

-

CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS



Daniel Slunečko

Scheduling of F-shaped Tasks with Replication to Maximize Execution Probability

Department of Control Engineering

Thesis supervisor: **Ing. Antonín Novák**

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne.....

.....

DIPLOMA THESIS AGREEMENT

Student: Slunečko Daniel

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: Scheduling of F-shaped Tasks with Replication to

Guidelines:

The thesis aims at investigating and solving the scheduling problem that arises in time-triggered systems with mixed-critical activities. Commonly, such systems are safety-critical and typically embed problems such as message scheduling or the scheduling of computational tasks. The common link of those problems is how to schedule activities with uncertain processing time such that the critical ones are executed with a high probability.


1. Define the problem, study related literature and investigate known solutions.
2. Formulate the problem as a scheduling problem with F-shaped tasks considering the replication and temporal constraints.
3. Propose and implement an algorithm that maximizes the execution probability of scheduled tasks.
4. Evaluate the proposed solution and discuss the influence of the key parameters.

Bibliography/Sources:

- [1] Seddik, Y., Hanzalek, Z. - Match-up scheduling of mixed-criticality jobs: maximizing the probability of jobs execution. (under review in European Journal of Operational Research). 2016
- [2] Hanzálek, Z. - Tunys, T. - Sucha, P.: An Analysis of the Non-preemptive Mixed-criticality Match-up Scheduling Problem, Journal of Scheduling, Volume 19, Issue 5, pp. 601?607, doi: 10.1007/s10951-016-0468-y
- [3] Alain Girault, Hamoudi Kalla, Mihaela Sighireanu, and Yves Sorel. An algorithm for automatically obtaining distributed and fault-tolerant static schedules. In Proceeding of International Conference on Dependable Systems and Networks, pages 165?190. IEEE, 2003.
- [4] Kenli Li, Xiaoyong Tang, Bharadwaj Veeravalli, and Keqin Li. Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. IEEE Transactions on Computers, 64(1):191?204, 2015.

Diploma Thesis Supervisor: Ing. Antonín Novák

Valid until the end of the summer semester of academic year 2017/2018



prof. Dr. Michal Pěchouček, MSc.
Head of Department



prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 19, 2017

Acknowledgements

My sincere thanks go to my supervisor who guided me throughout the entire work on this thesis. For all the time he was willing to invest in reading new versions of the text again and again and for providing useful insights.

I also want to thank to my family, for their support and prayers in moments of despair during my studies.

Last, but not least, I want to thank to my friends who encouraged me when I felt down and to Eduard Rindt, whose hard work always inspired me.

Abstract

In this work, we focus on a problem of scheduling F-shaped tasks with replication to maximize probability of execution. We introduce replication to the domain of F-shape scheduling. We extend the existing scheduling model so that it is useful for multiprocessor environment with replicated F-shapes and time lags. We propose a way for computation of execution probability in such systems. We also propose two heuristic scheduling algorithms for this problem. The algorithms are evaluated on instances with various setup and the influence of replication on the quality of solutions is demonstrated.

Abstrakt

V rámci této práce jsme se zaměřili na problém rozvrhování F-tvarých úloh s replikacemi pro navýšení pravděpodobnosti provedení těchto úloh. Nově jsme zavedli koncept replikace v oboru rozvrhování F-tvarých úloh a rozšířili jsme stávající model tak, aby byl použitelný i pro problémy s replikovanými F-tvarými úlohami v prostředí s časovými hranami. Navrhli jsme způsob výpočtu pravděpodobnosti provedení jednotlivých úloh. Zároveň jsme navrhli a implementovali dva heuristické algoritmy pro rozvrhování takovýchto problémů. Oba algoritmy jsme otestovali na instancích s různými parametry a ukázali jsme vliv replikace na kvalitu výsledných rozvrhů.

Contents

1	Introduction	1
1.1	Related Work	2
1.2	Contribution	5
1.3	Outline	5
2	Problem Statement	6
2.1	Job as F-shape	10
2.2	Online Execution of the Schedule	12
2.3	Replication	13
2.4	Summary	17
2.5	Problem Complexity	17
2.6	Blanket Definition	18
2.7	Example of Task Instance	19
3	Time Lags	21
3.1	Interpretation of Time Lags	21
3.2	Observations and Notes	24
4	Probability of Job Execution	25
4.1	Without Job Replication	25
4.2	With Job Replication	25
4.3	Zero Time Lags and Job Replication	27
5	Scheduling Algorithms	33
5.1	Mixed-Integer Linear Program	33
5.2	Simulated Annealing	35
5.3	Iterative Resource Scheduling Algorithm	38
5.4	Implementation	41
6	Computational Experiments	43
6.1	Criterion Computation Given a Schedule	43
6.2	Instances of $P temp, mc = 3, mu \sum_i w_i P_i$	44
6.3	Success Rate	45
6.4	Time Requirements	45
6.5	Impact of Replication on Criterial Function	46
7	Conclusion	52
	Appendix A CD Content	55
	Appendix B List of abbreviations	57

CONTENTS

List of Figures

1	Example of a schedule with F-shapes on two processing units	2
2	Visualization of some job J_1 in a form of an F-shape with three possible processing times (p_1^1, p_1^2, p_1^3)	11
3	Example of a feasible schedule with 3 jobs on one processing unit.	11
4	Schedule consisting of F-shapes	13
5	Possible scenarios in a schedule with five F-shapes on a single processing unit. The dotted line represents the executing level function.	14
6	A "trap" example. In this schedule, job J_2 has nonempty coverage, but is replicated so that it is always executed.	16
7	Blanket example	18
8	An instance solution and graph of time lags	20
9	An example with extended blanket of size equal to n . Dotted lines represent zero time lags.	29
10	Time required for criterion computation over multiple instances by brute force	43
11	Time required for criterion computation over multiple instances using blankets	44

LIST OF FIGURES

1 Introduction

Nowadays there is a wide spectrum of scheduling problems that are classified by their specific features such as number of processing units, types of temporal and precedence constraints used, optimization criteria and other. We focus on a specific case of non-preemptive scheduling problem with multiple processing units and relative temporal constraints. Namely, this thesis focuses on scheduling of F-shaped tasks¹ with replication. A motivation for the problem at hand can be the following situation.

Let us imagine an example taken from autonomous driving. There, a heterogeneous computational platform is processing a large number of visual data. Part of the problem is to decide which CPU will process which task. Tasks represent a computational load, such as road sign detection, pedestrian detection, collision avoidance etc. Furthermore, the exact processing time of tasks is not known in advance due to the nature of detection algorithms and their low-level implementation aspects. Therefore, it might happen that some task needs more time to complete. However, if it is a safety-critical task (pedestrian detection), we allow its prolongation, and as a trade-off, we will skip some less critical task (reading out a proximity sensor).

Similarly, semi-autonomous drones and other systems that are supposed to guarantee safety, yet requiring to process jobs with non-deterministic processing times, are good motivation for the scheduling problem that we address. Typically in scheduling, tasks are scheduled exactly once on a resource. However, to improve safety we assume that jobs can be scheduled more than once. The replication of scheduled jobs is in our case supposed to improve the safety of such systems even further. Since we allow the less critical jobs to be rejected in favor of more critical ones, the replication should increase the probability of execution even for the less critical tasks in the schedule, i.e. it should give us the opportunity to execute some other replica in case we skip the first one in favor of some more critical task.

The so-called F-shapes are well suited for environments such as autonomous driving, communication on networks and other problems with high probability of unexpected disturbing events, where the adaptation of the system has to be fast and schedules can't be recomputed during the online execution. This means that the F-shape scheduling is by nature proactive rather than reactive approach, i.e. it aims at making the schedule robust in advance rather than react on events. Ideally, we would like to remove the need for rescheduling altogether.

The focus of this work is on scheduling of F-shaped tasks onto a set of homogeneous processing units with unit capacity. We aim at optimization of the schedule with respect to safety. Thus the used criteria function is a weighted sum of probabilities of execution for all the scheduled tasks. We will work with instances with general temporal constraints

¹Where F-shape is a job with multiple processing times. See Section 2.1

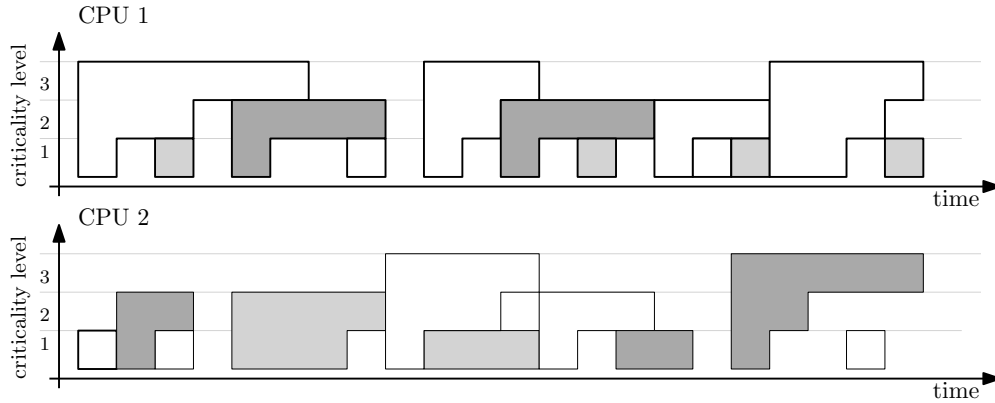


Figure 1: Example of a schedule with F-shapes on two processing units

specifying the mutual position of the tasks in the schedule.

We will extend the existing notation for the mixed-criticality match-up scheduling and make it suited for multiple processing units and job replication. Also, we will propose an interpretation of time lags with respect to replicated jobs and implement two heuristic algorithms seeking and optimizing a feasible schedule for instances of this problem.

In this thesis we work with tasks having up to three possible criticality levels. We have chosen this number, since it shows some problems (regarding the probability of execution) that are not present when using only two levels and yet the complexity of the optimization criterion is not prohibitively high. We will address this further in the Section 2. Also, the use of three criticality levels should be in practice sufficient as the usual number of used criticality levels in similar cases is two [3].

1.1 Related Work

In this work we are dealing with non-preemptive mixed-criticality match-up scheduling, closely related to the one addressed by Hanzálek et al. in [10] and to the preemptive scheduling with multiple possible processing times introduced by Vestal in [22]. The topic is connected to the following scheduling problems and research areas.

Stochastic Scheduling

Stochastic scheduling works with uncertain processing times. We can say that in general the stochastic scheduling is concerned with scheduling problems where the processing times of the tasks are not known in advance. Nevertheless the processing times can be modeled as random variables with some specific distribution. The distribution of the processing times is, in stochastic scheduling, usually continuous.

Kenli et al. [14] consider a stochastic scheduling that deals with the problem of scheduling stochastic tasks on heterogeneous cluster system. Its optimization criterion is minimization of makespan. The tasks are precedence constrained, non-preemptive and the processing times have normal distributions. (Even though it seems to be a different problem, there is an underlying similarity.) Have we not allowed the task duplication and task skipping, we would be solving the same problem only with discrete distributions of processing times.

Another related work is how to create robust partial order schedules for *RCPSP/max* by Fu et al in [8]. This article proposes an approach of Benders Accelerated Cut Creation for Handling Uncertainty (BAC-CHUS). The proposed approach is proactive, non-preemptive and works with temporal constraints.

Canon and Jeannot [4] focus on scheduling problem where stochastic graph (the durations of tasks are uncertain) and heterogeneous environment are given. Their goal is to find a schedule where the average makespan and the standard deviation of the makespan are both minimized. They discuss possible robustness metrics, design a bicriteria scheduling strategy (with respect to makespan and robustness) and propose a MOEA (multiobjective evolutionary algorithm).

Stochastic scheduling has various optimization criteria but it usually doesn't allow to skip any task. We, on the other hand, allow for the jobs to be rejected during online execution. We also, unlike in stochastic scheduling, assume that the distributions of processing times are discrete. This means that the jobs have constant processing time for each criticality level, instead of just interval boundaries for possible processing times.

Job Replication

Since we want to use replication as a tool for maximization of execution probability, we survey works featuring similar ideas. In [18] it is shown how to use task replication for scheduling tasks of DAG onto a heterogeneous system. We can find a different motivation for task replication (task redundancy) in [9, 17] and other works related to fault-tolerant systems. In these works the motivation for replication is a desire to make a fault tolerant schedule and therefore they expect that entire processor can fail. Unlike these systems, we do not aim at ensuring a resistance to some degree of failure, but we mean to replicate some low-criticality tasks to compensate for its rejection.

Temporal Constraints

In general, there are two main attitudes toward temporal constraints. The first one requires the jobs to be scheduled in a given time interval. These constraints are usually expressed by set of release dates and deadlines or due dates. The second possible attitude is concerned with relative temporal constraints between the jobs rather than their position on the time axis. Such constraints are usually expressed by time lags that impose constraints on the relations between start times of jobs present in the schedule. The concept of time

lags was introduced by Mitten in [16].

The time lags can be represented as a directed weighted graph, where the nodes represent jobs and the edges represent the time lags. What does the concept of time lags look like in practice can be seen e.g. in [11].

Match-up Scheduling

The idea of the match-up scheduling is that after a disruption, the system reschedules, but attempts to return to the original schedule later on instead of creating a completely new schedule. Bean et al. introduce a match-up scheduling in [2].

In our case, the match-up is used after executing some job in a high criticality mode, i.e. it can be viewed as a form of robustness. As any needed prolongation of execution time for some critical task is a disturbance in the execution. When the prolonged job ends, we return to the original schedule, instead of rescheduling.

Scheduling with Rejection

Scheduling with rejection is based on the idea that not all jobs have to be scheduled, but some of them can be left unscheduled. The trade off for not scheduling a job is a penalty added to the optimization criteria.

The main difference between our problem and scheduling problems with rejection is, that we require all jobs to be scheduled, whereas the rejection scheduling problem formulation doesn't. Or else, we allow for some jobs to be rejected during online execution of the schedule, but we expect that every job has at least one replica scheduled. On the other hand, the scheduling with rejection expects that some jobs are rejected during the scheduling phase, i.e. prior to the online execution. Thus the scheduling with rejection naturally leads to bicriteria scheduling, but our scheduling has a single criterial function. For further reading on the topic of rejection scheduling a survey on rejection scheduling was done by Shabtay et al. in [20].

The scheduling with rejection, although it might resemble our problem, is a different approach to scheduling and we mention it here primarily to make the distinction clear.

