A framework for dynamic rescheduling problems

Larsen, Rune; Pranzo, Marco

Published in:
International Journal of Production Research

Link to article, DOI:
10.1080/00207543.2018.1456700

Publication date:
2019

Document Version
Peer reviewed version

UNIVERSITÀ DI SIENA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

A Framework for Dynamic Rescheduling Problems

Rune Larsen, Marco Pranzo

DII Technical Report **n.2012-01**

Research Group: Decision and Management Methods **April 30, 2012**

A Framework for Dynamic Rescheduling Problems

Rune Larsen, Marco Pranzo

April 30, 2012

Abstract

Academic scheduling problems usually assume deterministic and known in advance data. However this situation is not often met in practice, since data may be subject to uncertainty and it may change over time. In this paper we introduce a general rescheduling framework to address such dynamic scheduling problems. The framework consists in a solver and a controller. The solver can also assume deterministic and static data, whereas the controller is in charge of triggering the solver when needed and when possible. Extensive tests are carried out for two well-known scheduling problems, namely the single machine weighted completion time problem and the job shop problem. The tests show how the proposed framework allows to deal with different scheduling problems and to select the best framework configuration.

Keywords: Scheduling, Simulation, Dynamic rescheduling.

1 Introduction

Scheduling problems are among the hardest and the most studied problems in operations research. However, as observed by many authors [10, 12], there is still a gap between theory and practice, in fact most formulations of scheduling problems usually assume that data is deterministic and static over time. It is well known that both uncertainties and dynamic environments are often present in real-world applications. Nevertheless these presences are often neglected since they are hard to model and to take into account algorithmically. Usually, simplifying assumptions are done to assume all information deterministic, static over time and known in advance. In some contexts these assumptions may be reasonable since they lead to simplified and more tractable models. However, a careful validation analysis on the problem is often required to assess whether simplified models are reasonable or if they induce an oversimplification.

When the environment is assumed to be non-deterministic and dynamic the problem is to formulate a plan in which the presence of the uncertainties has been taken into account during the solution process. At least two kind of approaches have been proposed in the literature:

1. Approaches that try to produce the best possible plan incorporating all the available information on the uncertainties. Among these approaches we can mention *stochastic programming* [21] and *robust optimization* [4].

In stochastic programming the idea is trying to include the uncertainty sources directly in the formulation of the problem by generating and solving several scenarios, if necessary allowing changes in the solution over time. However, allowing recurse decisions makes the problem fundamentally intractable. In robust optimization the idea is to try to produce a "robust" solution satisfying additional constraints taking into account possible realizations of the uncertainty.

2. Approaches that try to represent the dynamic nature of the problem as an additional layer between the solver and the application. In this case the problem is usually maintained deterministic and the presence of the uncertainty is addressed in a external wrapper. As time passes, whenever some conditions are met, the deterministic scheduling solver is invoked and a new problem corresponding to a current "snapshot" of the system is solved. The results are then considered as a new plan to be followed during the execution.

The advantages of dynamic rescheduling based approach over stochastic or robust optimization are threefold:

Tractability A rescheduling approach allows solving larger instances within reasonable computation times allowing, if application permits, hundreds of reschedules over time.

Easiness of implementation This approach does not require the development of a new solver for the problem at hand, if the deterministic solver can be adapted to meet some simple requirements. Hence, it can be put in production faster.

Knowledge of the problem This approach is robust with respect to errors in the uncertainty modelling, i.e., the effects of different probability distributions and parameters, since there is no need to embed the uncertainty in the solver.

However there are also shortfalls for these approaches. In fact, better performances can be achieved if the solver takes the information on the uncertainties into account within the optimization process. Moreover, no theoretical results are known for this kind of approach.

In this paper we take a dynamic rescheduling approach, in which a deterministic scheduling solver is dynamically executed to update the plan in order to adapt uncertainties as they happen. The focus of this work is to introduce a general framework to model a dynamic rescheduling problem in an unified way. The main components of the system are:

Solver The solver is in charge of actually solving at each iteration the scheduling problem and thus producing a plan that is going to be executed.

Controller The Controller checks for rescheduling conditions during the execution of the schedule. Whenever a trigger condition is met, a new rescheduling problem is formulated according to the information available at the moment and solved by the solver.

Simulator The simulator component loads all the information on the real-world realizations of the probability functions when started, and it is responsible for simulating the unpredicted deviations and disturbances that may happen in real-time. Note that the information is not sent to the solver.

Our aim is to study when the uncertainty starts to be relevant in terms of feasibility and optimality and how different rescheduling policies behaves. The emphasis is given to the general applicability of the method rather than to the specific application or solver used. In fact, the answers to these questions may be problem specific, however, it is important to have a single tool able to address them.

The paper is organized as follows: In the next section we introduce definition and notation. Section 3 briefly reviews the related literature and in Section 4 we introduce and describe the proposed framework and all its structural components in detail. Section 5 shows the application of the framework to two different scheduling problems. Finally conclusions and future research directions follow.

2 Definitions and notation

A scheduling problem can be classified as *static* if all the data is available at the planning stage and no new information is added to or modified in the problem during the execution of the planned schedule. On the other hand, *dynamic* scheduling refers to problems where data may change during the execution of the scheduling and some information is not available to the scheduler at the planning stage.

A problem is *deterministic* if all information is certain. In *stochastic* scheduling problems, some data may be uncertain at planning stage. Different kinds of information could be available to the scheduler such as the probability distribution or expected values. A *stochastic instance* has probability distributions associated to each process time, and the deterministic instance resulting from sampling these distributions will be called a *sample instance*.

When solving a scheduling problem, two phases can be distinguished namely, a *planning phase*, in which the scheduler has to plan a schedule to solve the instance at hand, and an *execution phase*, in which the plan is executed and, depending on the application, it may be modified or not. Usually in scheduling research the main focus is on the planning phase [14].

A *disruption/disturbance* of the plan is an unforeseen event that affects the plan typically during its execution. It may consist in the realization of an uncertain event which was known in the form of probability distribution, or it may be a more disruptive event such as the breakdown of a machine, failure of operations, the arrival of some new and urgent job/order to be processed or canceled jobs etc. When a disruption occurs, and if the application setting makes it possible, the scheduler has to decide whether to do nothing or reschedule. In the former case the scheduler continues to follow the plan. While in the latter case the scheduler should build/update the previous plan to reflect changes due to the disruption.

Rescheduling is the act of modifying the offline plan in response to disruptions. Rescheduling is usually an expensive activity in terms of costs, time

and/or information exchange. Depending on the actual application the reschedule may not be feasible at all or it can be allowed with continuity as a monitoring process executed along the schedule execution process. The *rescheduling frequency* states the minimum time interval between two consecutive reschedules, i.e., it regulates how often a reschedule process can be pursued. The *rescheduling policy* decides when a new rescheduling process can be started. Rescheduling policies can be classified as *periodic*, *continuous*, *event-driven* or *hybrid* [14]. The periodic policy states that a rescheduling can be started after a fixed amount of time. In continuous rescheduling the rescheduling process is carried out after every timestep. Hybrid rescheduling is a strategy in which rescheduling is started after a fixed time interval or in response to some events. The event-driven policy states that a reschedule can start in response to a some specified events. It can be noted that the event-driven category is the most general since it can contain all the other cases. The *reaction time* is the time between a disturbance and when the solver starts the rescheduling process. Clearly, it depends on the rescheduling frequency but there may also be a minimum reaction time caused by the physical infrastructure. Finally a *rescheduling objective* has to be specified. We distinguish between *complete optimization* in which the rescheduling algorithm optimizes the same objective function used in the planning phase and a *partial reoptimization* in which the aim of the rescheduling is to minimize a surrogate objective function. Partial rescheduling (schedule repair) occurs when the rescheduling process tries to minimize the deviation from the available offline plan and possibly taking the "real" objective function into account. This is common in applications where the system should follow a plan known in advance [7]. Observe that, the deviation with respect to the plan can be limited also by acting on the constraints of the problem [23], i.e., by keeping precedence relations fixed or imposing time windows on the starting/ending time of the operations. Other approaches may consider a multiobjective problem in which both the "real" objective function and the surrogate function to consider stability are simultaneously optimized [18].

The *rolling horizon* is the length of the time period for which a schedule has to be produced. Events outside the rolling horizon are not to be considered in the rescheduling. The presence of a short rolling horizon causes smaller instances since one has to schedule only the operations within the rolling horizon but it may generate myopic schedules, while longer rolling horizon generates larger instances and potentially leads to better schedules. An acceptable trade-off between these two contrasting needs should be found.

