S. Hartmann and R. Kolisch. “Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem”. In: *European Journal of Operational Research* 127 (2000), pp. 394–407.

[HL04a] W. Herroelen and R. Leus. “Robust and reactive project scheduling: a review and classification of procedures”. In: *International Journal of Production Research* 42.8 (2004), pp. 1599–1620.

[HL04b] W. Herroelen and R. Leus. “Robust and reactive project scheduling: a review and classification of procedures”. In: *European Journal of Operational Research* 156.3 (Aug. 2004), pp. 550–565.

[HL04c] W. Herroelen and R. Leus. “The construction of stable project baseline schedules”. In: *European Journal of Operational Research* 156 (2004), pp. 550–565.

[HL05] W. Herroelen and R. Leus. “Project scheduling under uncertainty: Survey and research potentials”. In: *European Journal of Operational Research* 165 (2005), pp. 289–306.

[Mas03] M Mastrolilli. "The use of buffers in project management: The trade-off between stability and makespan". In: *Journal of Scheduling* 6 (2003), pp. 521–531.

[MG10] M. Mastrolilli and L. Gambardella. *Effective Neighborhood Functions for the Flexible Job Shop Problem.* Tech. rep. IDSIA - Istituto Dalle Molle di Studi sull´Intelligenza Artificiale, Sept. 2010.

[NK08] S. Nadarajah and S. Kotz. "Exact Distribution of the Max/Min of Two Gaussian Random Variables". In: *IEEE transactions on very large scale integration (VLSI) systems.* 16.2 (2008), pp. 521–531.

[PAH] Guido Passage, Marjan van den Akker, and Han Hoogeveen. "Improving the performance of local search for stochastic parallel machine scheduling by estimating the makespan". MA thesis.

[Pin12] M. Pinedo. *Scheduling: Theory, Algorithms and Systems.* Fourth Edition. Springer, 2012.

[RAH] D.J. Roermund, Marjan van den Akker, and Han Hoogeveen. "Robustness in parallel machine scheduling". MA thesis.

[RCL17] S. Rostami, S. Creemers, and R. Leus. "New strategies for stochastic resource-constrained project scheduling". In: *Journal of Scheduling* (2017). URL: https://doi.org/10.1007/s10951-016-0505-x.

[RS64] B. Roy and B. Sussmann. *Les problemes d´ordonnancement avec contraintes disjonctives.* Tech. rep. SEMA, 1964.

[Sal] E. Saliby. "Descriptive Sampling: A Better Approach to Monte Carlo Simulation". In: *J. Opl Res. Soc.* 41.12 (), pp. 1133–1142.

[TB06] Vincent T'Kindt and Jean-Charles Billaut. *Multicriteria Scheduling: Theory Models and Algorithms.* Springer, 2006.

[Thi] D.A. Thierens. *Metaheuristic Search for Combinatorial Optimization.* URL: http://www.cs.uu.nl/docs/vakken/ea/slides/MetaHeuristic.pdf.

[VBQ05] Vincente Valls, Francisco Ballestín, and Sacramento Quintanilla. "Justification and RCPSP: A technique that pays". In: *European Journal of Operational Research* 165 (Nov. 2005), pp. 375–386.

[Von^{+}05] S. van de Vonder et al. "The use of buffers in project management: The trade-off between stability and makespan". In: *Int. J. Production Economics* 97 (2005), pp. 227–240.

Appendix A

Notation

An overview of the symbols used in the 3 field notation:

- P_m Parallel machine scheduling on m machines
- $C(\sigma)_j$ The completion time of job j in schedule σ
- r_j The release date of job j: the time at which it becomes available for processing
- d_j The due date of job j: the time by which we would prefer to have job j finished (a soft constraint).
- $\bar{d}_j$ The deadline of job j: the time by which job j must be finished (a hard constraint).
- $C_{\max}(\sigma)$ The maximum completion time in schedule σ: The smallest time by which all jobs are finished. Also known as the *makespan* $C_{\max}(\sigma) = \max_j\{C(\sigma)_j\}$
- $L_{max}(\sigma)$ The maximum lateness in schedule σ: The largest difference between due date and completion time of a job. $L_{max}(\sigma) = \max_j\{C(\sigma)_j d_j\}$
- prec Denotes that precedence relations between jobs exist. This work discusses only precedence constraints where a job may start as soon as all it's predecessors are completed, known as *0-lag finish-start* precedence contraints.
- $\mathbf{p}_j$ Denotes that the processing times are stochastic.

An overview of abbreviations used:

- RM: Robustness measure
- PMS: Parallel machine scheduling

Symbol	Meaning
Job properties	
p_j	Mean processing time of job
$\mathbf{p_j}$	Realised processing time of job j
r_j	Release date of job
Problem instance properties	
σ_j	Direct successors of job j
π_j	Direct predecessors of job j
${\sigma_j}^*$	Transitive successors of job j
${\pi_j}^*$	Transitive predecessors of job j
Schedule Properties	Sometimes followed by (S) to indicate the schedule.
π_j^M	Machine Predecessor of job j in a schedule.
σ_j^M	Machine Successor of job j in a schedule.
s_j	Planned start time of job j in a schedule.
$s_j(ESS)$	Start time of job j in the earliest start schedule
$s_j(LSS)$	Start time of job j in the latest start schedule
$\mathbf{s_j}$	Realised start time of job j in a schedule.

Table A.1: List of symbols frequently used in formulae.

- SPMS: Stochastic parallel machine scheduling
- RCPSP: Resource constrained project scheduling problem
- SRCPSP: Stochastic resource constrained project scheduling problem
- LS: Local Search

Appendix B

Other starting heuritics for the MLS procedure.

Algorithm 6: Greedy Load Balancing

```
Data: A problem instance
Result: A feasible assignment of jobs to machines
1 Set the load of each machine to 0. ;
2 while Unassigned jobs exist do
3     Select the next job without unassigned predecessors;
4     Assign it to the lowest indexed machine of minimum load;
5     Update the load;
6     Update list of unassigned jobs;
7 end
```

Algorithm 7: Round Robin Assignment

```
Data: A problem instance
Result: A feasible assignment of jobs to machines
1 $M \leftarrow$ the number of machines;
2 CurrentMachineID $\leftarrow$ Random Integer between 0 and $M$ ;
3 while Unassigned jobs exist do
4     Select the next job without unassigned predecessors;
5     Assign it to the machine with ID CurrentMachineID;
6     CurrentMachineID $\leftarrow$ CurrentMachineID $+1 \mod M$;
7     Update list of unassigned jobs;
8 end
```