Mixed-Criticality Scheduling

Jan et al. [12] use a stretching factors to prevent a deadline miss of low-criticality tasks inspired by an Elastic Mixed-Criticality task model from [21]. The authors try to avoid dropping the low-criticality tasks. In order to achieve this the deadline of the tasks is considered to be a flexible parameter that can be extended. We (unlike them) do not allow for the change of temporal constraints. The maximization of the probability of execution is to be done by a task replication within the boundaries of temporal constraints.

Cincibus in [5] proposes three algorithms for non-preemptive mixed-criticality scheduling. The tasks are bounded by temporal constraints in a form of positive and negative time lags. The proposed algorithms are made for processing units with unit capacity and the used criterion is makespan minimization.

A variant of our problem for single processing unit, but with release times and deadlines instead of time lags, is addressed in the article on Match-up Scheduling of Mixed-Criticality Jobs by Seddik and Hanzálek [19]. This article studies the mixed-criticality scheduling on one processing unit, using the weighted sum of execution probabilities for scheduled jobs as the optimization criterion. It proposes a dynamic programming algorithm for the case of fixed sequence of jobs and two criticality levels as well as branch and bound algorithm for general problem on one processing unit. It also studies the complexity of the problem and states a MILP formulation of special case with fixed sequence of the jobs. A more specific version of the problem, where the scheduled jobs have to have a triangular shape, is addressed in [7] by Dürr et al.

The problem at hand is a generalization to problems in these works since we use job replication and use more than one processing unit. On the other hand, we allow at most three possible criticality levels and thus in this aspect our problem is less general.

1.2 Contribution

This work follows up on other works done in the field of F-shape scheduling. Our main contribution is the introduction of replication and study of its impact on the probability of execution. For this purpose we extend the existing notation. We also propose a way to compute the criterial function defined as weighted sum of probabilities of execution in the environment with multiple processing units and zero time lags. Furthermore, we propose and implement two algorithms for solving this scheduling problem.

1.3 Outline

The outline of this thesis is the following. Section 2 introduces the problem statement, where we define the problem formally and we introduce the used notation. We also explain the introduced concepts in that section. In Section 3, we further describe how we interpret temporal constraints within the scope of this work. Section 4 introduces the concepts, notations and definitions used for the criterial function definition and an algorithm implementing the forementioned. In Section 5 we provide description of feasible schedule by a mixed integer linear program as well as descriptions of heuristic algorithms we have proposed and implemented for solving the scheduling problem that we address. The experiments we did are to be found together with their results in Section 6. The work is concluded in Section 7.

2 Problem Statement

The problem we are addressing is a scheduling problem, i.e. a problem of finding a schedule (an assignment of processing units and start times) for a set of jobs under specific set of constraints. In accordance with [1, 10, 19] we can describe our problem using standard $\alpha|\beta|\gamma$ notation as $P|temp, mc = 3, mu|\sum_i w_i P_i$. That is a subproblem of $P|temp, mc = \mathcal{L}, mu|\sum_i w_i P_i$, where $mc = \mathcal{L}$ stands for mixed-criticality with \mathcal{L} possible criticality levels, mu stands for match-up, w_i is a weight assigned to job with index i and the P_i is probability of execution of at least one replica for given job. In our case we allow for at most 3 possible criticality levels, hence we denote it as $mc = 3$. The scheduled jobs are non-preemptive.

For the sake of consistency we keep as much as possible from notation introduced by Seddik and Hanzalek in [19]. Thus a portion of the notation conventions and definitions is an extension of formulations used for less general case in the aforementioned publication.

As a part of an input for the problem we get the number m , which is a number of non-dedicated homogeneous (identical) processing units with unit capacity. We denote the set of these machines by $\mathcal{M} = \{\mathcal{M}_1, \dots, \mathcal{M}_m\}$.

We expect a set $\{J_1, \dots, J_n\}$ of n replicable non-preemptive jobs, with up to three possible processing times each, to be given as part of any instance input as well. For each such job, the number of distinct processing times corresponds to its criticality. Also, for every job J_i a value w_i representing its weight is provided. The weight of a job is a multiplicative constant influencing its contribution to the criterial function.

Notation 1. For job J_i we denote its criticality by $\chi_i \in \{1, 2, 3\}$. ($\chi_i \in \{1, \dots, \mathcal{L}\}$ in general case)

Every job J_i with criticality χ_i has a set of χ_i processing times.

Definition 1. For any job J_i we denote its possible processing times by vector $p_i = (p_i^{(1)}, p_i^{(2)}, p_i^{(3)}) \in \mathbb{N}_+^3$ (or $p_i = (p_i^{(1)}, \dots, p_i^{(\mathcal{L})}) \in \mathbb{N}_+^{\mathcal{L}}$ in the generalized case). Since any job J_i should have only χ_i processing times, we will require that $\forall k, \chi_i < k \leq 3 : p_i^{(k)} = 0$ ($\forall k, \chi_i < k \leq \mathcal{L} : p_i^{(k)} = 0$ in the generalized case).

As the processing times at higher criticality levels are meant to provide longer computational time to critical jobs, if these jobs need it, we assume it to hold that processing times at higher criticality levels are longer. To write this formally we extend the first definition by requirement:

Definition 1a. We require it to hold:

$$\forall J_i, J_i \in \{J_1, \dots, J_n\}, k, l \in \{1, \dots, \chi_i\}, k < l \implies p_i^{(k)} < p_i^{(l)}$$

Considering the job replication, we assume that all replicas of one job are assigned to the same processing unit. This assumption seems to be reasonable with regards to possible applications.

Notation 2. For any job J_i we define its assignment to some processing unit by function $\mu(i) : \{1, \dots, n\} \rightarrow \mathcal{M}$, and for convenience we denote the value $\mu(i)$ by μ_i .

The distinct replicas are denoted as follows:

Notation 3. If some job J_i is replicated in a schedule we denote its first, r -th and last replica by $J_{i,1}, J_{i,r}, J_{i,max}$ respectively.

Since we use replication only to increase the safety of the system, we request that only one replica is executed during the execution of the schedule. It means, that whenever some r -th replica of job J_i is executed all other replicas $J_{i,r_1}, r_1 > r$ will not be executed.

Furthermore, the jobs are bounded by the temporal constraints and the replication is used only for execution probability maximization. Thus, it is reasonable to deem it true without a loss of generality that for any instance of the problem, there is a fix upper bound on number of possible replicas per task. Therefore we say that the number of replicas is limited. Justification for this claim is very straightforward. Whenever a job is constrained by some bounded time interval, it can be easily counted how many replicas can be scheduled within given interval. If a job is not restricted by a bounded interval, it can be in theory replicated without limits, and we can then make sure that at least one replica can be scheduled so that it is always executed. In such a case, all the other replicas are not necessary, since they can't increase the execution probability. This means, that for any given instance of the problem we can set max (a maximum number of replicas per task) without a loss of generality of the problem.

Notation 4. For any job J_i we will denote its starting time in a schedule as s_i (or $s_{i,1}$ for replicated jobs). Also, if the job is replicated, we will denote starting times of second, r -th and last replica by $s_{i,2}, s_{i,r}$ and $s_{i,max}$ respectively.

An example of the notation we have just introduced can be seen in Figure 3.

It follows from the nature of the problem that we can't know in advance with which level of criticality a job will be executed during the online execution of the schedule we create. But we expect that for any job the probabilities for distinct criticality levels are known, given the job is executed.

Notation 5. We denote by $B_{i,r,k}$ the probability that r -th replica of job J_i has execution level k , given that it is not rejected.

Notation 6. We denote by $A_{i,r,k}$ the probability that the execution level of $J_{i,r}$ is greater than k , given that $J_{i,r}$ is not rejected.

Theoretically the value of the $B_{i,r,k}$ could vary for different replicas, but since we do not view the replicas as different jobs, but rather as exact copies of the same job, we expect that $\forall r_1, r_2 : B_{i,r_1,k} = B_{i,r_2,k}$ holds. And we denote this value by $B_{i,k}$ unless it is useful to stress out the index of the replica. We assume that the value $B_{i,k}$ is given as part of the problem instance for every tuple (i, k) where $i \in \{1, \dots, n\}, k \in \{1, \dots, \chi_i\}$.

It is obvious, that we can define the value $A_{i,r,k}$ as:

Definition 2. For any $i \in \{1, \dots, n\}, k \in \{1, \dots, \mathcal{L}\}$ it holds:

$$A_{i,r,k} = \sum_{j=k+1}^{\chi_i} B_{i,r,j}.$$

Thus the value of $A_{i,r,k}$ is also not dependent on the value of r , and we can substitute it with just $A_{i,k}$.

We suppose some (any number) of these jobs to be linked by a general temporal constraints in a form of time lags. The set of all such constraints \mathcal{T} is a set of all time lags present in given instance of the problem. The time lag is value from \mathbb{Z} that represents a form of temporal constraint. Unlike releases and deadlines that pose direct constraint on the domain of possible values of variable representing a start time of some job, a time lag imposes a constraint only relatively to some other scheduled job. Namely:

Definition 3. For two jobs J_i, J_j (where $J_i \neq J_j$), a *time lag* $l_{i,j} \in \mathcal{T}$ is a value, such that for their start times s_i, s_j respectively, it has to hold:

$$\forall l_{i,j} : \begin{cases} l_{i,j} \neq 0 \implies s_i + l_{i,j} \leq s_j \\ l_{i,j} = 0 \implies s_i = s_j \end{cases}$$

Every time lag $l_{i,j}$ from job J_i to job J_j creates a relative time-window, i.e. a time interval relative to job J_i in which the job J_j can be scheduled. We will further divide the time lags into three groups.

Notation 7. We say that time lag $l_{i,j}$ is *positive (minimal)* if $l_{i,j} > 0$. We say that time lag $l_{i,j}$ is *negative (maximal)* if $l_{i,j} < 0$. We will call a time lag $l_{i,j} = 0$ a *zero time lag*.

One consequence of the interpretation of time lags we have introduced is, that any two jobs bounded by a zero time lag have to have the same starting time. This means that they can't be scheduled onto the same processing unit.

We will propose how to interpret the time lags with respect to replicated jobs in Section 3. For now we simply demand that, if a schedule is feasible, than all jobs are scheduled within intersection of corresponding relative time windows implied by the time lags.

The solution to the problem is an assignment of all jobs to the processing units and scheduling them on such time slots that the temporal constraints are fulfilled. For a formal definition of this goal, we have to introduce the concept of a schedule.

Definition 4. A *schedule* S is defined as a matrix of tuples ($S \in (\mathbb{Z}^2)^{max \times n}$)

$$S = \begin{pmatrix} [s_{1,1}, \mu_1] & [s_{2,1}, \mu_2] & \cdots & [s_{n,1}, \mu_n] \\ [s_{1,2}, \mu_1] & [s_{2,2}, \mu_2] & \cdots & [s_{n,2}, \mu_n] \\ \vdots & \vdots & \ddots & \vdots \\ [s_{1,max}, \mu_1] & [s_{2,max}, \mu_2] & \cdots & [s_{n,max}, \mu_n] \end{pmatrix}$$

where $s_{i,r}, i \in \{1, \dots, n\}, r \in \{1, \dots, max\}$ is the starting time of r -th replica of job J_i (i.e. of $J_{i,r}$) and $\mu_i, i \in \{1, \dots, m\}$ is index of the processing unit to which all the replicas of given job are assigned.

As the problem is defined, every job has to be scheduled, but not all jobs have to have the same number of replicas. Yet, for the scheduled replicas we will define the following:

Definition 5. Let r_L be last scheduled replica of some job J_i , then

$$\forall r \in \{1, \dots, r_L - 1\} : s_{i,r} < s_{i,r+1}$$

.

For the sake of syntactic convenience we will also define, that:

Definition 6. Let r_L be last scheduled replica of some job J_i , then

$$\forall r \in \{r_L + 1, \dots, max\} : s_{i,r} = s_{i,r_L}$$

.

Definition 6 is only a technical formality, allowing us to describe a schedule by matrix with fix number of rows.

The definition of schedule is in fact only a generalization of the original vector used in [19] for schedule description. Furthermore this definition could be easily modified to assign each replica of one job to a different processing units if so needed (by introducing $\mu_{i,r}$, an index of processing unit to which $J_{i,r}$ would be assigned). Nevertheless, it is reasonable assumption that all replicas of the same job are scheduled on the same processing unit, considering cache and other technical limitations in real-world applications.

It is worth noting that not every possible matrix in this form is feasible with respect to the problem formulation, neither to specific instances. That is why we define a *feasible schedule*.

Definition 7. We define a *feasible schedule* as a schedule that satisfies the following conditions:

1. Temporal constraints are fulfilled, i.e.:

$$\forall l_{i,j} \in \mathcal{T} : \begin{cases} l_{i,j} \neq 0 \implies s_{i,max} + l_{i,j} \leq s_{j,1} \\ l_{i,j} = 0 \implies s_{i,q} = s_{j,r} \iff q = r \end{cases}$$

2. At each level of criticality it holds that jobs do not overlap. Equivalently, jobs do not overlap at their highest common level. We provide formal definition of this constraint in subsection Section 2.1.

It is worth stressing out that the condition $l_{i,j} \neq 0 \implies s_{i,max} + l_{i,j} \leq s_{j,1}$ can be also reformulated as,

$$l_{i,j} \neq 0 \implies \forall r, s_{j,r} \in [s_{i,max} + l_{i,j}, \infty)$$

This half-closed interval is the relative time-window we have talked about at the beginning of this section. We reason about this time lag interpretation in Section 3

The optimization criteria for this scheduling problem is weighted sum of execution probabilities. If a job is replicated, then by execution probability we denote the probability that at least one replica is executed (computed over all possible execution scenarios). An exact formulation of execution probability is provided in the Section 4. To describe the problem even better we will now focus on other specific features it has.

2.1 Job as F-shape

We have assumed that the jobs we are scheduling can have up to three possible processing times. That leads us to the problem of scheduling jobs with discrete distributions over possible processing times of each job. We have also stated that the number of processing times is the same as the criticality level of given job.

For visualization of jobs with multiple processing times we use so-called *F-shape* abstraction. An example of F-shape visualization is shown in Figure 2. In this form of visualization, every level of the F-shape represents one possible processing time of given job. Thus the classical visualization of a job as a rectangle is substituted by more general "F-shaped" object as introduced in [10].

For such jobs, a constraint requiring every job to be scheduled after the end of the previous one, would lead to very low utilization of the processing units. To avoid this, we can "stack" some of the less critical jobs under the more critical ones as shown in Figure 4.