The *frozen period* is the length of the part of the schedule that cannot be changed, and that should be maintained. Operations already in execution or operations with imminent starting times that cannot be postponed, usually are considered *locked*, i.e., in the frozen period. Technological constraints may cause different length of frozen periods depending on the actual application.

The *allowed computation time* is the maximum time allotted to the optimization algorithm to compute a schedule. It is clearly dependent on the application and it is influenced by the rolling horizon and frozen times. Some applications may require fast algorithms whereas in others settings the length of the allowed computation time for the rescheduling can be comparable to the CPU time for calculating the offline schedule.

Figure 1 illustrates the frozen period, the rolling horizon and the allowed computation time ($t_{end} - t_{start}$). The proposed rescheduling framework can

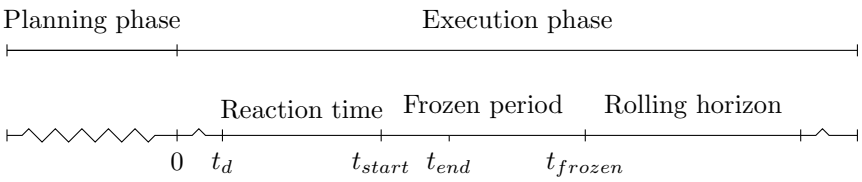

Figure 1: A time line for rescheduling scenarios. A disturbance occurs at time t_d, rescheduling starts at t_{start} and ends before t_{end}.

reduce the complexity by considering an allowed computation time $t_{computation}$ and all operations starting before t_{frozen} locked.

2.1 Mathematical notation

In this paper we assume that the deterministic scheduling problems solution can be represented using a temporal network (i.e., a graph representation) such as [20, 8, 11]. The scheduling problem can be represented as a set of events/operations $\{o_0, o_1, \ldots, o_n\}$ and a set of precedence relations among operations. Each operation has a *minimum starting time* t_i. A *precedence relation* (i, j) is a constraint on the starting time of operation o_j which must be greater or equal to the starting time of the predecessor o_i plus a deterministic given *delay/processing time* w_{ij}. w_{ij} is known in advance and can assume positive, null or negative values. Precedence relations are divided into two sets: *fixed* F and *alternative* A. Alternative precedence relations are partitioned into pairs, i.e., $((i, j), (h, k)) \in A$. There are two dummy operations o_0 and o_n preceding and following all the other operations. Without loss of generality we assume $t_0 = 0$.

Scheduling decisions are in form of disjunctions. That is, one of the two alternative disjunctive arcs has to be selected, i.e., added to the current graph. Given a pair $((i, j), (h, k)) \in A$ in a solution we have either $(i, j) \in S$ or $(h, k) \in S$, where S is the set of selected arcs. A *schedule* is an assignment of starting times $t_0, t_1, \ldots, t_n$ to operations $o_0, o_1, \ldots, o_n$ respectively, such that all fixed precedence relations, and exactly one for each pair of the alternative precedence relations, are satisfied. The goal is to *minimize* a linear combination of the starting time of the operations. This problem can be formulated as a particular *disjunctive program*, i.e. a linear program with logical conditions involving operations "and" ($\wedge$, conjunction) and "or" ($\vee$, disjunction), as in [3] and generalizes disjunctive graph [20] and some of its extension [11].

$$
\begin{array}{lll}
\min & c_0 t_0 + c_1 t_1 + \ldots + c_n t_n & \\
s.t. & t_j - t_i \geq w_{ij} & (i, j) \in F \\
 & (t_j - t_i \geq w_{ij}) \vee (t_k - t_h \geq w_{hk}) & ((i, j), (h, k)) \in A \\
 & t_0 = 0 &
\end{array}
$$

A solution to this problem can be conveniently represented as a temporal network $G = (N, F \cup S)$ where N is a set of nodes of the graph representing events and $F \cup S$ is a set of weighted directed arcs and the starting time t_i is

given by the longest path from o_0 to o_i. This is a general representation that can model most well known scheduling problems.

A stochastic version of a scheduling problem can be represented by considering the weight of the arc as a probability function $f(w_{ij})$. By $f_s(w_{ij})$ we denote the s-th sample of the distribution $f(w_{ij})$.

We use the following notation to represent the uncertainty and the dynamic evolution of a dynamic scheduling problem. Let $\delta = (i, j, f(w), t)$ be an *exogenous change* in the problem where t is the time in which the change in the distribution is revealed to the scheduler, and i, j and $f(w)$ are the starting, ending node and the new probability function for the arc weight, respectively. Let Δ_t be the set of exogenous changes known at time t. Hence, Δ_∞ represents all the exogenous changes that will happen during the execution of the dynamic instance. We use the notation that if there exist two arcs connecting the same pair of nodes i and j then the most recent arc overwrites the old arc. Moreover, observe that, setting the weight to $-\infty$ is equivalent to removing the arc from the graph. Let $\bar{\Delta}_t$ be the set of changes caused by the solver until time t. These internal changes are algorithmic decisions carried out before time t that cannot be undone because they are already active and operation o_i is already in execution, i.e., scheduling decisions such as adding one alternative arc (i, j) to the current selection S.

An instance for the dynamic scheduling problem I is the pair $I = (G, \Delta_\infty)$. An instance contains all the exogenous future changes that will take place in the system. However, when solving a single deterministic scheduling problem the algorithm faces only the information that has been revealed so far since the future information is not available to the solver. The *state of the system* Σ_t at time t is given by the tuple $\Sigma_t = (G, \Delta'_t, t)$ where the set $\Delta'_t = \Delta_t \cup \bar{\Delta}_t$ contains the exogenous changes Δ_t known at time t and the possibly empty set of all the algorithmic changes $\bar{\Delta}_t$ introduced by the algorithm in the previous iterations. Observe that, in general, Σ_t and Σ_{t+1} can lead to two different deterministic scheduling problems even if Δ'_t coincides with Δ'_{t+1}, because moving from time t to time $t + 1$ may cause some operation to enter the frozen period and thus leading to two different deterministic instances.

Let $S((G, \Delta'_t))$ be a solution to a deterministic scheduling problem with the information available at the state $\Sigma_t = (G, \Delta'_t, t)$ and let $z^*((G, \Delta'_t, t), (G, \Delta''_t, t))$ be the value of the objective function obtained when applying the optimal deterministic solution computed for the system state (G, Δ'_t, t) at time t to a possibly different system state (G, Δ''_t, t).

Theorem 2.1 $z^*((G, \Delta'_t, t), (G, \Delta'_t, t)) \leq z^*((G, \Delta''_t, t), (G, \Delta'_t, t))$

Proof Assume $z^*((G, \Delta'_t, t), (G, \Delta'_t, t)) > z^*((G, \Delta''_t, t), (G, \Delta'_t, t))$ then there exists a solution with a lower objective function value than $z^*((G, \Delta'_t, t), (G, \Delta'_t, t))$ which was defined to be optimal. Thus we must reject the assumption.

Observe that the optimal solution $S^*(G, \Delta_\infty)$ of the instance $I = (G, \Delta_\infty)$ may not be obtained by optimally solving all the deterministic problems arising as soon as the uncertainty is revealed $z^*((G, \Delta'_t, t), (G, \Delta'_t, t)), \forall t$. To show this, it is enough to observe that, as in analogy with greedy algorithms, the solver may be forced to take some decision (i.e., adding either arc (i, j) or (h, k) to S) before some relevant information about that decision is actually known. And, once the decision is taken, it cannot be undone.

$z^*((G, \Delta_\infty, \infty), (G, \Delta_\infty, \infty))$ is the optimal value of the *ex-post* optimization, i.e, when all the uncertainty is known in advance, and it is a lower bound for the best attainable solution (Theorem 2.1).

$z^*((G, \Delta_{-\infty}, 0), (G, \Delta_{-\infty}, 0))$ is the optimal value of the *ex-ante* optimization, i.e, when no uncertainty is known in advance. Observe that it does not necessarily give an upper bound for the optimal solution of I. To show this, it is enough to consider a set $\Delta_\infty = \{\delta_t, \delta_{t'}\}$, where the exogenous change δ_t overwrites the original value $f(w_{ij})$ and $\delta_{t'}$ brings it back to the original value. The ex-ante optimal solution therefore coincides with the ex-post optimal solution, while the optimal solution computed at time t may be worse.