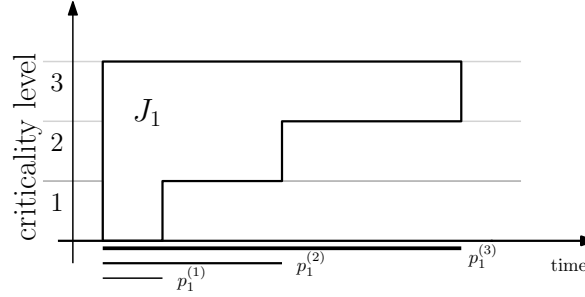


Figure 2: Visualization of some job J_1 in a form of an F-shape with three possible processing times (p_1^1, p_1^2, p_1^3)

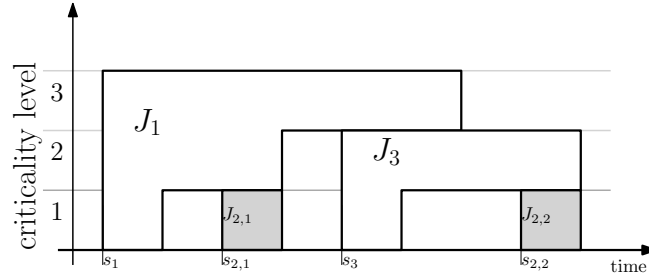


Figure 3: Example of a feasible schedule with 3 jobs on one processing unit.²

If, however, the more critical job is prolonged during the schedule execution, the less critical one is rejected.

A formal notation for this proposal is:

Definition 8. For any two job replicas $J_{i,r}, J_{j,s}$ of jobs J_i, J_j such that $\mu_i = \mu_j$ and $s_{i,r} < s_{j,s}$ it has to hold: $s_{i,r} + p_i^{(\chi_{min})} \leq s_{j,s}$, where $\chi_{min} = \min(\chi_i, \chi_j)$.

In Figure 3, the job J_3 and job replica $J_{2,1}$ start before the worst case processing time of J_1 is finished. In such a case we say that job J_3 covers the other two.

Definition 9. Given a feasible schedule S and two scheduled job replicas $J_{i,q}, J_{j,r}$ in S , we say that job replica $J_{i,q}$ *covers* job replica $J_{j,r}$ if and only if $s_{i,q} < s_{j,r} < s_{i,q} + p_i^{\chi_i}$ and $\mu_i = \mu_j$.

We also use the following notation.

Notation 8. We denote by $cov(J_{i,r}, S)$ ($cov(J_{i,r})$ when no ambiguity possible) the set of all job replicas covering $J_{i,r}$ in schedule S and we call this set a *coverage* of job replica $J_{i,r}$.

²An example with computation of execution probability is shown at the end of Section 2.3

We will use this concept of *coverage* extensively, for formal definition of probability of execution later on.

In general, such a set can have up to $\mathcal{L} - 1$ elements. Since we allow only 3 possible criticality levels, the cardinality of covering set for any job, has to be lower than 3. Nevertheless, a single job can be member of more than one coverage, i.e. it can cover more than one job. For example, in Figure 3 the job J_1 covers both job replicas $J_{2,1}$ and J_3 . Thus in that example $cov(J_{2,1}) = cov(J_3) = \{J_1\}$.

We define a *coverage level* as follows:

Definition 10. In schedule S , given a job replicas $J_{i,q}, J_{j,r}$ and $i, j \in \{1, \dots, n\}, i \neq j$ and $r, q \in \{1, \dots, max\}$ such that $J_{j,r} \in cov(J_{i,q})$. We define the *coverage level* of $J_{i,q}$ by $J_{j,r}$ as $c_{i,q,j,r}(S)$ (or $c_{i,q,j,r}$ when no ambiguity is possible) so that it holds:

$$c_{i,q,j,r} = l \iff s_{i,q} \in [s_{j,r} + p_j^{(l)}, s_{j,r} + p_j^{(l+1)}].$$

for some $l \in \{1, \dots, \chi_j\}$.

2.2 Online Execution of the Schedule

As we have previously stated it is not known in advance which processing time will a scheduled job need for its completion. For any schedule S , composed of F-shapes, we only know the probability distributions over the possible processing times of the F-shapes. This means that there are multiple scenarios in which such a schedule can be executed. Hanzalek et al. [10] proposed what the on-line execution with match-up should look like for a single processing unit. They propose a function $e : [t] \rightarrow \{1, 2, \dots, \mathcal{L}\}$ (t represents time) called execution level. The function is revealed only by execution of the scheduled tasks and $e(t)$ defines the criticality level on which the schedule was executed at time t . This function can then be used to describe the possible execution scenarios.

The original definition of execution level is not sufficient for multiple processing units, so we extend it for our needs.³ This being said, we define the execution level function as $f_e : [t, \mathcal{M}] \rightarrow \{1, 2, \dots, \mathcal{L}\}$, where $\mathcal{M} = \{\mathcal{M}_1, \dots, \mathcal{M}_m\}$ is set of all processing units and t is time. For all jobs, we will denote a processing unit to which a job J_i is assigned by μ_i . With respect to this notation, the function f_e is described by the following rules:

1. At origin it holds that $\forall \mathcal{M}_i \in \mathcal{M} : f_e(0, \mathcal{M}_i) = 1$
2. For $i \in \{1, \dots, n\}, r \in \{1, \dots, max\}$ let job replica $J_{i,r}$ be scheduled on processing unit μ_i and let $\epsilon > 0$ be a small constant. While executing job replica $J_{i,r}$ on execution level f_e at time instant $(s_{i,r} + p_i^{(l)} - \epsilon)$ for some $l \in \{1, \dots, \chi_i\}$ the value of $f_e(s_{i,r} + p_i^{(l)}, \mu_i)$ is:

³We also use a different name, because we want to keep the e reserved for notation of another concept.

- set to 1 if the job replica $J_{i,r}$ is completed.
 - incremented (by one) if the job replica $J_{i,r}$ is not completed and $f_e(s_{i,r} + p_i^{(l)} - \epsilon, \mu_i) < \chi_i$.
 - undefined if the job replica $J_{i,r}$ is not completed and $f_e(s_{i,r} + p_i^{(l)} - \epsilon, \mu_i) = \chi_i$. In such a case, it is not possible for the job to finish within its maximal possible processing time, which we consider to be an ill defined problem instance.
3. Otherwise the execution level remains constant.

Furthermore it holds that if $f_e(s_{i,r}, \mu_i) = 1$ and no other previous replica of job J_i was executed, then job replica $J_{i,r}$ is started at $s_{i,r}$ on processing unit μ_i , otherwise job replica $J_{i,r}$ is rejected.

The execution level function has different shapes for different executions of the schedule. Possible execution scenarios for a simple schedule with one processing unit are shown in Figure 5.

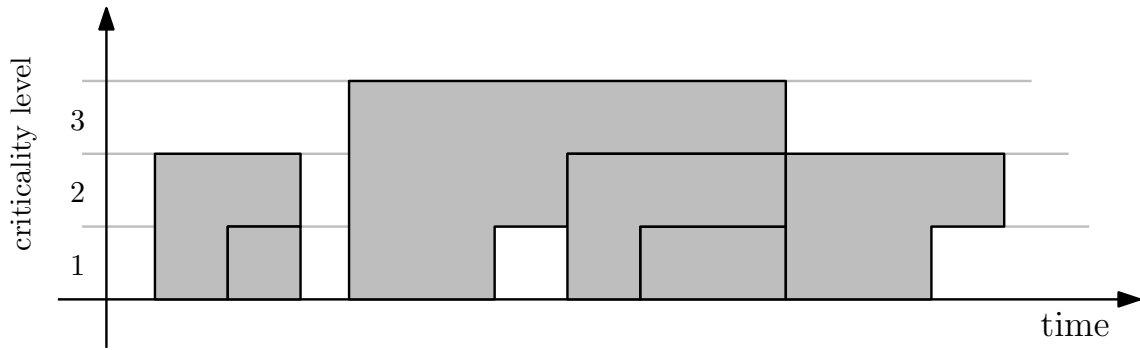


Figure 4: Schedule consisting of F-shapes

2.3 Replication

In this section we introduce the concept of replication to the field of F-shape scheduling. By replication we mean a repeated scheduling of the same job, as was previously mentioned. A simple example of schedule with replication can be seen in Figure 3.

If a job is covered by some other, more critical job, it will inevitably be rejected in some scenarios. The higher the probability of these scenarios, the lower the chance, that the less critical job will be executed. It also holds, that with growing utilization of a processing unit and higher ratio of high criticality jobs in the schedule, the possibilities for scheduling less critical jobs, so that they are not covered by the more critical ones, shrink. However,

it is possible to increase the number of execution scenarios in which some low criticality job J_i is executed, by replicating it.⁴

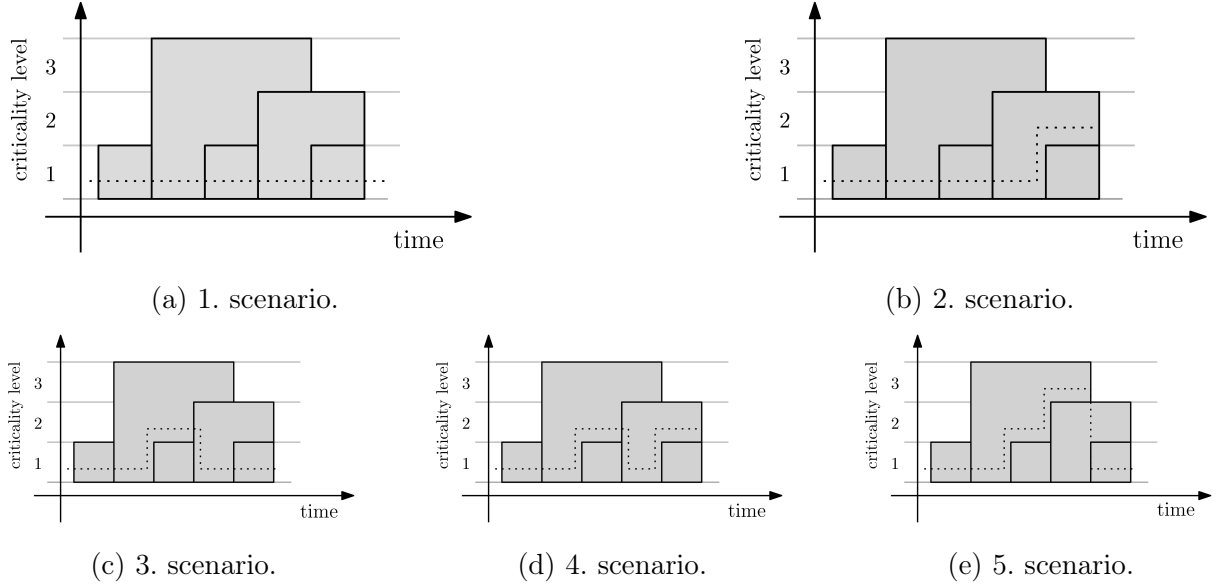


Figure 5: Possible scenarios in a schedule with five F-shapes on a single processing unit. The dotted line represents the executing level function.

As we have mentioned, we suppose that whenever a replica is executed during the online execution, all consequent replicas are rejected. Thus the probability that at least one replica of given job is executed (computed over all possible scenarios) equals the probability that exactly one replica is executed. And we call this probability the *probability of execution*.

To introduce this concept formally, we will first define *execution scenario* as follows:

Definition 11. An *execution scenario* sc is a matrix of levels at which all job replicas are executed during online execution of given schedule S . For each replica of each job there is exactly one value.

$$sc = \begin{pmatrix} e_{1,1} & e_{2,1} & \cdots & e_{n,1} \\ e_{1,2} & e_{2,2} & \cdots & e_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ e_{1,max} & e_{2,max} & \cdots & e_{n,max} \end{pmatrix}$$

where $e_{i,r} \in 0, \dots, \chi_i, i \in \{1, \dots, n\}$ and $r \in \{1, \dots, max\}$. The value of $e_{i,r} = 0$ if and only if given replica is unscheduled or rejected.

A consequence of the fact that at most one replica is executed in one scenario is, that if

⁴We demonstrate this in an example at the end of this section.

job replica $J_{i,q}$ is executed in given scenario, then all replicas $J_{i,r}, q < r \leq \max$ are rejected, therefore if in given scenario $e_{i,q} > 0$, it means that $\forall r, q < r \leq \max : e_{i,r} = 0$.

Notation 9. We denote by

$$sc^* = \begin{pmatrix} e_{1,1}^* & e_{2,1}^* & \cdots & e_{n,1}^* \\ e_{1,2}^* & e_{2,2}^* & \cdots & e_{n,2}^* \\ \vdots & \vdots & \ddots & \vdots \\ e_{1,\max}^* & e_{2,\max}^* & \cdots & e_{n,\max}^* \end{pmatrix}$$

the (initially unknown) scenario that occurs at runtime.

Similarly to the schedule, this definition of scenario is in fact a generalization of the original sc^* used in [19] (there it is a vector, not matrix, as the jobs are not replicated). Furthermore, the relation between the concept of scenario we have just introduced and the function f_e is the following.

Definition 12. Let

$$S = \begin{pmatrix} [s_{1,1}, \mu_1] & [s_{2,1}, \mu_2] & \cdots & [s_{n,1}, \mu_n] \\ [s_{1,2}, \mu_1] & [s_{2,2}, \mu_2] & \cdots & [s_{n,2}, \mu_n] \\ \vdots & \vdots & \ddots & \vdots \\ [s_{1,\max}, \mu_1] & [s_{2,\max}, \mu_2] & \cdots & [s_{n,\max}, \mu_n] \end{pmatrix}$$

be a feasible schedule for some instance of our problem. Let

$$sc^* = \begin{pmatrix} e_{1,1}^* & e_{2,1}^* & \cdots & e_{n,1}^* \\ e_{1,2}^* & e_{2,2}^* & \cdots & e_{n,2}^* \\ \vdots & \vdots & \ddots & \vdots \\ e_{1,\max}^* & e_{2,\max}^* & \cdots & e_{n,\max}^* \end{pmatrix}$$

be an execution scenario revealed during online execution of schedule S and let $f_e : [time, M] \rightarrow \{1, 2, \dots, \mathcal{L}\}$ be an execution level function revealed during the same execution of schedule S and $\epsilon > 0$ be a small constant. Then for any scheduled job replica it holds that:

$$\forall J_{i,r} \begin{cases} f_e(s_{i,r}, \mu_i) = 1 \wedge \forall q < r, e_{i,q}^* = 0 \implies e_{i,r}^* = f_e((s_{i,r} + p_i^{x_i} - \epsilon), \mu_i) \\ f_e(s_{i,r}, \mu_i) \neq 1 \vee \exists q < r, e_{i,q}^* \neq 0 \implies e_{i,r}^* = 0 \end{cases}$$

Now, when we have introduced the concept of scenario we can proceed to the definition of execution probability.

Notation 10. We will denote the execution probability for any job replica $J_{i,r}$ by P_i . If the job is replicated, we will denote by $P_{i,r}$ the probability of execution for r -th replica of given job, and we will define $P_i = \sum_{r=1}^{\max} P_{i,r}$.

We can only say what the value of execution probability for given job is, with respect to some specific schedule (i.e. it makes no sense to speak about execution probability before the job is scheduled). Hence, whenever we speak about a probability of execution, we suppose that a feasible schedule was already created.

Definition 13. Given a feasible schedule S , the execution probability $P_{i,r}$ of some job replica $J_{i,r}$ can be defined as $P_{i,r} = P(e_{i,r}^* \neq 0)$.

We should stress out that since $e_{i,r}^* = 0$ for all unscheduled replicas, the value of $P_{i,r}$ of unscheduled replicas is always equal to zero.

Computation of the execution probability according to the definition is, however, not feasible for large instances. Therefore, we will introduce a better way for its computation later.

Considering the replication, it doesn't only provide us with more opportunities for job execution, but it has its disadvantages as well. On one hand, the replication provides us with the benefit of possible execution probability increase. On the other hand, it requires new interpretation of the time lags and increases the computational complexity of criterial function.

To provide an example of execution probability increase caused by job replication, we show a simple schedule consisting of three jobs, and we evaluate the execution probability with and without replication in Figure 6. In this example the job J_2 is always executed, although both its replicas have non empty coverage ($cov(J_{2,1}) = \{J_1\}$ and $cov(J_{2,2}) = \{J_3\}$). This statement is easy to prove, as job replica $J_{2,1}$ is executed whenever, $e_1^* < 3$. If $e_1^* = 3$ then job replicas $J_{2,1}$ and J_3 are rejected and thus the job replica $J_{2,2}$ is executed. This means, that $P_2 = P_{2,1} + P_{2,2} = 1$. If either of the replicas $J_{2,1}, J_{2,2}$ is omitted from the schedule, the probability of execution for job J_2 would decrease, as it would be executed in only one of the possible cases ($e_1^* = 3; e_1^* < 3$).

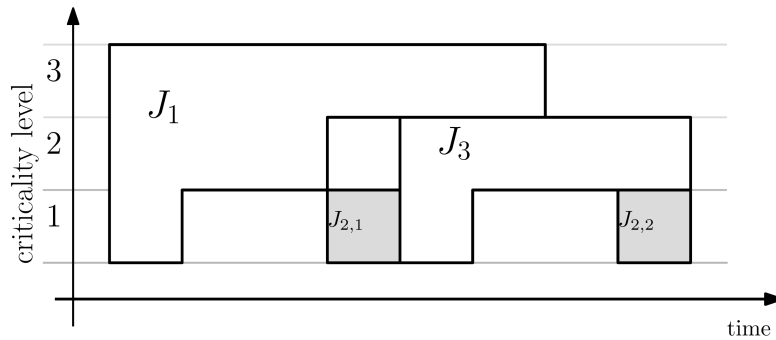


Figure 6: A "trap" example. In this schedule, job J_2 has nonempty coverage, but is replicated so that it is always executed.

2.4 Summary

We have introduced the problem and explained the concept of F-shape and replication. The interpretation of time lags in an environment with replication as well as the computation of execution probability will be further discussed in Sections 3 and 4 as these two topics require a more thorough explanation.

We have also introduced new notation, and extended a notation originally proposed for a less general cases of a similar problem.

2.5 Problem Complexity

The problem we have specified is strongly \mathcal{NP} -hard. Y. Seddik and Z. Hanzalek in [19] show that $1|r_j, \tilde{d}_j, mc, mu|\sum_i w_i P_i$ is strongly \mathcal{NP} -hard. This suggests that even our problem could be strongly \mathcal{NP} -hard, since its generalized version $P|temp, mc, mu|\sum_i w_i P_i$ contains $1|r_j, \tilde{d}_j, mc, mu|\sum_i w_i P_i$ as its subproblem.

Also, the decision problem $1|mc = 2, mu|C_{max} \leq \epsilon$ is p -reducible to $1|temp, mc = 2, mu|feasibility$ scheduling problem. The transformation can be done by Algorithm 1. It holds, that whenever we find a solution to the $1|mc = 2, mu|C_{max} \leq \epsilon$ problem, a feasible solution to $1|temp, mc = 2, mu|feasibility$ has to exist and it can be obtained by simply adding the dummy task d_0 to the beginning, shifting all original start times by 1 and appending d_{n+1} to the end. Similarly, whenever there is a feasible solution to $1|temp, mc = 2, mu|feasibility$, we can obtain a solution to $1|mc = 2, mu|C_{max} \leq \epsilon$ by omitting the dummy tasks d_0, d_{n+1} from the schedule and shifting start times of all other jobs by one to the left.

As the problem of $1|temp, mc = 2, mu|feasibility$ is obvious subproblem of $P|temp, mc = 3, mu|\sum_i w_i P_i$, we can claim that our problem is strongly \mathcal{NP} -hard, since the problem $1|mc = 2, mu|C_{max}$ is strongly \mathcal{NP} -hard as was proved in [10] by Hanzalek et al.

Algorithm 1 $1|mc = 2, mu|C_{max} \leq \epsilon \triangleleft_p 1|temp, mc = 2, mu|feasibility$

Require: instance of $1|mc = 2, mu|C_{max} \leq \epsilon$ with n jobs

- 1: add dummy tasks d_0, d_{n+1}
 - 2: $\chi_0 = 1 = \chi_{n+1}$ and $p_0^{(1)} = p_{n+1}^{(1)} = 1$
 - 3: for each job j_i add time lags $l_{0,i} = 1$ and $l_{i,n+1} = 1$
 - 4: add time lag $l_{n+1,0} = -(\epsilon + 2)$
 - 5: add time lag $l_{0,n+1} = 1$
-

2.6 Blanket Definition

Finally, we introduce a few more important concepts that will become useful in the section dedicated to computation of execution probability.

Therefore, in addition to the definitions we have introduced, we make a simple observation about coverage.

Observation 1. If for any two job replicas $J_{i,q}, J_{i,r}$ it holds that $(cov(J_{i,q}) \subset cov(J_{i,r})) \wedge (\forall J_{j,s} \in cov(J_{i,q}) : c_{i,q,j,s} = c_{i,r,j,s})$ then by omitting the replica $J_{i,r}$ from the schedule, the probability of execution for job J_i doesn't decrease.

Proof. The replica $J_{i,r}$ will be rejected in all scenarios where replica $J_{i,q}$ is rejected, unless $J_{i,q}$ is rejected by execution of $J_{i,r}$. \square

And we further extend the concept of coverage by introduction of *blanket*.

Definition 14. We define *blanket* for job replica $J_{i,r}$ as a set of all job replicas $J_{j,q}$ such that:

1. $\forall J_{j,q} \in cov(J_{i,r}) : J_{j,q} \in blanket(J_{i,r})$
2. $\forall r_1 < r : J_{i,r_1} \in blanket(J_{i,r})$
3. $\forall J_{j,q} \in blanket(J_{i,r}), \forall q_1 \leq q : J_{j,q_1} \in blanket(J_{i,r})$
4. $\forall J_{j,q}, \forall J_{k,s} \in cov(J_{j,q}) : J_{j,q} \in blanket(J_{i,r}) \implies J_{k,s} \in blanket(J_{i,r})$
5. the cardinality of the set is minimal.

We will denote this set as $blanket(J_{i,r})$.

Example of the blanket is shown in Figure 7.

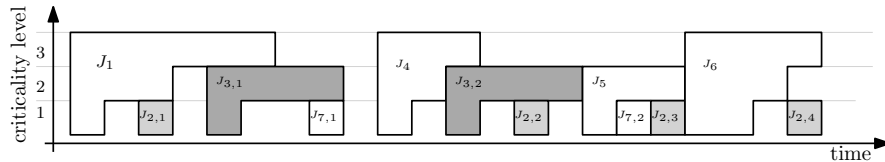


Figure 7: Blanket example

In this example the $blanket(J_{2,4}) = \{J_1, J_{2,1}, J_{3,1}, J_4, J_{3,2}, J_{2,2}, J_5, J_{2,3}, J_6\}$.

2.7 Example of Task Instance

An instance of the problem has to define:

- $m \in \mathbb{N} \sim$ number of processing units
- $n \in \mathbb{N} \sim$ number of jobs
- $max \in \mathbb{N} \sim$ maximal number of replicas per task. If this limitation is not provided, we can compute it as we have shown previously, but for simplicity we expect it to be present in the instance description.
- $\chi_i \in \{1, 2, 3\}, \forall i \in \{1, \dots, n\} \sim$ criticality level of each job
- $p_i^l \in \mathbb{N}, \forall i \in \{1, \dots, n\}, \forall l \in \{1, \dots, \chi_i\} \sim$ processing times for each job
- $w_i \in \mathbb{N}, \forall i \in \{1, \dots, n\} \sim$ weight for each of the jobs
- $B_{i,l} \in [0, 1], \forall i \in \{1, \dots, n\}, \forall l \in \{1, \dots, \chi_i\} \sim$ probabilities that a i -th job reaches criticality level l during its execution.
- $\mathcal{T} \sim$ set of all time lags present in the instance

Thus a small instance that consists of only five jobs, can be defined in the following way:

- $m = 2$
- $n = 5$
- $max = 3$
- $\chi_1 = 1, \chi_2 = 2, \chi_3 = 3, \chi_4 = 1, \chi_5 = 1$
- $p_1^{(1)} = 1, p_2^{(1)} = 1, p_2^{(2)} = 3, p_3^{(1)} = 2, p_3^{(2)} = 3, p_3^{(3)} = 5, p_4^{(1)} = 1, p_5^{(1)} = 2$
- $w_1 = 1, w_2 = 2, w_3 = 3, w_4 = 5, w_5 = 1$
- $B_{1,1} = 1, B_{2,1} = 0.5, B_{2,2} = 0.5, B_{3,1} = 0.8, B_{3,2} = 0.1, B_{3,3} = 0.1, B_{4,1} = 1, B_{5,1} = 1$
- $l_{1,2} = 2, l_{1,5} = 2, l_{2,3} = 0, l_{2,4} = 2, l_{3,1} = -4, l_{3,2} = 0, l_{4,1} = -6, l_{5,1} = -5$

A feasible solution, yet not an optimal one, is shown in Figure 8a and the graph representing the time lags can be seen in Figure 8b.

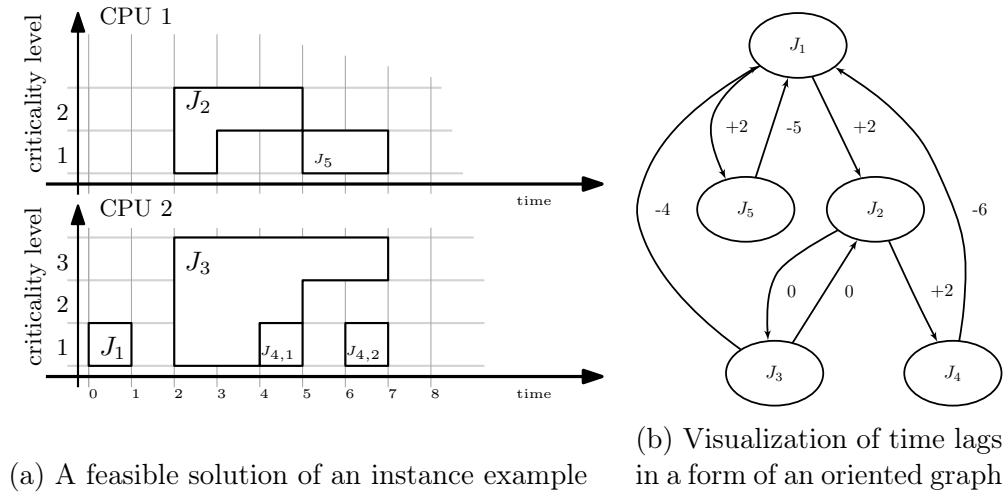


Figure 8: An instance solution and graph of time lags

3 Time Lags

The concept of temporal constraints in a form of time lags is not new and its interpretation is well defined for schedules without replication. This section is therefore focused solely on the use of time lags with respect to replicated jobs.

3.1 Interpretation of Time Lags

The source of possible complications when interpreting the time lags is the fact, that we don't know in advance which replicas will be rejected during the online execution of a schedule. This means that there are several possible interpretations. We could just decide that we will take into account only the first replicas (or some other arbitrary index). This would be a very simple solution, yet a wrong one, as this would mean that the time lags provide no guarantee on the relation between the start times of the replicas that will actually be executed. As we consider similar ideas to go against the original meaning of time lags, we propose the following definition.

Definition 15. We consider any time lag $l_{i,j} \neq 0$ between replicas of jobs J_i and J_j fulfilled if and only if $\forall r \in \{1, \dots, max\}, \forall q \in \{1, \dots, max\} : s_{i,r} + l_{i,j} \leq s_{j,q}$.

An obvious consequence of this definition is a fact, that we formulate in the next observation.

Observation 2. Let r_L be an index of the last scheduled replica of some job J_i . Then it must hold:

$$(s_{i,r_L} + l_{i,j} \leq s_{j,1}) \implies (\forall r \in \{1, \dots, r_L\}, \forall q \in \{1, \dots, max\} : s_{i,r} + l_{i,j} \leq s_{j,q}).$$

Furthermore, since we have defined that we assign the value of start time of the last scheduled replica to all the unscheduled replicas of the same job, we can now define the meaning of the time lags as follows.

Definition 16. From Definitions 5 and 6 it follows that:

$$\forall i \in \{1, \dots, n\}, \forall r \in \{1, \dots, max\} : s_{i,r} \leq s_{i,r+1}$$

therefore we considered any time lag $l_{i,j} \neq 0$ between replicas of jobs J_i and J_j fulfilled if and only if

$$s_{i,max} + l_{i,j} \leq s_{j,1}$$

.

Such interpretation of positive and negative time lags is in accordance with the original interpretation for schedules without replication. This can be easily shown by setting the $max = 1$. In such a case we would get the same formulation we have introduced in Definition 3. If $max > 1$ (i.e. replication is present), then this interpretation guarantees the original time lag to be valid whenever at least one replica of both jobs bounded by the time lag is executed.