3 Literature and applications

Dynamic scheduling problems are an interesting practical extension of classical scheduling problems but they are not deeply investigated [14]. Among the possible approaches, the simulation based are often applied to address the rescheduling problem [17] since they combine classical scheduling problems with widely applied techniques such as the discrete-events simulation.

The recent surveys by Vieira et al. [24] and Aytug et al. [1] review these approaches and applications.

The common simulation-based approach makes it possible to either simulate field disruption or gather them from the real-world application. The architecture of simulation-based approaches is usually composed of a solver (which solves the deterministic scheduling problem) a controller (which decides whether or not the plan has to be updated) and a simulator (which simulates the world and decides the exogenous events). Some papers have proposed general purpose rescheduling frameworks that correspond approximately to the same architectural approach [5].

In their paper, Honkomp et al. [9] models two chemical processes as a State Task Network (STN). The offline scheduling problem is solved using CPLEX to solve a MILP formulation of the problem and with a Bayesian heuristic. Different online strategies are evaluated; the no rescheduling, rescheduling with penalties and full rescheduling. The evaluated objective functions is the deviation from the deterministic objective function.

Cowling and Johansson [6] considered the classical single machine problem $1||\Sigma C_i$ which the SPT rule solves to optimality. Perturbations happen only in the first half of the schedule, and when information arrives three strategies are evaluated: the no rescheduling, a repair strategy and a full reschedule according to shortest processing time (SPT) rule. As evaluated objective functions they consider a linear combination of utility (sum of completion times) and stability.

Pfeiffer et al. [15] developed the same simulation-based framework and addressed three different problems: *i)* a single machine minimizing the average flow time with releases, *ii)* a small flexible job shop problem (5 machines and 8 jobs) and *iii)* industrial application with the objective of minimizing the tardiness. The offline solution is computed using heuristics, and the rescheduling process is triggered either by a fixed rescheduling interval (periodic rescheduling) or when a threshold between the planned and simulated solutions is exceeded. They evaluated the effects of evaluation of machine breakdowns and stochastic processing times. In their tests, the evaluated objective functions takes both

stability and efficiency into consideration.

Bidot et al. [5] consider a dynamic job shop scheduling problem which is solved offline using the ILOG Scheduler. As online strategies they consider a full rescheduling (where the rescheduling process is activated by a trigger) and progressive techniques. The tests are carried out using (10x10) benchmark instances and only efficiency is evaluated in the tests.

Recently, Rasconi et al. [19] consider a full rescheduling and a rescheduling with penalties strategies. The tests are carried out on RCPSP/max benchmark instances and evaluate CPU requirements, frequency of rescheduling, rescheduling success rate as well as stability and efficiency.

The simulation based approach is also applied in this paper. The proposed framework has capabilities similar to other frameworks found in the literature, but to our knowledge it is the only framework that features all of these properties simultaneously:

- Swappable and multiple solvers during the rescheduling process.
- Swappable and multiple objective functions evaluated during the rescheduling process.
- Storing multiple solutions.
- Storing multiple scenarios for comparisons of solvers or solver parameters.
- Decision support through generated visual representations.
- Event driven with the possibility of interfacing directly with a real life application.

In the following section we introduce in details of all the components of the proposed framework.

4 Architecture

This section describes the implementation of the dynamic rescheduling framework proposed in this paper. The framework is implemented in Java and uses the statistical package R version 2.13.1 for generating plots.

Figure 2 presents the framework for dynamic rescheduling, which is composed of a Controller, a Simulator, a Solver, an Instance Writer and Solution Reader module. Moreover the framework should interact with both the Reality (either the real world application or a its simulation) and possibly with a human supervisor.

4.1 Reality module

In laboratory experiments the Reality module is in charge of replicating or interfacing with the real-world. Thus it contains the "real" information about uncertainty and probability distributions and provides it to the Simulator in form of exogenous events Δ_t. Observe that all the other modules are not aware of how such exogenous events are generated. Typical changes to a distribution associated with an uncertain event makes the event become less uncertain as

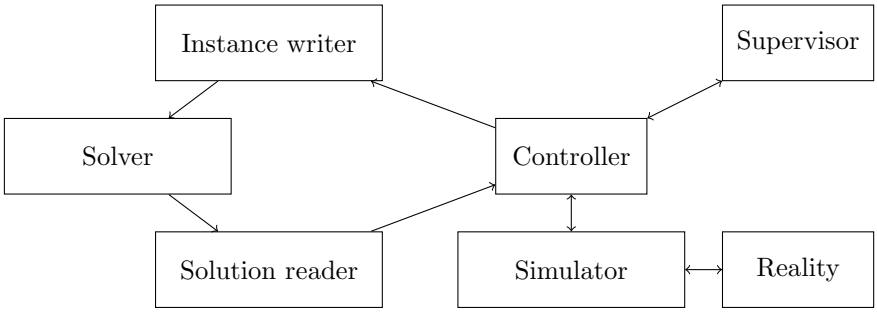

Figure 2: Architecture of the Dynamic Rescheduling Framework.

more information becomes available. This process terminates when all stochasticity is removed and the actual duration of each operation is finally known. Depending on the application the actual duration can be known in advance with respect to the starting time of the operation or while the operation is being processed.

4.2 Simulator module

The Simulator maintains the system state information and updates it as time passes. This is done by keeping a representation of the graph G from the system state Σ_t. The state can change as a result of an *event*. We distinguish three types of events:

End of the rescheduling process. This event is communicated to the Simulator by the Solution Reader module when a new scheduling solution is available.

Exogenous change. This event is communicated by the Reality module signifying a changed distribution of an operations duration.

Time progress. This event is internally generated by the Simulator and consists in updating the system state Σ_t to $\Sigma_{t'}$, i.e., from time t to time t'.

To bootstrap the loop in Figure 2, the Simulator is fed an initial solution to the instance $(G, \Delta_{-\infty})$ computed in the planning phase. From this solution, the initial state $\Sigma_{-\infty} = (G, \Delta'_{-\infty}, -\infty)$ is created.

To analyse the stochasticity of the problem, the Simulator generates a predetermined amount m of Monte Carlo samplings by instantiating the probability distributions associated to the uncertain events. Such distributions in general may be different from the probability distribution used in the Reality module. These are realised as a set of m different scenario graphs $G_s \in G_1 \dots G_m$. G_0 refer to the graph that would be converted to an instance if rescheduling is done. The Simulator then solves a series of longest path problems rooted in the source node to all other nodes in the graphs to obtain start times $t_{i,s}$ for each node in G_s. These values are then used to decide if a rescheduling is warranted, and to analyse the distribution of objectives.

Starting times are calculated using a longest path version of the Bellman-Ford algorithm which is able to detect possible positive length cycles that indicate infeasibility (i.e., an operation preceding itself). When an event Δ_t causes a decreased length of an edge (i, j) in a sample graph G_s a simple check can verify if o_j has a changed start time $t_{j,s}$. If and only if that is the case, the longest path calculations are redone from scratch. If the change causes an increase, o_j is marked as changed in the Bellman-Ford algorithm, and the longest path algorithm is restarted.

4.2.1 Human Interaction

In some applications the Simulator could work with no human intervention or supervision, while in other applications a decision maker could decide whether or not a new rescheduling should be started and supervise the solutions found by the Solver. More often, the presence of a Human Supervisor should be considered. Therefore there is the need to find a suitable way to represent the current solutions of the system and give the supervisor the overwrite possibility (i.e., the chance to change the decisions taken by the automated system).

A possible way to provide information on the solution to the supervisor is to make use of stochastic Gantt charts, as the one shown in Figure 3. Such plots are based on predictive Gantt charts introduced in [2] and are useful for displaying stochastic schedules. They are drawn as regular Gantt charts, but the boxes signifying an event occurring with certainty at a given time, have been replaced by a shape representing the approximate likelihood of an event happening at a given time.

The height of the row representing each machine is not explicitly drawn, but can be seen as the topmost and bottommost part of the operations belonging to that machine. Figure 3 is a solution to a problem with two machines and two jobs ($J1$,$J2$) each with two operations of which one is shown. The red vertical line at time 350 intersects the shapes representing two operations labelled $J0, 0$ and $J1, 1$ respectively. The red line can be divided into three parts:

The area below the shape representing $J1, 1$ represents the chance that $J1, 1$ has finished at time 350, the area intersecting represents the chance that the operation $J1, 1$ is currently running. The area above represents the chance that $J1, 1$ has not started at time 350, as this area is occupied by the shape representing $J0, 0$, we must conclude that the machine is occupied by the previous operation $(J0, 0)$ in these cases.