To make the interpretation complete, we should address the zero time lags. The original interpretation of zero time lags is that the two jobs have to start at the same time. This interpretation seems to be very natural and we will try to preserve it. This means that if some jobs J_i and J_j are constrained by time lag $l_{i,j} = 0$ we will replicate them together. It means that for any feasible schedule we demand:

Definition 17. For any two jobs J_i and J_j bounded by time lag $l_{i,j} = 0$, it has to hold:

$$\forall l_{i,j} = 0, \forall q, r \in \{1, \dots, max\} : s_{i,r} = s_{j,q} \iff q = r$$

.

By combining Definitions 16 and 17 we get the definition:

Definition 18. We say that a schedule fulfills all time lags if and only if:

$$\forall i, j \in \{1, \dots, n\} : \begin{cases} \exists l_{i,j} \neq 0 \implies s_{i,max} + l_{i,j} \leq s_{j,1} \\ \exists l_{i,j} = 0 \implies \forall q, r \in \{1, \dots, max\} : s_{i,r} = s_{j,q} \iff q = r \end{cases}$$

This interpretation is safe in the sense, that even if the constraints, implied by the time lags, are active, the schedule remains feasible regardless of which replica will be executed. On the other hand the weakness of this solution is, that job replication shrinks the relative time windows, or even closes them completely and therefore the possibility of job replication is limited.

Also, it is not enough to demand that the replicas are scheduled together. But we should make sure that replicas of these tasks are executed (or dropped) together as well and we expect the online executor of the schedule to abide this rule. A consequence of such interpretation of this schedule is:

Observation 3. If two jobs J_i, J_j are bounded by zero time lag, then for any possible scenario

$$sc = \begin{pmatrix} e_{1,1} & e_{2,1} & \cdots & e_{n,1} \\ e_{1,2} & e_{2,2} & \cdots & e_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ e_{1,max} & e_{2,max} & \cdots & e_{n,max} \end{pmatrix}$$

it has to hold:

$$\forall r \in \{1, \dots, max\} : e_{i,r} \neq 0 \iff e_{j,r} \neq 0$$

For the sake of clarity we define two other terms.

Definition 19. We say, that jobs J_i, J_j are bounded by *transitive zero time lag*, denoted by $l^T(J_i, J_j)$, if either:

$$\exists J_k, l_{i,k} = 0 = l_{k,j}$$

or

$$\exists J_k, J_h, l_{i,k} = 0 = l_{k,h} \wedge l^T(J_h, J_j)$$

.

It makes sense to speak about sets of jobs bounded by transitive zero time lags as about batches.

Definition 20. We define *batch containing job replica $J_{i,r}$* , denoted by $Batch(J_{i,r})$, as a set of all job replicas $J_{j,q}$ such that $l^T(J_i, J_j)$ and a job replica $J_{i,r}$.

One consequence of introducing zero time lags is that the execution function f_e , from Section 2.2, has to be modified to put up with the concept of batches. Namely, the rule for job execution has to be reformulated as follows:

Proposition 1. *For any scheduled job replica $J_{i,r}$ it has to hold: If $\forall J_{j,q} \in Batch(J_{i,r})$ the value of $f_e(s_{j,q}, \mu_j) = 1$ and no previous replica of job J_j was executed, then all job replicas in $Batch(J_{i,r})$ are started at $s_{i,r}$, otherwise all job replicas in $Batch(J_{i,r})$ are rejected.*

Similarly, from Observation 3 and Definition 19 we get:

Observation 4. If two jobs $J_{j,q}, J_{i,r}$ are bounded by a transitive zero time lag $l^T(J_i, J_j)$, then for any possible scenario

$$sc = \begin{pmatrix} e_{1,1} & e_{2,1} & \cdots & e_{n,1} \\ e_{1,2} & e_{2,2} & \cdots & e_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ e_{1,max} & e_{2,max} & \cdots & e_{n,max} \end{pmatrix}$$

it has to hold:

$$\forall r \in \{1, \dots, max\} : e_{i,r} \neq 0 \iff e_{j,r} \neq 0$$

.

Specially, from Observation 3 and Definition 20 we get the rule that all replicas in a batch are either executed or rejected together, i.e.

$$\forall J_{j,r} \in Batch(J_{i,r}) : e_{i,r} \neq 0 \iff e_{j,r} \neq 0$$

3.2 Observations and Notes

Furthermore, we can make some interesting observations about the schedules that are worthy of being mentioned.

Observation 5. If some job is not included in any cycle (considering the graph of temporal constraints), then it can be shifted or replicated so that probability of its execution is equal to one. Specially, if the graph doesn't contain any cycle (i.e. also no relative time windows are present), it has a trivial optimal solution.

We do not prove this observation here, but the idea on which such a proof could be based is simple. We do not require to minimize makespan of the schedule, but only to maximize probability of execution. This means, that by enlarging the makespan, we can't violate any relative temporal constraints, and yet for any job J_i that is not present in any cycle, it is possible to find such a time slot that $cov(J_i) = \emptyset$.

Observation 6. Any job replica $J_{i,r}$ such that $cov(J_{i,r}) = \emptyset$ and $Batch(J_{i,r}) = \{J_{i,r}\}$ is always executed.

Observation 7. Any job replica $J_{i,r}$, $\chi_i = 3$, $Batch(J_{i,r}) = \{J_{i,r}\}$ is always executed, unless bounded by a zero time lag to some covered job replica. In general, any job replica $J_{i,r}$ such that $\chi_i = \mathcal{L}$ and $Batch(J_{i,r}) = \{J_{i,r}\}$ is always executed.

4 Probability of Job Execution

In this section we will focus on evaluation of the optimization criteria. That is, the computation of probability of execution for all jobs in given schedule. At first, we will address the easiest version when we ignore both the zero time lags and the possibility of job replication.

4.1 Without Job Replication

The most straightforward solution to this problem is a brute force computation from definition (i.e. computing probability for each job over every possible scenario). This, however, isn't possible for large instances, because the number of scenarios grows exponentially with respect to the number of scheduled jobs.

Much faster solution for computation of probability of execution is to compute the probability of execution for any job J_i taking into account only its coverage $cov(J_i)$ as is shown in [19]. There, the following formula is used.

Proposition 2. *Given a schedule S , we have for every job J_i in S :*

$$P_i = 1 - \sum_{J_j \in cov(J_i)} (P_j \times A_j, c_{i,j})$$

where the $c_{i,j}$ is coverage level of J_i by J_j .

We can adapt this method for our use if we do not take into account the job replication and the zero time lags.

However, the replication and zero time lags do both complicate the computation. We will first explain how to handle the complication caused by the possible job replication.

4.2 With Job Replication

When we allow the job replication to be present in the problem, the complexity of computation of the criteria function grows. The brute force solution is still possible in theory, yet intractable for larger instances. In fact, it fails even for relatively small instances containing no more than 23 jobs (as is shown in Section 6.1).

Furthermore, it is no longer possible to compute the probability of execution for job J_i using only the $cov(J_i)$. To justify this claim we remind the reader of the fact that any replica $J_{i,r}$ is executed only if all the replicas $J_{i,r_1}, r_1 \leq r$ are rejected. Another factor complicating

the computation even further is that even jobs in the $cov(J_i)$ might be replicated. Thus, even previous replicas of a covering job are influencing the probability of execution.

This creates a need for generalization of coverage. Therefore, we would like to find a set $cov^*(J_{i,r})$ that is a generalization of the coverage of job replica $J_{i,r}$ used in the algorithm for computation of probability of execution in the easier case. To find such a set we first formulate what properties this set has to have and then we will define it formally.

Observation 8. Since we have said that we talk about generalization, we request that

$$cov(J_{i,r}) \subseteq cov^*(J_{i,r})$$

is true statement. We need this, because if we set $max = 1$, the set $cov^*(J_{i,1})$ must be equal to set $cov(J_{i,1}) = cov(J_i)$.

Observation 9. The probability of execution for any job replica $J_{i,r}$ is influenced by execution or rejection of all previous replicas of the same job. From this we get:

$$\forall r_1 \leq r : J_{i,r_1} \in cov^*(J_{i,r})$$

This seems to give us at least some idea about what the $cov^*(J_{i,r})$ should look like, but we need to specify it fully.

We want the set $cov^*(J_{i,r})$ to contain everything that influences the probability of execution for job replica $J_{i,r}$ in a schedule with replication but no zero time lags. Therefore, we need this set to be recursive in a fashion described by the next observation.

Observation 10.

$$\forall J_{j,q} \in cov^*(J_{i,r}) : cov^*(J_{j,q}) \subseteq cov^*(J_{i,r}).$$

By combining all these observations about the set $cov^*(J_{i,r})$ we get a set that contains all the job replicas that can influence the probability of $J_{i,r}$. The last thing we would like to be true about $cov^*(J_{i,r})$ is, that $cov^*(J_{i,r})$ should have the minimal possible cardinality (i.e. it shouldn't contain any other job replicas).

These requirements lead us to the simple fact, that:

Observation 11. By comparing the requirements for this generalized set and the Definition 14 (where the *blanket* is defined) we get:

$$cov^*(J_{i,r}) = blanket(J_{i,r})$$

It is also obvious that for a fixed number of replicas max and fixed number of criticality levels \mathcal{L} it must hold:

$$|blanket(J_{i,r})| \leq (max^1 + max^2 + \dots + max^{\mathcal{L}-2} + max^{\mathcal{L}-1} + max^{\mathcal{L}-1}).$$

This boundary is the worst-case scenario where every job replica $J_{j,q}$ on criticality level l is covered by distinct job replica $J_{k,max}$ on the level $l + 1$, for all $l < \mathcal{L}$. On the highest criticality level it makes no sense to replicate the tasks, since they can be never dropped as their coverage is always empty.

Since we focus only on the case where number of criticality levels is 3, we know that $|blanket(J_{i,r})| \leq (max + 2 \cdot max^2)$. This means that for small number of replications it is possible to compute the probability of execution of $J_{i,r}$ by brute force over the $blanket(J_{i,r})$. The results for this method are shown in Section 6.1.

4.3 Zero Time Lags and Job Replication

The zero time lags add an extra layer to the complexity. We have set no constraints limiting which jobs can be bounded by zero time lags. This means that even the most critical jobs can be in a *Batch* set of some job with the lowest criticality. From Proposition 1 we get:

Observation 12. Let $J_{i,r}, \chi_i \leq \mathcal{L}$ be a replica of some job and let $J_{j,q}, \chi_j = \mathcal{L}$ be some other job replica, such that $J_{j,q} \in Batch(J_{i,r})$. Then, whenever $cov(J_{i,r}) \neq \emptyset$, there exists a scenario sc , in which both replicas $J_{i,r}, J_{j,q}$ are rejected.

As a consequence, it sometimes makes sense to replicate even the most critical jobs, if they are bounded by zero time lags to some less critical job. In a single processor instance, the previously introduced concept of blanket is sufficient, as no zero time lag can be present. However, the blanket is not sufficient for setup with multiple processing units, because the blanket only takes into account the jobs scheduled on the same processing unit, but execution of some job replicas can be influenced by execution of job replicas on other processing units as well. This means that the generalization we have found is not general enough and we will have to generalize it further. Thus, we will seek an extension of the blanket. But first we will introduce an extended version of a coverage.

Definition 21. We define an *extended coverage*, denoted $cov^{ex}(J^*)$ where J^* is a set of job replicas, as a union of coverages for job replicas in J^* . That is:

$$\forall J_{i,r} \in J^*, \forall J_{j,q} : J_{j,q} \in cov(J_{i,r}) \implies J_{j,q} \in cov^{ex}(J^*)$$

A special case of extended coverage is an extended coverage of some batch.

Notation 11. We define a *batch coverage* as an extended coverage of some batch. We will denote a batch coverage $cov^{ex}(Batch(J_{i,r}))$ by $B-cov(J_{i,r})$.

It is worth noting that a batch coverage, unlike the original coverage, contains job replicas that are scheduled on more than one processing unit. This leads us to a simple observation.

Observation 13. As the original coverage can have up to $\mathcal{L} - 1$ elements, the batch coverage can have up to $(\mathcal{L} - 1) \cdot b_s$ elements, where b_s is size of the batch (i.e. number of distinct job replicas in given batch). That is $(\mathcal{L} - 1) \cdot m$ elements in the worst case.

Since we have observed that a batch is executed or rejected as a whole in Observation 4, it is obvious that probability of execution for any batch can be very small, as any of the job replicas in its batch coverage can cause a rejection of the entire batch.

Notation 12. Let us denote by $blanket^{ex}(J_{i,r})$ an *extended blanket* of job replica $J_{i,r}$.

The extended blanket is a set of all job replicas having an influence on execution probability for given job replica in a schedule with replication and zero time lags. We will again start by formulation the requirements for such a set.

Since in the environment with only one processing unit, the blanket would be what we seek, it has to hold:

Observation 14. $blanket(J_{i,r}) \subseteq blanket^{ex}(J_{i,r})$

The probability of execution for any job in a batch is also influenced by execution or rejection of all previous batches containing replica of given job.

Observation 15. $\forall i \in \{1, \dots, n\}, \forall r \in \{1, \dots, max\} : cov^{ex}(\bigcup_{q=1}^r Batch(J_{i,q})) \subseteq blanket^{ex}(J_{i,r})$

That is: $\bigcup_{q=1}^r B-cov(J_{i,q}) \subseteq blanket^{ex}(J_{i,r})$

We also have to mimic the recursion present in the original blanket.

Observation 16. $\forall J_{j,q} \in blanket^{ex}(J_{i,r}), \forall J_{k,s} \in blanket^{ex}(J_{j,q}) : J_{k,s} \in blanket^{ex}(J_{i,r})$

Thus from Observations 14 to 16 we get:

Definition 22. We define *extended blanket* for job replica $J_{i,r}$ as a set of all job replicas $J_{j,q}$ such that:

1. $\forall J_{j,q} \in B-cov(J_{i,r}) : J_{j,q} \in blanket^{ex}(J_{i,r})$
2. $\forall r_1 < r : Batch(J_{i,r_1}) \in blanket^{ex}(J_{i,r})$
3. $\forall J_{j,q} \in blanket^{ex}(J_{i,r}), \forall q_1 \leq q : J_{j,q_1} \in blanket^{ex}(J_{i,r})$
4. $\forall J_{j,q}, \forall J_{k,s} \in B-cov(J_{j,q}) : J_{j,q} \in blanket^{ex}(J_{i,r}) \implies J_{k,s} \in blanket^{ex}(J_{i,r})$

5. the cardinality of the set is minimal.