Each shape is drawn based on linear interpolation between points drawn for the following inputs to the quantile function: .01, .025, .05, .1, .2, .3, .4, .5, .6, .7, .8, .9, .95, .975 and .99. Note that the first and the last percentile are not represented to avoid visual artefacts on long tailed and unbounded distributions.

If other visualizations are required, the supervisor can request density plots of timings of any operation, or for an objective function. This can be done for multiple solutions, enabling comparisons such as the one depicted in Figure 4.

4.3 Controller module

The Controller implements the rescheduling policy, i.e., it decides when the Solver should be invoked. This is done by a set of trigger conditions and when the condition of a Trigger is satisfied then the Controller calls the Solver for a

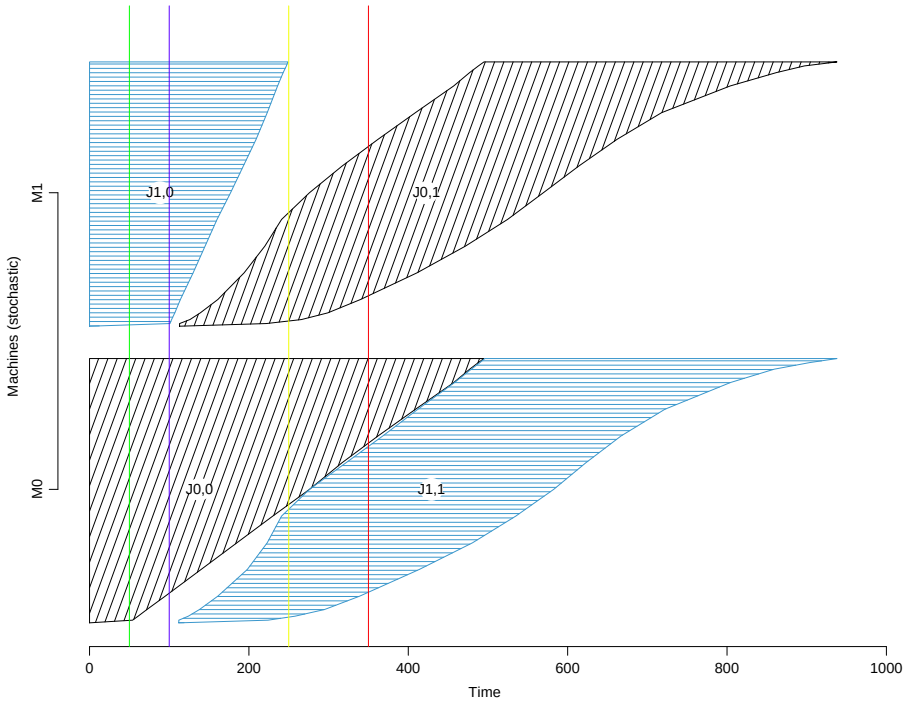

Figure 3: A small stochastic Gantt chart showing a single machine in a solution to a two machine job shop problem.

reoptimization. A trigger is a function $T(\Sigma_t, t, t_{last}, \Delta_t \setminus \Delta_{t_{last}})$ that maps the current state of the system Σ_t, the time t, the last time a solver was run t_{last}, and the events $\Delta_t \setminus \Delta_{t_{last}}$ into a boolean value.

Triggers can be grouped by type according to the data that trigger them. *Deterministic triggers* are triggered by changes in the deterministic data potentially passed on to the solver G_0, while *stochastic triggers* work on the Monte Carlo trials on the graph representing the system state $G_1 \dots G_m$.

Though the dynamic rescheduling framework is not limited to these, the following types describe some of the most common deterministic triggers:

Changed makespan is triggering when the longest path from source to the sink node changes.

Lateness of an operation is triggered when the longest path from source node to the corresponding node plus the duration of the operation exceeds the due date of the operation.

Infeasibility of the solution originally provided by the solver.

Sum of lateness exceeding a given value.

Stochastic triggers are based on a boolean expression that can be evaluated on each of the sample graphs $G_1 \dots G_m$. They trigger when the boolean condition C is satisfied with a given percent chance with confidence p. For example the reoptimization can be called when, with 95% confidence, there is at least 10% chance of worsening the objective function.

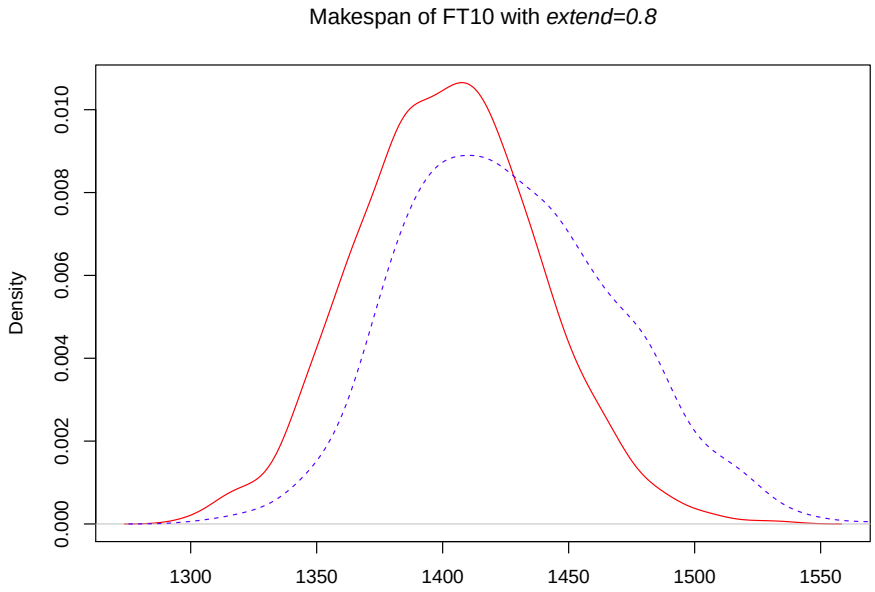

Figure 4: Density plot of the makespan for two solutions to the FT10 job shop instance with added stochasticity ($extend = 0.8$). The red solid line represents a superior solution to the blue dotted line given that we want to minimize the makespan. This can be seen as a shift toward lower makespans (left on the x axis).

Custom triggers can be implemented by adhering to a provided interface.

4.4 Solution Reader and Instance Writer modules

The Solution Reader and Instance Writer modules must be implemented separately for each Solver that is to be used with the framework.

The Solution Reader module must generate an temporal network from the instance file and the solution produced by the deterministic solver.

The Instance Writer module is responsible for creating a deterministic instance based on the simulation of the current system state Σ_t. The primary challenge when implementing the Instance Writer module is the progressing time during simulation. The Instance Writer should enforce that the starting time of locked events should never be modified by the Solver. Common ways of doing this includes leaving them out of the instance and mimicking their presence by enforcing release times on other operations, or alternatively introduce deadlines and release times that fixes the original operations in time. The choice is determined by the ability of the solver to handle the constraints needed.

4.5 Solver module

The Solver is the optimization core of the rescheduling system. The proposed dynamic rescheduling framework is Solver independent. Both exact and heuristic solvers can be interfaced with the framework. If the solver uses a deterministic algorithm, it is not going to be invoked twice on the same instance. In Section 5 we show how a dispatching rule yielding the optimal solution or a truncated commercial solver can both be used as Solvers within the framework.

4.6 Parameters and Input

The framework requires the following application specific parameters:

Knowledge parameter $k \in (-\infty; 1]$ is used when the framework is not connected to a real life application. It describes when the actual duration of an operation is revealed to the framework.

A negative integer value denotes the amount of time units before the operation is scheduled to begin in the current schedule. A value in the interval $[0; 1]$ denotes the percentage completion an operation must be at before the process time becomes known to the framework. $k = 0$ thus corresponds to generating the event when an operation starts, $k = 1$ when it ends and $k = -1$ the time unit before the operation is scheduled to start.

In case of negative k, gained information is retained even if the operation is rescheduled thereafter

Frozen period the amount of time ahead of the starting time of an operation, where it should be considered locked in the schedule.

Allowed CPU time the maximum amount of time the solver is allowed to use for reoptimization.

Frequency parameter $f \in [1; \infty]$ is a lower bound in time units between runs of the solver.

Finally the framework must be given information detailing how and when to reschedule.

Number of samples $n \in [0; \infty)$ determines the number of graphs $G_1 \dots G_n$ created by Monte Carlo samplings.