The best upper bound (the tightest one) on the number of elements in an extended blanket is the number of jobs present in the schedule.

We justify this claim by a counter example shown in Figure 9. In that schedule, the $\text{blanket}^{ex}(J_8)$ contains all the other jobs.

This means that in the worst-case scenario, the computation of execution probability over extended blanket has the same complexity as computing the execution probability by brute force.

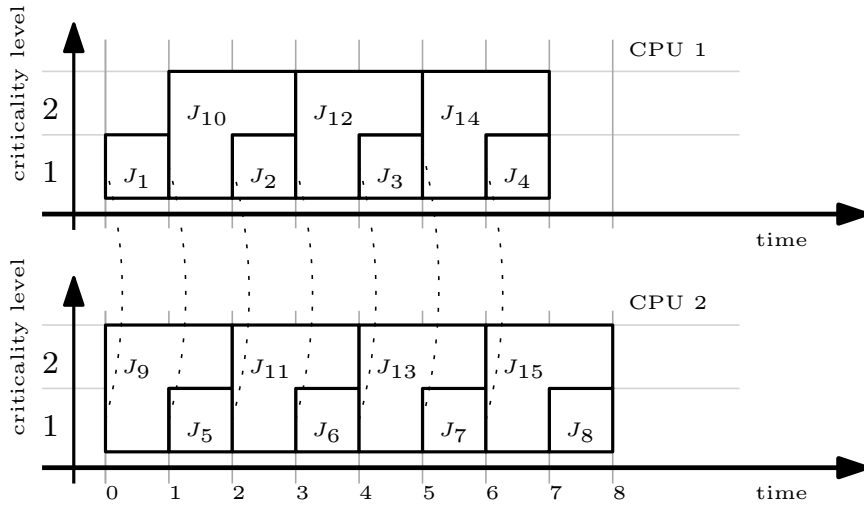


Figure 9: An example with extended blanket of size equal to n . Dotted lines represent zero time lags.

However, since the counter example we have used in Figure 9 was made just to show the worst case, we expect that in practice such schedules are rare as it is a very artificial instance of the problem.

We have defined the extended blanket of any job replica $J_{i,r}$, so that it consists of all and only of all job replicas influencing the probability of execution for $J_{i,r}$. We have not succeeded in finding any easier way to compute the execution probabilities, then to compute it by brute force computation over the extended blankets. The brute force in this case means that we evaluate the probability for every possible scenario over the jobs in the extended blanket, while ignoring all job replicas that are not present in it. Such a computation can be done by a recursive function. A pseudocode for this function is provided as Algorithm 2.

We should highlight the fact that the number of such scenarios is at worst exponential

to the number of job replicas in the extended blanket. This follows from the definition of the scenario that was introduced previously.

The input of the function for computation of execution probability are the following values:

- $J_{i,r}$ - a job replica we compute execution probability for
- $blanket$ - $blanket^{ex}(J_{i,r})$ as a list, sorted by start times of its members in such a way, that the $J_{i,r}$ is the last element of this list.
- cml - A boolean value indicating whether we want to accumulate the execution probability or not. If true, we compute the sum of execution probabilities for all replicas $J_{i,r_1}, r_1 \leq r$. If false we compute only the execution probability for the $J_{i,r}$.
- $probSoFar$ - A probability of a scenario we are currently evaluating. This value is used to accumulate value of execution probability and should be initialized with 1.
- $Btch_{rej}$ - A set of batches that are rejected in given scenario. Should be initialized with an empty set.
- $Reps_{rej}$ - A set of replicas rejected in given scenario. It should be initialized with an empty set as well.

This function goes through the given blanket and computes the execution probability for the job replica $J_{i,r}$ (or for all the replicas J_i at once, if in cumulative mode). To do this, we traverse the extended blanket and at every job replica we recursively follow all possible sub-scenarios for each level of this job and accumulate the probability of execution of $J_{i,r}$ in these recursive calls.

Once the computation is done, we can multiply the final probability by the weight assigned to given job. This multiplication gives us the contribution of this job to the criterial function. By computation of this value for every job replica in the schedule and summing it up, we obtain the value of the criterial function.

In the Algorithm 2 a function ISREJECTED is called. This function is a simple method that verifies whether a job replica is or isn't rejected within given scenario. To compute this effectively we use the aforementioned sets $Btch_{rej}$ and $Reps_{rej}$. In these variables we store the information about the batches and single job replicas that must be rejected in given scenario. Using these, we can easily make decision about a rejection of any job replica. The pseudocode for the function ISREJECTED is to be seen in Algorithm 3.

Algorithm 2 Probability of Execution - Recursive Computation

```
1: function GETPROB( $j_{i,r}, cml, blanket, probSoFar, Btch_{rej}, Reps_{rej}$ )
2:   if  $blanket.isEmpty$  then
3:     return 0 ▷  $J_{i,r}$  isn't executed in this scenario, so we backtrack
4:   end if
5:   if ISREJECTED( $j_{i,r}, Btch_{rej}, Reps_{rej}$ ) then
6:     return 0 ▷  $J_{i,r}$  is rejected in this scenario, so we backtrack
7:   end if
8:   if ISREJECTED( $blanket.head, Btch_{rej}, Reps_{rej}$ ) then
9:     ▷ current head of the extended blanket is rejected, skip it
10:    return  $getProb(j_{i,r}, cml, blanket.tail, probSoFar, Btch_{rej}, Reps_{rej})$ 
11:  end if
12:  if  $blanket.head = j_{i,r}$  then
13:    ▷  $J_{i,r}$  is last in blanket, thus cannot be rejected, once reached.
14:    ▷ So the probability of its execution is equal to probability of this scenario
15:    return  $probSoFar$ 
16:  end if
17:  if  $blanket.head = j_{i,r_1} \wedge r_1 < r$  then
18:    if  $cml$  then
19:      ▷ If we want to include all the previous replicas, then whenever we reach
20:      return  $probSoFar$ 
21:    else
22:      ▷ Else, we backtrack, since the following replicas are rejected
23:      return 0
24:    end if
25:  end if
26:   $s_r \leftarrow \emptyset$  ▷ Set of newly rejected replicas
27:   $s_{rb} \leftarrow \emptyset$  ▷ Set of newly rejected batches
28:   $p \leftarrow 0$  ▷ Sum of execution probability over possible scenarios
29:  for all  $l \in \{1, \dots, \chi_i\}$  do ▷ For each criticality level
30:     $s_r \leftarrow s_r \cup \{j_{j,q}, c_{j,q,i,r} = l\}$  ▷ We reject all replicas covered at this level
31:     $s_{rb} \leftarrow s_{rb} \cup \{Batch(j_{j,q}) : c_{j,q,i,r} = l\}$  ▷ And we reject all coresponding batches
32:    ▷ We sum the probability of execution of  $J_{i,r}$  over all possible subscenarios, i.e.
33:     $p \leftarrow p + getProb(j_{i,r}, cml, blanket.tail, B_{i,r,l} \times probSoFar,$   

over the rest of the blanket  

 $Btch_{rej} \cup s_{rb}, Reps_{rej} \cup s_r)$ 
34:  end for
35:  return  $p$  ▷ We return the probability of execution over scenarios in this blanket
36: end function
```

Algorithm 3 Function for decision about rejection of job replicas

```

1: function ISREJECTED( $j_{i,r}$ ,  $Btch_{rej}$ ,  $Reps_{rej}$ )
2:   if  $Reps_{rej}$  contains  $j_{i,r}$  then
3:     return true  $\triangleright j_{i,r}$  is rejected directly, by some member of its coverage
4:   else if  $\exists Batch(j_i, r) \in Btch_{rej}$  then
5:     return true  $\triangleright j_{i,r}$  is rejected, since it is a member of some rejected batch
6:   else
7:     return false  $\triangleright$  otherwise  $j_{i,r}$  is not rejected
8:   end if
9: end function

```

5 Scheduling Algorithms

So far, we have stated the problem, introduced some concepts for working with it and described the criterial function. In this section we propose scheduling algorithms to solve instances of such problem. We start by formulating Mixed-integer linear program (MILP), for finding a feasible solution. Then we propose a baseline solution based on Simulated Annealing. We also describe a faster algorithm, namely an algorithm called Iterative Resource Scheduling Algorithm (or IRSA for short).

5.1 Mixed-Integer Linear Program

We formulate the model for finding a feasible schedule for an instance of our problem as a MILP in Algorithm 4. It also serves as a straightforward way to summarize the constraints that we require to hold for every feasible schedule.

First three constraints (Line 3 to Line 5) in Algorithm 4 ensure that no two F-shapes overlap. The Line 6 and Line 7 formulate the constraints implied by the time lags. Line 8 is a constraint that ensures that every job is assigned to some processing unit.

Since the constraints at Line 4 and Line 5 have to hold only conditionally, for they depend on job precedence and assignment to processing units, we have to use a supporting variables that can relax these constraints when they are not required to hold. To keep the formulation of the program linear, we had to use two linearizations of these supporting variables and we describe their meaning in the next paragraph.

We use binary variable $x_{i,j,r,q}$ to indicate that job replica $J_{i,r}$ is scheduled to start earlier or at the same time as $J_{j,q}$. (If $x_{i,j,r,q} = 1$ then it is true). Similarly we use variable $v_{i,r}$ to indicate that the job replica $J_{i,r}$ is scheduled. Variable $y_{i,j} = 1$ if, and only if, the jobs J_i and J_j are scheduled onto a common processing unit. Variable $\mu_i^*(k)$ indicates whether the job J_i is scheduled onto processing unit \mathcal{M}_k . This means that we require it to hold:

$$\forall i \in I, \forall k \in K : \mu_i^*(k) = 1 \implies \mu_i = \mathcal{M}_k$$

And finally, $w_{i,j,r,q} = 1$ whenever job replicas $J_{i,r}, J_{j,q}$ are both scheduled, and $J_{i,r}$ starts earlier or at the same moment as $J_{j,q}$. We also use a large constant M for conditional relaxation of some constraints. For the program to work, we expect the value of M to be larger than sum of all processing times and absolute values of time lags.

We do not provide the formulation of objective function as part of this MILP, instead we leave it as an open problem.

Algorithm 4 MILP for Finding Feasible Schedule with F-shapes

Require: For scope of this MILP it holds:

- $$I = \{1, \dots, n\}; R = \{1, \dots, max\}; K = \{1, \dots, m\}$$
- 1: min: \emptyset
 - 2: s.t:
 - 3: $s_{i,r} \geq 0$ $\triangleright \forall i \in I, \forall r \in R$
 - 4: $s_{i,r} + p^{(\chi_i)} \leq s_{i,q} + M \times z_{i,i,r,q}$ $\triangleright \forall i \in I, \forall q \in R \setminus \{1\}, \forall r \in R \setminus \{max\}, r < q$
 - 5: $s_{i,r} + p_i^{(min(\chi_i, \chi_j))} \leq s_{j,q} + M \times (2 - w_{i,j,r,q} - y_{i,j})$ $\triangleright \forall i, j \in I, \forall r, q \in R$
 - 6: $s_{i,max} + l_{i,j} \leq s_{j,1}$ $\triangleright \forall i, j \in I, l_{i,j} \neq 0$
 - 7: $s_{i,r} = s_{j,r}$ $\triangleright \forall i, j \in I, \forall r \in R, l_{i,j} = 0$
 - 8: $\sum_{k \in K} \mu_i^*(k) = 1$ $\triangleright \forall i \in I$
 - 9: $w_{i,j,r,q} \in \{0, 1\}$ $\triangleright \forall i, j \in I, \forall r, q \in R$
 - 10: $x_{i,j,r,q} \in \{0, 1\}$ $\triangleright \forall i, j \in I, \forall r, q \in R$
 - 11: $y_{i,j} \in \{0, 1\}$ $\triangleright \forall i, j \in I$
 - 12: $\mu_i^*(k) \in \{0, 1\}$ $\triangleright \forall i \in I, \forall k \in K$
 - 13: $z_{i,j,r,q} \in \{0, 1\}$ $\triangleright \forall i, j \in I, \forall r, q \in R$
 - 14: $v_{i,r} \in \{0, 1\}$ $\triangleright \forall i \in I, \forall r \in R$
 - 15: $w_{i,j,r,q} + 1 \geq z_{i,j,r,q} + x_{i,j,r,q}$ $\triangleright \forall i, j \in I, \forall r, q \in R$
 - 16: $w_{i,j,r,q} \leq z_{i,j,r,q}$ $\triangleright \forall i, j \in I, \forall r, q \in R$
 - 17: $w_{i,j,r,q} \leq x_{i,j,r,q}$ $\triangleright \forall i, j \in I, \forall r, q \in R$
 - 18: $x_{i,j,r,q} + x_{j,i,q,r} = 1$ $\triangleright \forall i, j \in I, i \neq j, \forall r, q \in R$
 - 19: $x_{i,i,r,r} = 1$ $\triangleright \forall i \in I, \forall r \in R$
 - 20: $z_{i,j,r,q} + 1 \geq v_{i,r} + v_{j,q}$ $\triangleright \forall i, j \in I, \forall r, q \in R$
 - 21: $z_{i,j,r,q} \leq v_{i,r}$ $\triangleright \forall i, j \in I, \forall r, q \in R$
 - 22: $z_{i,j,r,q} \leq v_{j,q}$ $\triangleright \forall i, j \in I, \forall r, q \in R$
 - 23: $y_{i,j} \geq \mu_i^*(k) + \mu_j^*(k) - 1$ $\triangleright \forall k \in K$
 - 24: $v_{i,r} \geq v_{i,r+1}$ $\triangleright \forall i \in I, \forall r \in R \setminus \{max\}$
 - 25: $v_{i,1} = 1$ $\triangleright \forall i \in I$
-

5.2 Simulated Annealing

The first heuristics we proposed and tested is based on Simulated annealing. We have chosen the Simulated annealing for initial solution as it is a relatively simple algorithm, to provide a baseline performance.

The Simulated Annealing algorithm is a local search algorithm introduced by Kirkpatrick et al. in [13]. As the name of the algorithm hints, it is based on simulation of the annealing process. The "annealed thing" is a solution to the problem it is applied on. A pseudocode for this algorithm is well described by S. Luke [15] and we consider this algorithm to be well known, so we do not mention the general form here. We only provide a pseudocode for the specific adaptation of the algorithm that we have used, see Algorithm 5. The Simulated Annealing usually works with the concept of genome, which is supposed to represent a condensed form of a solution. To describe the proposed algorithm fully, we specify how we implemented the methods called by the Simulated Annealing and some other decisions we have made.

In our case, the genome is an array of integer values, with length of $2n$. The genome consists of ordered list of job indexes and assignment of the jobs to processing units. The encoding is done so, that at positions $genome(i_0)$, where $(i_0 \bmod 2 \equiv 0)$, we store ordered indexes of the jobs and on positions $(i \bmod 2 \equiv 1)$ we store values of $\mu_{genome(i-1)}$. For every solution that we obtain from its genome, it has to hold that if we sort the start times of jobs first replicas by the ordering of job indexes encoded in corresponding genome, we get a non decreasing sequence.

A solution, either temporal or final, is a schedule, represented by data structure with four maps. The first map represents an assignment of jobs to processing units, second map holds information about how many replications were scheduled for each job, third map assigns a starting time to every job replica and the last one maps every job replica on its coverage. It is obvious that only the first and the third map are needed to reconstruct the schedule, as the rest of the information can be computed from these two, but for sake of speed and implementation convenience, we keep the other two maps in the memory as well.

The transformation of the genome to the schedule is done by a dedicated method. Obtaining the first map from the genome is very straightforward as the information is already present in the genome. Once this extraction is done, we compute relative time windows. That is, we combine the job ordering encoded in the genome with relative temporal constraints (i.e the time lags) and we get for each job a set of intervals. These intervals define where replicas of given job can be scheduled, relative to other job replicas. How to compute these relative windows by a Floyd–Warshall algorithm is shown in [6].

We create a distance graph for our instance of the problem. In [6] the authors define a distance graph as a graph $G_d = (V, E_d)$ where each edge $i \rightarrow j$ has a weight a_{ij} representing linear inequality $X_j - X_i \leq a_{ij}$. There, the symbols X_j, X_i represent variables used for time

Algorithm 5 Simulated Annealing

```
1:  $t \leftarrow 10^9$  ▷ initial temperature
2:  $\tau \leftarrow 0$  ▷ time since last improvement
3:  $g \leftarrow$  random generated initial genome
4:  $S \leftarrow g$  transformed to initial solution
5:  $BestSoFar \leftarrow S$ 
6: repeat
7:    $g_1 \leftarrow \text{Tweak}(\text{Copy}(g))$ 
8:    $R \leftarrow g_1$  transformed to solution
9:   ▷ let  $r$  be a random number  $r \in [0, 1]$ 
10:  if  $\text{Quality}(R) > \text{Quality}(S) \vee r < e^{\frac{\text{Quality}(R) - \text{Quality}(S)}{t}}$  then
11:     $S \leftarrow R$ 
12:     $g \leftarrow g_1$ 
13:  end if
14:   $\text{Decrease}(t)$ 
15:  if  $\text{Quality}(S) > \text{Quality}(BestSoFar)$  then
16:     $BestSoFar \leftarrow S$ 
17:     $\tau \leftarrow 0$ 
18:  else
19:     $\text{increment}(\tau)$ 
20:  end if
21: until ( $BestSoFar$  is an optimal solution ▷ an optimal solution is a solution where  $\forall J_i : P_i = 1$ 
▷ we are stuck in local optima
▷ the temperature has dropped to zero
or  $\tau \geq 1000$ 
or  $t \leq 0$ )
22: return  $BestSoFar$ 
```

assigned to nodes. To apply this on our problem, we use the constraints defined at Line 6 and Line 7 of Algorithm 4. Thus we create a distance graph $G^* = (V^*, E^*)$. A set of nodes V^* will contain one node for each scheduled job replica $J_{i,r}$. We require that for every two job replicas $J_{i,r}, J_{j,q}$ the edge from node $v(i, r)$ to node $v(j, q)$ has weight $-l_{i,j}$, i.e. it satisfies the constraint $s_{i,r} - s_{j,q} \leq -l_{i,j}$ for all $l_{i,j} \neq 0$ and constraint $s_{i,r} - s_{j,q} = 0$ otherwise.

Whenever a genome represents an infeasible ordering, we detect it from the results of the Floyd-Warshall's algorithm, since the path from any node to itself has to have a zero length for any feasible ordering.

We create an instance of a schedule using the relative time windows. At first, we schedule one replica of each job, then, if some relative time windows are still large enough to allow for job replication, we replicate. Scheduling of the first replica is done greedily, that is, we find interval in which the replica can be scheduled, using the already scheduled replicas and the relative time windows, and we make a rough estimate of execution probability based solely on coverage the job would have, if we placed it on given spot present in the intersections of all such relative time windows. This estimated value is computed by taking into account only the jobs that are covering given spot and its value is computed using the formula from Proposition 2. It is only an estimation of the real execution probability since it completely ignores the time lags and replications.

Once this estimation is done for all possible spots, we choose the best spot and use it. When all jobs have at least one replica scheduled, we have a valid solution, but to increase the value of criterial function, we try to replicate the job for which the estimate of weighted probability of execution is lowest.

The other choices we have done regarding the implementation of the simulated annealing are the following:

- We decrease the the temperature in each iteration of the simulated annealing by multiplying it by value $\alpha = 0.9$.
- *Tweak* is a method implemented so that it either changes assignment of some job to a different processing unit with probability of 0.5 or it swaps two jobs in the ordering, but the assignment to processing units remains the same for each job.
- A quality of any solution is defined as a value of the criterial function for the schedule.
- To achieve better results we decided to run the algorithm with 11 restarts. To save time, we use 6 worker actors, and thus running 6 restarts in parallel.

The algorithm, as shown in experiments Sections 6.3 and 6.4 is able to solve simple instances, but is neither fast nor powerful enough to solve instances with number of jobs

higher than 80 tasks within reasonable time limits. Especially, if the number of time lags present in the instance is high.

5.3 Iterative Resource Scheduling Algorithm

The second algorithm we propose is Iterative Resource Scheduling Algorithm (or IRSA for short) that was originally introduced in [11]. We have modified this algorithm to solve our problem efficiently, implemented it and tested. We present the results of the experiments in Section 6.

Originally, the algorithm was designed for finding schedules with minimal makespan. Before we describe our modification we provide a brief description of the original version.

It starts with some loose upper and lower bound, an initial budget ratio and a priority of the scheduled jobs. The budget ratio specify how big budget the algorithm has to solve an instance with given number of jobs. During its run, the algorithm seeks a solution for given combination of boundaries and priorities and improves these value depending on success or failure. The solution is sought after by a method, that is allowed to do at only certain number of steps, that is limited by the budget. The algorithm seeks for better solution, until the boundaries converge to the same value or fails. The algorithm fails if all newly generated priorities where already used.

The initial priority of each job is defined as a length of a path in distance graph from given job to a dummy ending job, that has to be scheduled as the very last. Whenever a feasible solution is found within a specific number of steps, the upper bound is decreased and if no feasible schedule is found, the lower bound is increased. The full pseudocode for the original algorithm is to be found in [11].

In summary, we proposed the following modifications. We have modified the parameters of the algorithm since we do not need an upper bound on schedule makespan but instead we need to know how many replicas of each job have been scheduled already. For this purpose we have implemented a simple class that contains current priority and number of scheduled replicas for each job.

The initial priority setup for each job is defined as the length of the shortest path from given job replica to a dummy ending task and the initial number of replicas is one per job.

The IRSA loads an instance of the problem, and creates an initial setup. Then we compute distance graph (in the same way as in Section 5.2), we increment the number of enqueued tasks and call an *enqueue* method.

The *enqueue* method computes a hash for given setup. If we have already computed solution for such setup, we dequeue this task. Else, we send a message to worker actor to

find solution for given combination of instance and setup and we also provide the distance graph to it. If the number of enqueued tasks is zero, the run for this instance is finished. Whenever the worker actor is done with its processing of the message, it sends asks the master actor to *dequeue*.

The *dequeue* decreases the number of enqueued tasks and creates two new setups stp_1, stp_2 as copies of the original setup. If the worker actor was successful and we obtained a feasible schedule, we compute its fitness (i.e. the value of criterial function). If the fitness is better then fitness of the best solution we know so far, we set this solution to be a new best solution. And we modify stp_2 , by increasing number of replications for a job with the lowest weighted execution probability that has less then *max* replicas. Then we modify the stp_1 by changing the priorities so that every job has a priority set to value equal to difference between an end of the schedule and start time of its first replica. If the worker actor was not successful in finding a feasible schedule, we decrease the number of replicas in stp_2 for one or two jobs that caused the highest number of collisions, depending on how many unsuccessful tasks we have encountered. After we have solved this, we swap priorities for the two most colliding jobs and call *enqueue* on this instance with both setups stp_1, stp_2 .

The pseudocode for *enqueue* and *dequeue* methods is provided in Algorithms 6 and 7.

Algorithm 6 IRSA Master Actor Enqueue Method

```

1: function ENQUEUE(Instance, Setup,  $G^*$ )
2:   if computed contains (Setup.hash) then
3:     enqueued -= 1                                     ▷ this setup was already used
4:   else
5:     ask some workerActor to COMPUTESCHEDULE(Instance, Setup,  $G^*$ )
6:     computed += hash of Setup       ▷ we add this hash to the set of already used
7:   end if
8:   if enqueued = 0 then
9:     return BestSolution                               ▷ in this case, we are done
10:  end if
11: end function

```

The worker actor executes a complex algorithm for finding a schedule under given setup. At first it determines a *budget* for finding the solution. The *budget* is set to be equal to a sum of all job replicas required to be present in the schedule, multiplied by the *budgetRatio*. Then we run a tail recursive function that attempts to build the schedule from the specifications within given budget. This function does the following.

If the budget is exhausted, and some job has no replica scheduled, we return an infeasible solution. Else, if the budget is over and every job has at least one scheduled replica, we return the schedule we have build so far. If the budget is not over yet, we find a job with lowest number of replicas and highest priority and we try to find first time slot at a

Algorithm 7 IRSA Master Actor Dequeue Method

```
1: function DEQUEUE(Instance, Setup,  $G^*$ , Solution)
2:   enqueued += 1
3:    $stp_1, stp_2 \leftarrow$  copies of Setup
4:   if Solution is feasible then
5:     Fit  $\leftarrow$  fitness of Solution
6:     if Fit > fitness of BestSolution then
7:       BestSolution  $\leftarrow$  Solution
8:        $\triangleright$  we increase the number of replicas for a job with the lowest weighted execution
9:       probability, that has number of replicas < max
10:      stp_2.increaseReplications(Setup, WeightedExecutionProbs)
11:    end if
12:    stp_1.recomputePriorities(Solution)  $\triangleright$  use priorities corresponding to the
13:    ordering of first replicas in the solution
14:  else
15:    FailureCounter += 1
16:    if FailureCounter  $\geq \log(n * \text{max})$  then
17:       $\triangleright$  we decrease replications for second most conflicting job
18:      stp_2.decreaseReplicationsForSecond(Solution.conflicts)
19:      FailureCounter = 0
20:    end if
21:     $\triangleright$  we decrease replications for the most conflicting job
22:    stp_2.decreaseReplicationsForFirst(Solution.conflicts)
23:  end if
24:   $\triangleright$  swap priorities for two most conflicting jobs in stp_1 and stp_2
25:  stp_1.swapMostConflicting(Solution.conflicts)
26:  stp_2.swapMostConflicting(Solution.conflicts)
27:   $\triangleright$  and we enqueue these two setups
28:  enqueued += 2
29:  ENQUEUE(Instance, stp_1,  $G^*$ )
30:  ENQUEUE(Instance, stp_2,  $G^*$ )
31: end function
```

processing unit to schedule the job replica to. If a job is in bounded by some zero time lags, then entire batch is scheduled together. The processing unit is chosen when first replica of the job is being scheduled. All other replicas of the same job are scheduled onto the same processing unit. If a time slot is found, we schedule the job replica to it. If no such time slot is found, we schedule the job replica to the first time slot that is not violating any positive time lag, then we find the source of conflicts and remove the conflicting jobs from the schedule. Finally, we do the recursion call. A pseudocode for this method is provided in Algorithm 8

The method that is used for finding a time slot to schedule job replica to is greedy. That is, it finds the first spot in time on a corresponding processing unit (or first time slot at any processing unit for first replica) where we can schedule this F-shape without overlapping some other.

5.4 Implementation

The proposed algorithms were implemented in Scala language, using the Akka framework⁵ for parallelization. Regarding the IRSA algorithm, we have implemented parallel version. The main loop of the IRSA algorithm is implemented as a master actor and the method for finding schedules within given budget is run in parallel by worker actors. The entire Scala project containing both algorithms is considered to be part of this thesis and is provided on the attached CD. It can be builded and run using Scala Build Tool (SBT).

⁵<https://akka.io/>

Algorithm 8 IRSA Worker Actor Method for Finding a Schedule

```

1: function COMPUTESCHEDULE(Instance, Setup,  $G^*$ )
2:   Conflicts  $\leftarrow$  an empty Map
3:   AllReplicas  $\leftarrow \{J_{i,r} | \forall i, r \in \{1, \dots, \text{Setup.numReplicas}(i)\}\}$ 
4:   function FINDSCHEDULE(AlreadyScheduled, Budget, CurSchedule)
5:     if Budget = 0  $\wedge \exists J_{i,1} \notin \text{AlreadyScheduled}$  then
6:       return Solution(EmptySchedule, Conflicts)
7:     end if
8:     if Budget = 0  $\vee$ 
9:        $\forall J_i$  number of replicas in CurSchedule corresponds to Setup then
10:      return Solution(CurSchedule, Conflicts)
11:    end if
12:    candidates  $\leftarrow \text{AllReplicas} \setminus \text{AlreadyScheduled}$ 
13:     $J_{i_c, r_c} \leftarrow$  pick one from candidates with lowest  $r_c$  and highest priority
14:    LB  $\leftarrow \text{getLowerBound}(G^*, \text{AlreadyScheduled}, J_{i_c, r_c})$   $\triangleright$  returns a lower
    bound on where the replica can be scheduled. This lower bound is implied by positive
    time lags from already scheduled replicas.
15:    Slot  $\leftarrow \text{findTimeSlot}(\text{CurSchedule}, G^*, \text{LB}, J_{i_c, r_c})$ 
16:    if Slot.isEmpty then
17:      Slot  $\leftarrow \text{LB}$ 
18:    end if
19:    CurSchedule.addToSchedule( $J_{i_c, r_c}, \text{Slot}$ )
20:    Conf  $\leftarrow \text{findConflicting}(\text{CurSchedule}, J_{i_c, r_c}, \text{Slot})$ 
21:    for all  $J_{i,r} \in \text{Conf}$  do  $\triangleright$  For each conflicting job
22:      Conflicts( $J_i$ ).increase
23:      CurSchedule.unschedule( $J_{i,r}$ )
24:      AlreadyScheduled  $\setminus \{J_{i,r}\}$ 
25:    end for
26:    AlreadyScheduled.add( $J_{i_c, r_c}$ )
27:    FINDSCHEDULE(AlreadyScheduled, Budget - 1, CurSchedule)
28:  end function
29: return FINDSCHEDULE(emptySet, budget, emptyschedule)
30: end function

```

6 Computational Experiments

In this section, we provide description and results from experiments we have conducted. We demonstrate that the proposed algorithms have the capacity to solve instances of the problem at hand and we show their limitations.