Sampling strategy $g \in [0; 1]$ is used to generate a deterministic instance for the solver based on the stochastic instance under consideration. For bounded distributions this will be treated as reverse lookup in the cumulative distribution function. Thus $g = 0$ corresponds to the shortest possible duration, and $g = 1$ corresponds to the longest possible duration and $g = 0.5$ is the median of the distribution. For unbounded distributions, a maximum value must be enforced to avoid generating infinite values.

Triggers must be implemented beforehand, and can be enabled by a parameter.

Rolling Horizons can be handled if implemented in the instance reader and writer, and the solver must be able to solve the partial instance.

5 Computational Experiments

The computational experiments reported in this paper are based on two different scheduling applications. Namely we consider the Single Machine Weighted Completion Time problem and the academic Job-Shop Scheduling problem. In the results we show the deviation from the optimal value of the ex-post optimization $z^*((G, \Delta, \infty), (G, \Delta, \infty))$.

To illustrate the functionality of the framework, we perform an exploratory analysis of each of the two problems, followed by the analysis of an artificial test case. All the test have been carried out on a Intel Core i5 CPU 650 @ 3.20GHz processor.

5.1 Single Machine Weighted Completion Time (SMWCTP)

The single machine problem we consider is the classical $1||\sum w_i C_i$ single machine problem with the objective of minimizing the sum of the weighted completion time of each job. Given a job i, let p_i and w_i denote its processing time and its weight, respectively. Given a solution, we denote the completion time of job i with C_i. The problem is known to be polynomially solvable [22] by applying the Weighted Shortest Processing Time (WSPT) rule, that is to sequence jobs according to their non-decreasing ratios p_i/w_i.

We consider 10 randomly generated instances of SMWCTP, each having a number of jobs randomly selected as $n \in [10; 100]$. Each operation o_i has a weight $w_i \in [1; 100]$ and a processing time $p_i \in [1; 100]$. All values are uniformly likely in each interval. We then create stochastic instances by changing all of the process times into a uniform distribution ranging from its original duration $p_{i,j}$ to $p_{i,j} + p_{i,j} \cdot \mathit{extend}$ for a given value of the *extend* parameter. In our

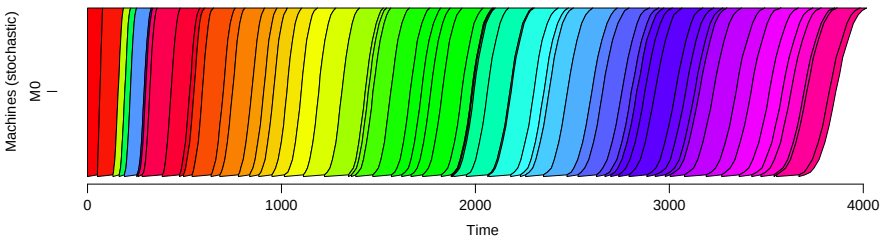

Figure 5: A stochastic Gantt chart for a SMWCTP instance with 60 jobs.

test, we set $extend \in \{0.1, 0.2, \ldots 0.9, 1.0\}$. For each of the 100 stochastic instances, we create 4 *sample instances* ($sample = 4$) by sampling each of the distributions associated with the processing times. Sample instances represent the actual realization of a given stochastic instance, and are used for generating consistent processing times across multiple runs of the framework. Therefore each stochastic instance is evaluated four times per run of the framework, one for each sample.

During the experiments we tested 15 different "test case" configurations (i.e., parameters that are influenced by the real-world application) by setting the knowledge parameter $k \in \{-400, -200, -100, -10, 0\}$ and the frequency parameter $f \in \{1, 10, 100\}$. Observe that there is no need to test cases with $k \geq 0$ since it indicates that information about an operations process time is revealed during the operations execution when the operation is already frozen. For SMWCT the residual problem is just sorting the remaining operations using the WSPT rule corresponding to the optimal ex-ante solution.

The sampling strategies were set to $g \in \{0, 0.25, 0.5, 0.75, 1\}$, furthermore we used 3 Triggers:

$T_{objective}$ Trigger when the objective function changes.

$T_{change10}$ Trigger when the sum of absolute values of changes in the process times exceeds 10 time units.

$T_{change100}$ Trigger when the sum of absolute values of changes in the process times exceeds 100.

For each of the triggers, the f parameter provides a lower bound on the time between rescheduling. Thus if $f = 10$ and $T_{objective}$ detects a changed objective at time $t = 1$ and $t = 2$, 10 time units will still pass between reschedules.

Overall, 10 instances of SMWCTP have been extended into 100 stochastic instances, which are subsequently solved in 4 different instantiations (using the 4 samples). 15 test cases and 15 configuration have been tested leading to 90000 executions of the framework. In Figure 5 a stochastic Gantt for a SMWCTP is shown.

5.1.1 Results

Figure 6 shows the average quality of the solutions obtained, as a function of the sampling strategy g for set values of k and f respectively. Figure 6 (a) shows that guessing a low process time p_i seems favorable for low values of f, i.e.,

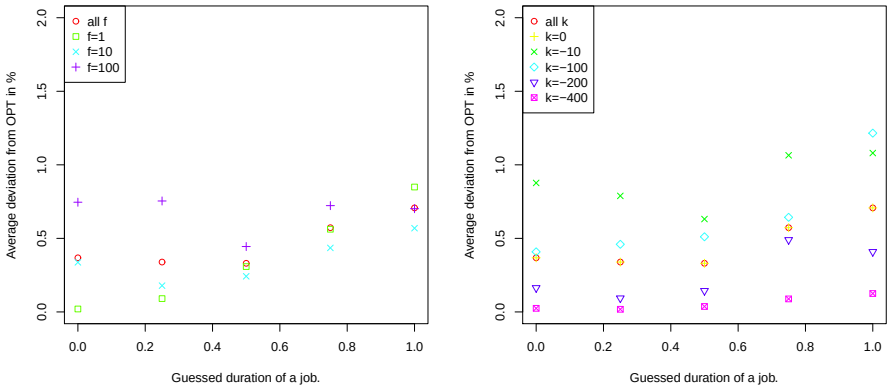

Figure 6: Solution quality as a function of the guess parameter g for the single machine weighted completion time with the $T_{objective}$ trigger. The left plot averaged over all k and the right plot averaged over all f.

Table 1: The average number of reschedules and the average deviation from the optimal solution when applying the three triggers T_1, $T_{change10}$ and $T_{change100}$.

	$T_{objective}$	$T_{change10}$	$T_{change100}$
Reschedules	31.0	19.7	4.8
Average deviation	0.41%	2.26%	38.89%

when rescheduling frequently, while less frequent optimizations favor g values closer to 0.5 and thus guesses closer to the expected value of p_i. The time at which the framework becomes aware of actual durations k (Figure 6 (b)) shows a tendency to favor lower values of g.

Table 1 shows that rescheduling at every change of a duration ($T_{objective}$) when the f parameter allows leads to solutions that deviate from the optimal offline solution by 0.41%, when averaged over all values of k, f, *extend* and g. If an exogenous cost is associated with each rescheduling, rescheduling only if the change in duration exceeds 10 time units ($T_{change10}$), becomes interesting as it saves 11.3 reschedules on average and obtaining solutions with 2.26% deviation from the ex-post optimum on average. The exogenous cost must be significant to justify using $T_{change100}$, as the average deviation increases to 38.89%.

Table 2 shows that the difference between rescheduling every one and ten time units is small compared to other sources of increase in solution quality.

Table 2: Average deviation in weighted completion times for different values of rescheduling frequency f with the trigger $T_{objective}$.

f	1	10	100
Average deviation	0.26%	0.38%	0.59%

Table 3: Average deviation in weighted completion times for values of k.

k	-400	-200	-100	-10
Average deviation	0.06%	0.22%	0.54%	0.81%

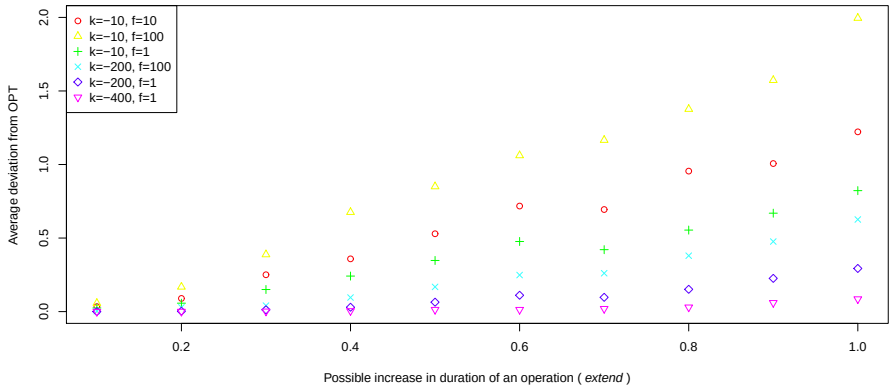

Figure 7: Solution quality as a function of the *extend* parameter for set values of k and f with the trigger $T_{objective}$.

Even a rescheduling every 100 time units provide comparable quality.

While, as expected, Table 3 shows an increase in deviation from offline optimum as k approaches 0.

Figure 7 shows effect of increased stochasticity on the solution quality. Enforcing a pause of 100 time units between each rescheduling ($f = 100$) comes at a small cost when $k = -200$, but the cost of increasing f rises when $k = 10$, because the framework sometimes gets events but is unable to react on the due to a recent rescheduling when $f = 100$.

5.1.2 Test cases analysis

Observe that the stochasticity *extend*, the knowledge k and the frequency f parameters are usually determined by the application. For this artificial test case we assume that $extend = 0.8$, $k = -10$ and $f = 10$, respectively.

The choices left to the managers are which trigger and sampling strategy to adopt. Assume that possible triggers choices are $T_{objective}$ and $T_{change10}$ and $g \in \{0, 0.25, 0.5, 0.75, 1\}$, leading to 10 possible configurations.

By running the proposed dynamic rescheduling framework it turns out that the best configuration is $T_{objective}$ and $g = 0.25$ since it provides the smaller deviation from the ex post optimum (as shown by Table 4).

If there is an exogenous cost associated with rescheduling, the problem becomes multiobjective, as $T_{objective}$ always causes more than 40 reschedules while $T_{change10}$ causes significantly less. Five solutions are non-dominated (in bold in Table 4): $g \in \{0.25, 0.5\}$ for both triggers, and $g = 1.0$ for $T_{objective}$.

Table 4: Average deviation from optimal ex-post solution and average number of reschedulings for $extend = 0.8$, $k = -10$ and $f = 10$.

	$g = 0.0$	$g = 0.25$	$g = 0.5$	$g = 0.75$	$g = 1.0$
$T_{objective}$ (dev.)	2.00%	**0.95%**	**0.96%**	1.35%	**1.72%**
$T_{objective}$ (# resch.)	40.9	**41.5**	**41.3**	43.8	**40.6**
$T_{change10}$ (dev.)	2.37%	**2.11%**	**2.94%**	5.12%	3.36%
$T_{change10}$ (# resch.)	32.2	**26.1**	**24.3**	26.7	31.5

5.2 Job shop scheduling

The classical job shop scheduling problem ($J||C_{max}$) is one of the most studied NP-hard problems [16]. As solver we use a standard constraint programming model for the classical job shop scheduling problem and it has been implemented using IBM ILOG CP Optimizer version 12.2. Since solving to optimality may take prohibitively long, the allowed CPU time is set to 60 seconds and the solver uses a single worker thread.

In order to generate our tests we consider the well known 10-job 10-machine benchmark instance FT10 [13] and we create stochastic instances by changing all of the process times into a uniform distribution ranging from its original duration $p_{i,j}$ to $p_{i,j} + p_{i,j} \cdot extend$ ($extend \in \{0.1, 0.2, \ldots 0.9, 1.0\}$). For each of the stochastic instances, we create 4 sample instances by sampling each of the distributions associated with the process times. Thus we generate 40 sampled stochastic instances in total.

As test cases configuration we use $k \in \{-400, -200, -100, -10, 0, 0.5, 1\}$ and $f \in \{1, 10, 100\}$, leading to 21 different application settings.

20 configurations of the framework, obtained by setting $g \in \{0, 0.25, 0.5, 0.75, 1\}$ and 4 Triggers, have been tested. The 4 Triggers are:

$T_{objective}$ Trigger when the objective function changes.

$T_{frequency}$ Trigger as soon as the f parameter allows, and any distribution has changed since last optimization.

$T_{change10}$ Trigger when the sum of absolute values of changes in the process times exceeds 10 time units.

$T_{change100}$ Trigger when the sum of absolute values of changes in the process times exceeds 100.

Overall $40 \cdot 21 \cdot 20 = 16800$ runs of the framework have been executed.

5.2.1 Results

In Figure 8 we show the behavior of the ex ante solution and 4 test cases configuration ($(k = 0, f = 10)$, $(k = -200, f = 1)$, $(k = -400, f = 1)$ and $(k = 1.0, f = 100)$) among the 21 considered as the stochasticity influence increases ($extend$).

Figure 8 indicates that for values of $extend$ below 0.2 there seems to be no benefit to rescheduling on average. For higher values of the extend parameter

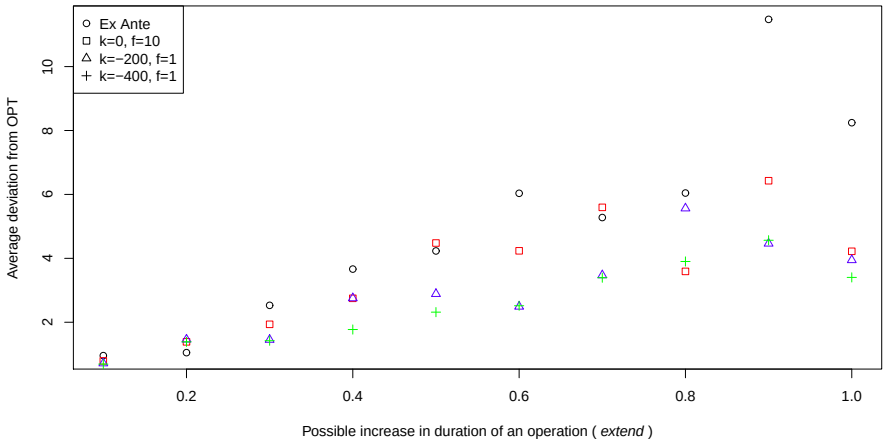

Figure 8: Gap between the ex-ante and ex-post solution compared with selected results by the framework.

Table 5: Solution quality as a function of rescheduling frequency f for the job shop problem.

f	1	10	100
Average deviation	3.75%	3.83%	4.33%

there is a gap that can actually be exploited by allowing dynamic rescheduling. Figure 8 also indicates that an earlier knowledge of events (i.e., lower values of k) is beneficial. Besides, even if the actual duration is only known when the operation starts ($k = 0$) and the rescheduling is not too frequent ($f = 10$) still allows rescheduling to outperform the ex ante solution. The worst configuration tested for k and f ($k = 1.0, f = 100$) seems to have no consistent improvement over the ex-ante solution.

To illustrate the impact of the extend parameter we show in Figure 9 and 10 the difference in ex-ante and ex-post solutions for $extend = 0.2$ and 1.0 respectively. Observe that, for $extend = 0.2$ the two solutions coincides thus the problem is deterministic enough to make rescheduling actions unnecessary. However, for higher values of the *extend* parameter the two optimal solutions differs, thus making rescheduling beneficial.

A clear benefit of obtaining knowledge earlier is expected. But when evaluating whether such information is worth the cost of obtaining it, the magnitudes of these benefits needs to be established. Figure 11 shows an approximately linear increase in solution cost as the k parameter increases. Contrary to the single machine problem, we observe little extra penalty for jobs being locked when the information about their durations are revealed ($k \geq 0$). This is due to other potentially affected operations possibly being unfrozen at that point.

Table 5 indicates the magnitude of the benefit of allowing frequent reschedules, but the data is averaged over all values for *extend*, g, s and k. On average,

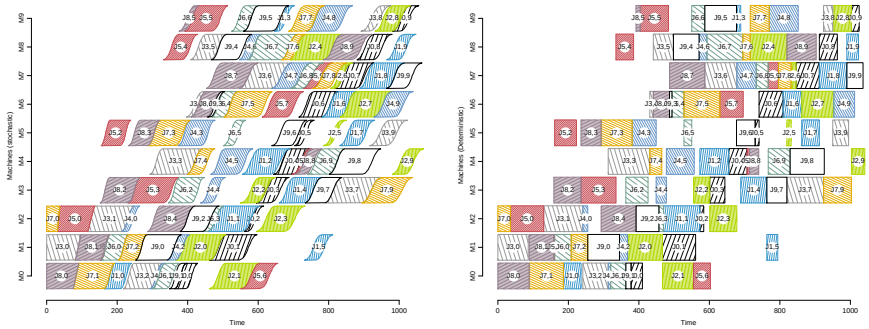

Figure 9: A stochastic Gantt chart for an ex-ante solution to a job shop instance for $g = 0.5$ and $extend = 0.2$ (top), and a deterministic Gantt chart for the ex-post solution to the same problem. Note the conservation of ordering of operations.

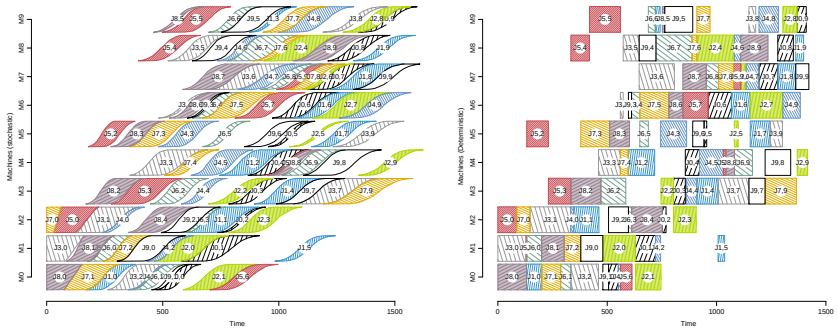

Figure 10: A stochastic Gantt chart for an ex-ante solution to a job shop instance for $g = 0.5$ and $extend = 1.0$ (top), and a deterministic Gantt chart for the ex-post solution to the same problem. Note that the order of operations are not conserved.

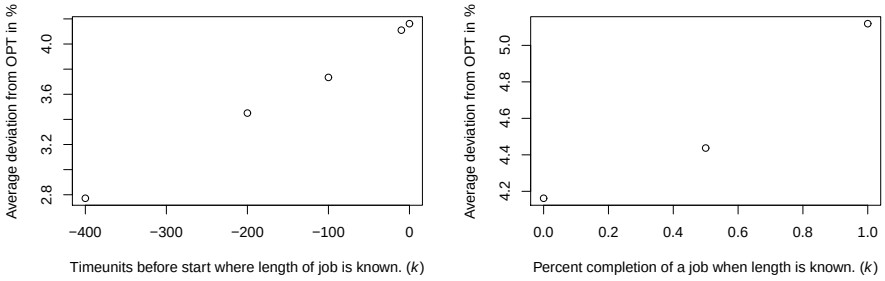

Figure 11: Solution quality as a function of the knowledge parameter k.

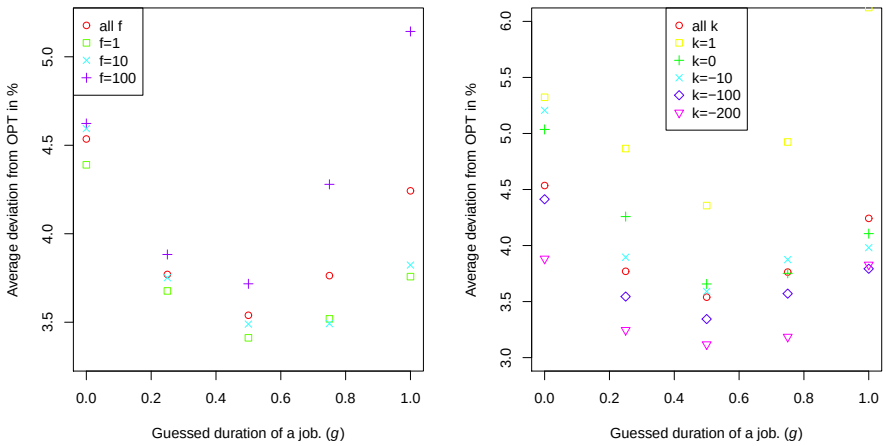

Figure 12: Solution quality as a function of the guess parameter g for JSP.

Table 6: The average number of reschedules and the solution quality when applying the four triggers $T_{objective}$, $T_{frequency}$, $T_{change10}$ and $T_{change100}$.

	$T_{objective}$	$T_{frequency}$	$T_{change10}$	$T_{change100}$
Avg. Reschedules	26.1	73.4	50.0	14.5
Average deviation	3.78%	3.75%	3.75%	3.95%

lower values for the frequency lead to better solutions.

To evaluate g's influence on the solution quality, we plot it for different values of f and k respectively in Figure 12. Contrary to the single machine problem we observe $g = 0.5$ results in the best results. This corresponds to passing expected values of probability functions to the solver.

If the job shop problem has exogenous costs associated with rescheduling fewer reschedules can be obtained through the use of triggers. Table 6 shows that postponing rescheduling until process times have changed by at least 10 units ($T_{change10}$), on average produces solutions of the same quality as rescheduling when a change is encountered and the f parameter allows ($T_{frequency}$) while using significantly fewer reschedules. Rescheduling only when events have affected the objective of the solution given by the last run of the solver ($T_{objective}$), yields slightly worse solutions but with a significant reduction in the number of reschedules necessary. Finally requiring an accumulated change in the process times of 100 time units provides reasonable results at even fewer reschedules.

During the execution, a supervisor can monitor the stochastic Gantt chart, or the development of the objective function as shown in Figure 13. In the plot the estimated solution quality is the makespan as computed by the deterministic solver (red marks). Recall that the solver ignores uncertainties and the actual solution quality as computed by the simulator module reflects the real makespan (blue marks). The gap between the two typically starts out being large and tends to decrease over time as the schedule is executed and uncertainties are

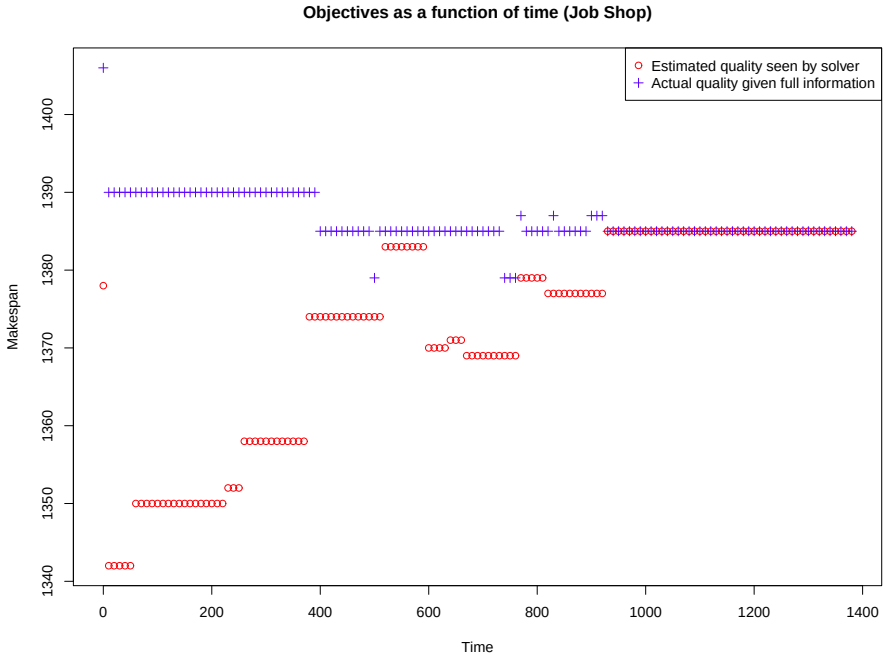

Figure 13: Progression of the objective function for the job shop problem with $extend \in \{0.5, 0.75, 1.0\}, k = -400, f = 10, g = 0.5$. Jumps in either graph could be due to a new solution being considered, but the estimated quality can change due to events changing distributions.

revealed. At the end of the execution the two values must coincide. In Figure 13 the attained makespan value is 1385. Note that the actual solution quality generally improves over time, but that a solution with a better makespan (1379) was obtained four times, but discarded due to lacking information. In a real life application, the full information is not known in advance and thus cannot be plotted. In this case another scenario (e.g. worst case, best case) can be monitored instead.

5.2.2 Test cases analysis

For this test case we assume that $extend \in \{0.5, 0.75, 1.0\}, k = -100$ and $f = 10$ but with the option of either setting $k = -200$ or $f = 1$ at a cost.

It is clear from Figure 14 that getting information earlier by setting $k = -200$

Table 7: The average number of reschedules when applying the four triggers $T_{objective}$, $T_{frequency}$, $T_{change10}$ and $T_{change100}$ for $extend \in \{0.5, 0.75, 1.0\}$, $f = 10$ and $k \in \{-100, -200\}$.

	$T_{objective}$	$T_{frequency}$	$T_{change10}$	$T_{change100}$
Avg. Reschedules	26.2	58.9	44.8	14.5

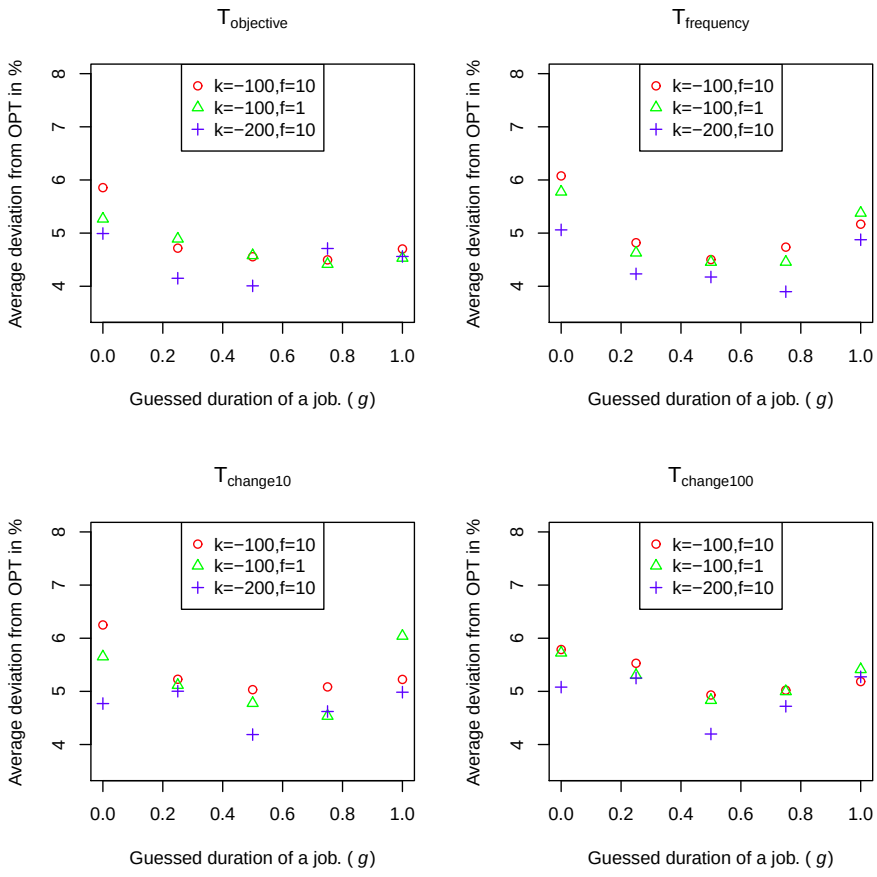

Figure 14: Average deviation from optimal ex-post solution as a function of g for each trigger for $extend \in \{0.5, 0.75, 1.0\}$.

is worth more on average than allowing rescheduling in each time step ($f = 1$), independently of the trigger utilized. The general quality of the solution appears to have no clear correlation with the trigger used, which further substantiates the hypothesis that the rescheduling is done often enough to take advantage of the available information.

Given that the triggers appear to give solutions of comparative quality, a decision maker would likely choose $T_{objective}$ or $T_{change100}$ to minimize the number of reschedules performed (Table 7). For both of these triggers, setting $g = 0.5$ and $k = -200$ yields the best results: 2.96% and 3.07% respectively. To calculate if lowering the value of k is worth it, the benefit of 0.29% and 0.39% for $T_{objective}$ and $T_{change100}$ respectively must be compared to any exogenous costs.

5.3 Comments

From our tests it turns out that having information earlier is more important than frequent reschedules.

For the JSP the most important factor turns out to be the magnitude of the uncertainties. When the problem allows only small changes (less than 20%) in the processing times there is no need of dynamic rescheduling since the ex ante solutions are already good enough. As the uncertainty increases the need of allowing rescheduling increases too.

When coming to the influence of the parameter setting of the framework, we observe a different behavior for the two problems. In the SMWCTP the choice of the trigger is influential, and while giving different results the sampling strategies do not radically change the quality of the solutions if $f \neq 1$. Whereas in the JSP the choice of the trigger does not seem to have a strong influence on the solution quality while the number of reschedules changes a lot. When considering the sampling strategy g the best results are obtained by setting it to the expected (median) values of the processing times.

6 Conclusions and future research

In this paper we introduced a simulation-based framework for addressing dynamic rescheduling problems.

The functionality of this framework has been shown by considering two different dynamic scheduling problems: A single machine and a job shop scheduling problem.

The framework has in this paper been used to investigate some issues: *(i)* comparing the relative performance of different solver configurations and the absolute performance with ex-ante/ex-post optimization; *(ii)* evaluating the robustness of the ex ante solution to uncertain distributions; *(iii)* the influence of triggers on the number of times rescheduling is performed, and the effect on the solution quality.

Future developments include the application of this framework to specific problems and use of spare CPU time of the Solver to perform alternative future event branch prediction and the development of better strategies to produce robust solutions.

Acknowledgements

The authors would like to thank Geoff Robinson and Andreas Ernst for providing R code of a preliminary version of the stochastic Gantt charts.

References

[1] Haldun Aytug, Mark A. Lawley, Kenneth McKay, Shantha Mohan, and Reha Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161(1):86 – 110, 2005.

[2] Kenneth R. Baker and Dan Trietsch. *Principles of Sequencing and Scheduling*. Wiley Publishing, 2009.

[3] E. Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 5:3 – 51, 1979.

[4] A. Ben-Tal and A. Nemirovski. Robust optimization – methodology and applications. *Mathematical Programming*, 92:453 – 480, 2002.

[5] Julien Bidot, Thierry Vidal, Philippe Laborie, and J. Beck. A theoretic and practical framework for scheduling in a stochastic environment. *Journal of Scheduling*, 12:315–344, 2009.

[6] Peter Cowling and Marcus Johansson. Using real time information for effective dynamic scheduling. *European Journal of Operational Research*, 139(2):230 – 244, 2002.

[7] A. D'Ariano, D. Pacciarelli, and M. Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183:643–657, 2007.

[8] Salah E. Elmaghraby. Activity nets: A guided tour through some recent developments. *European Journal of Operational Research*, 82:383 – 408, 1995.

[9] S. J. Honkomp, L. Mockus, and G. V. Reklaitis. A framework for schedule evaluation with processing uncertainty. *Computers & Chemical Engineering*, 23(4 - 5):595 – 609, 1999.

[10] B.L. Maccarthy and Jiyin Liu. Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, 31(1):59 – 79, 1993.

[11] Alessandro Mascis and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498 – 517, 2002.

[12] Kenneth N. McKay and Vincent C.S. Wiers. Unifying the theory and practice of production scheduling. *Journal of Manufacturing Systems*, 18(4):241 – 255, 1999.

[13] J.F. Muth and G.L. Thompson. *Industrial Scheduling*. Kluver Academic, Amsterdam, 1963.

[14] Djamila Ouelhadj and Sanja Petrovic. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12:417 – 431, 2009.

[15] András Pfeiffer, Botond Kádár, and László Monostori. Stability-oriented evaluation of rescheduling strategies, by using simulation. *Computers in Industry*, 58(7):630 – 643, 2007.

[16] M. Pinedo. *Scheduling – theory, algorithms and systems. Int. Series in Industrial and System Engineering*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[17] R Ramasesh. Dynamic job shop scheduling: A survey of simulation research. *Omega*, 18(1):43 – 57, 1990.

[18] R. Rangsaritratsameea, William G. Ferrell, and Mary Beth Kurzb. Dynamic rescheduling that simultaneously considers efficiency and stability. *Computers & Industrial Engineering*, 46:1–15, 2004.

[19] Riccardo Rasconi, Amedeo Cesta, and Nicola Policella. Validating scheduling approaches against executional uncertainty. *Journal of Intelligent Manufacturing*, 21:49–64, 2010.

[20] B. Roy and B. Sussman. Les problèm dordonnancement avec contraintes disjonctives. Technical report, Note DS No. 9bis. Paris: SEMA, 1964.

[21] A. Ruszczynski and A. Shapiro. *Stochastic Programming. In: Handbooks in Operations Research and Management Science*. Elsevier, Amsterdam, 2003.

[22] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[23] A.T. Unal, R. Uzsoy, and A.S. Kiran. Rescheduling on a single machine with part-type dependent setup times and deadlines. *Annals of Operations Research*, 70:93–113, 1997.

[24] Guilherme E. Vieira, Jeffrey W. Herrmann, and Edward Lin. Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling*, 6:39–62, 2003.