6.1 Criterion Computation Given a Schedule

We start this section by a demonstration of the speedup in execution probability computation when we use the blanket instead of computing the value from definition. To compare the times required for the computation, we have randomly generated 200 schedules for each value of $n \in \{11, 12, 13, \dots, 23\}$. Then we have computed the value of criterial function for each such schedule by brute force and we measured the time we needed. The results are shown in Figure 10. The amount of time was prohibitively high even for instances containing as little as 23 instances. For instances of this size we needed over 5 seconds in more than a half of all the runs.

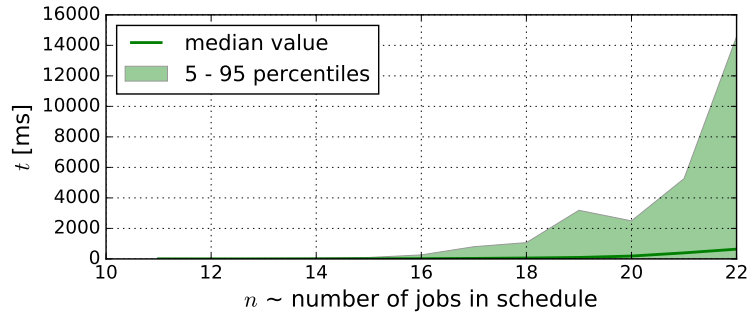


Figure 10: Time required for criterion computation over multiple instances by brute force

Similarly, we have generated 200 schedules for each value of $n \in \{10, 20, 30, \dots, 240\}$ and we have measured the time needed to compute criterial value for these schedules using the computation over blankets. As can be seen in Figure 11, we were able to compute the criterial value for instances containing as many as 210 jobs in less then 0.5 seconds for more than 95% of all the cases.

Therefore, we claim that the speedup gained by the computation of the criterial value over the blankets is significant.

These two tests were run on the same machine with an Intel Core i7-2640M @ 2.8GHz CPU. The maximal number of replicas per job was limited to $max = 5$ in both cases.

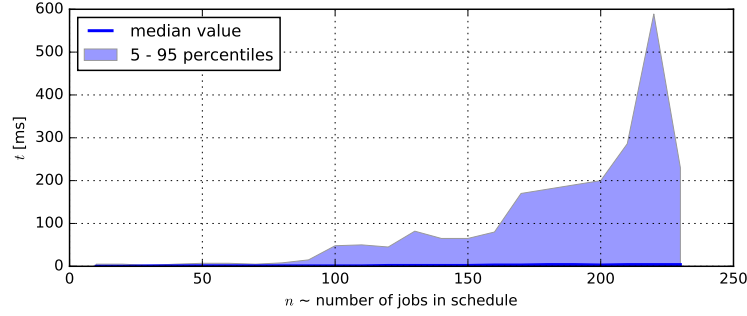


Figure 11: Time required for criterion computation over multiple instances using blankets

6.2 Instances of $P|temp, mc = 3, mu| \sum_i w_i P_i$

For the following experiments we have generated a data set composed of randomly generated instances with various parameters. The first parameter is the number of processing units m . A second parameter is number of jobs n . The last parameter is the number of time lags present in the instance. Instead of using fix values for all the instance sizes, we use number of time lags proportionate to the number of jobs. We therefore define *time lag ratio* as a number of time lags present in an instance with n jobs. This means that time lag ratio is defined as a ratio between the number of time lags that are present in the instance and the value of n .

For each combination of $m \in \{2, 3, 4\}$, $n \in \{10, 20, 30, \dots, 90\}$ and the time lags ratios $\{0.1 \cdot n, 0.2 \cdot n, 0.6 \cdot n\}$ we have generated 20 random instances. The weights of the jobs were set to 1, so that optimal value of criterial function is always equal to the number of jobs. This makes evaluation of results easier.

Processing times for jobs are generated using the following rule. The processing time for first criticality level is a random integer value taken from a uniform distribution over interval from 1 to 5 inclusive. If the job has more than one criticality level, then every higher level has processing time equal to the previous one plus a random value generated in the same manner.

For any job J_i , we initialize the values of $B_{i,k}$ as $\forall k \in \{1, \dots, \chi_i\} : B_{i,k} = \beta_{i,k} / \sum_{k=1}^{\chi_i} \beta_{i,k}$, where $\beta_{i,k}$ is a random double value from interval $[0.1, 1.1)$.

Regarding the budget ratio used by an IRSA algorithm, we set it to 8 as is proposed in the original formulation of IRSA in [11].

For each such instance, we have then evaluated both algorithms with and without replication and we have logged three measurements. We have measured the time required by the algorithm, fitness of the solution (the value of criterial function) and whether or not is the solution feasible.

The reason why we have used these values and why we haven't tested it on other parameter setups is the time required for the run of all the tests. Although the run times per instance were not intractable, the time for running all these tests was relatively high and for more complex instances it would grow further. Thus we have made the choice to test the algorithm on a larger set of easier instances, than to show few results for complex ones.

6.3 Success Rate

The first question, with regard to the proposed solution, is what size of instances are these algorithms able to solve. To answer this question, we have run both algorithms on the generated instances and we logged whether the algorithm succeeded or not. The aggregated results are presented in Tables 1 and 2.

There, the success rate is computed as a percentage of successful runs per each combination of m, n and the time lag ratio to n . From these data, we see that the number of jobs to be scheduled is problematic mainly in the situations where the the number of time lags is high. This is a behaviour that is expected, as the higher number of time lags creates a higher number of constraints on the positions of the jobs. In such cases the space of feasible solutions shrinks and finding a feasible solution becomes harder.

We can also see that the IRSA algorithm is better at finding a feasible solution. Also, it seems that a higher number of processing units makes it easier for the Simulated Annealing to find a feasible solution. However, to prove this claim we would need to test it more thoroughly.

6.4 Time Requirements

The second criterion for effectiveness of the proposed solution, is the amount of time required by these algorithms. To see how fast the algorithms are in practice, we provide the measured run times. Since this value is dependent on the machine on which the algorithm is run, we first mention its specification. The algorithms were run on a server with two Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz processors and were allowed to use at most 12GB of ram. The server is equipped with DIMM DRAM Synchronous RAM, working at 2400 MHz. We allowed each run to use at most two cores of this processor.

We have aggregated the measured values into Tables 3 and 4 as follows. For every combination of m, n and the time lag ratio we provide three values representing 5th, 50th and 95th percentile. These percentiles are computed over the twenty instances that we have generated for such combination. Both algorithms were run on the same instances to allow for comparison of the required times.

The evaluation shows us that the time consumption of the Simulated Annealing is relatively high, as it needs more than one hour to finish for the large instances with time lag ratio of $0.6 \cdot n$. And as we have seen in the previous section it still fails to find a feasible solution. The IRSA algorithm also is relatively slow as it needs almost half an hour to finish such instances, but unlike the Simulated Annealing it finds a feasible solution. We can therefore claim that the IRSA algorithm offers a huge improvement compared to the Simulated Annealing, as it is faster and manages to solve harder instances on which the Simulated Annealing fails. Even the IRSA algorithm needs for a single run over an instance for 4 processing units and 90 jobs with only 54 temporal constraints up to half an hour to solve. However, since we haven't intended this algorithm for real-time scheduling, but for an offline proactive creation of schedules, the time demands are not a problem. Therefore we claim that the proposed algorithms are tractable and working.

6.5 Impact of Replication on Criterial Function

Finally, we demonstrate that the idea of replication is meaningful (i.e. that it has a positive impact on the probability of execution.)

To do this, we have compared average objective values of the solutions with and without replication. We have aggregated the measured values into Tables 5 and 6 as follows. For every combination of m, n and the time lag ratio we provide two values. The first one is an average fitness over instances with these parameters with replication. Second value is average fitness without when no replication is allowed.

The results show that the replication does make difference in general but not always. Also it seems that the number of processing units has almost no influence on how big improvement the replication provides. The number of time lags, on the other hand, seems to have an impact as higher number of time lags tends to diminish the benefits of replication. This does correspond to our prediction that the replication can shrink relative windows for other jobs. It also is not surprising since the higher number of time lags means that there are more constraints on feasibility of the schedule. And thus the opportunity for schedule optimization disappears.

For the time lag ratios of $0.1 \cdot n$ and $0.2 \cdot n$ the Simulated Annealing produces schedules with better objective values, but the run times are significantly higher, sometimes even by several orders of magnitude. Also, for higher time lag ratio the Simulated Annealing fails to find feasible solutions at all. We ascribe this property to the fact, that Tweak method based on job permutations samples the space of solutions well on these easier instances, but on the more constrained ones it is not enough. Also, the advantage of IRSA on the more complex instances is that it can detect infeasibility of a schedule and resolve during its creation.

Success rates of IRSA algorithm				
# time lags:		0.1 · n	0.2 · n	0.6 · n
m=2	n=10	100%	100%	95%
	n=20	100%	100%	100%
	n=30	100%	100%	100%
	n=40	100%	100%	100%
	n=50	100%	100%	100%
	n=60	100%	100%	100%
	n=70	100%	100%	100%
	n=80	100%	100%	100%
	n=90	100%	100%	100%
m=3	n=10	100%	100%	95%
	n=20	100%	100%	100%
	n=30	100%	100%	100%
	n=40	100%	100%	100%
	n=50	100%	100%	100%
	n=60	100%	100%	100%
	n=70	100%	100%	100%
	n=80	100%	100%	100%
	n=90	100%	100%	100%
m=4	n=10	100%	100%	100%
	n=20	100%	100%	100%
	n=30	100%	100%	100%
	n=40	100%	100%	100%
	n=50	100%	100%	100%
	n=60	100%	100%	100%
	n=70	100%	100%	100%
	n=80	100%	100%	100%
	n=90	100%	100%	100%

Table 1:
Success rates for IRSA algorithm

Success rates of Simulated Annealing				
# time lags:		0.1 · n	0.2 · n	0.6 · n
m=2	n=10	100%	100%	95%
	n=20	100%	100%	95%
	n=30	100%	100%	95%
	n=40	100%	100%	60%
	n=50	100%	100%	25%
	n=60	100%	100%	5%
	n=70	100%	100%	5%
	n=80	100%	100%	0%
	n=90	100%	90%	0%
m=3	n=10	100%	100%	100%
	n=20	100%	100%	100%
	n=30	100%	100%	100%
	n=40	100%	100%	95%
	n=50	100%	100%	60%
	n=60	100%	100%	15%
	n=70	100%	100%	0%
	n=80	100%	100%	0%
	n=90	100%	100%	0%
m=4	n=10	100%	100%	100%
	n=20	100%	100%	100%
	n=30	100%	100%	95%
	n=40	100%	100%	100%
	n=50	100%	100%	85%
	n=60	100%	100%	40%
	n=70	100%	100%	20%
	n=80	100%	100%	0%
	n=90	100%	100%	0%

Table 2: Success rates for Simulated Annealing

Runtimes of IRSA algorithm in seconds										
# time lags:		0.1 · n			0.2 · n			0.6 · n		
percentiles:		5th	50th	90th	5th	50th	90th	5th	50th	90th
m=2	n=10	0.02	0.05	0.41	0.02	0.06	0.31	0.01	0.06	0.44
	n=20	0.02	0.31	0.88	0.05	0.31	0.94	0.04	0.23	1.27
	n=30	0.09	0.47	2.06	0.12	0.81	1.77	0.13	0.86	10.41
	n=40	0.2	1.08	4.09	0.1	0.78	7.93	0.35	2.78	12.96
	n=50	0.2	1.59	11.09	0.19	1.06	17.52	0.76	3.13	1182.68
	n=60	0.41	1.67	5.29	0.32	1.91	40.13	1.07	6.07	1082.15
	n=70	0.63	3.86	13.55	0.65	6.75	31.72	1.52	56.89	1166.95
	n=80	1.26	7.61	50.14	1.62	9.76	955.96	4.66	454.82	1342.71
	n=90	2.37	6.43	28.89	2.11	14.82	173.13	15.74	625.07	1163.2
m=3	n=10	0.02	0.04	0.1	0.02	0.04	0.06	0.01	0.03	0.07
	n=20	0.1	0.3	0.65	0.12	0.33	0.77	0.03	0.31	2.75
	n=30	0.14	1.49	3.97	0.08	0.82	2.59	0.55	5.29	1083.05
	n=40	0.45	3.69	13.23	0.2	1.85	10.63	0.39	20.73	1155.65
	n=50	0.35	1.93	37.03	0.21	1.87	61.38	0.74	16.61	1121.07
	n=60	0.35	2.85	49.66	1.09	6.27	1077.78	1.86	937.82	1441.45
	n=70	1.36	4.87	932.66	0.7	5.14	154.01	3.96	1017.7	1491.33
	n=80	0.86	8.02	63.27	3.09	17.22	1003.86	0.9	1173.01	1561.75
	n=90	1.62	3.56	161.47	2.93	18.21	1171.54	11.13	975.94	1440.67
m=4	n=10	0.02	0.02	0.05	0.01	0.02	0.04	0.01	0.02	0.05
	n=20	0.08	0.23	0.57	0.09	0.24	1.02	0.03	0.21	0.81
	n=30	0.19	0.81	2.65	0.15	1.02	2.42	0.25	1.9	946.74
	n=40	0.23	1.89	10.87	0.15	1.64	13.74	0.32	8.32	1170.54
	n=50	0.7	3.97	11.19	0.9	2.03	20.78	0.43	69.93	1461.29
	n=60	0.74	5.49	33.3	0.57	2.08	37.88	6.32	1039.72	1433.64
	n=70	1.2	4.27	47.07	1.07	11.13	774.85	2.54	1080.86	1442.56
	n=80	1.22	3.58	32.74	2.23	9.8	1107.76	37.44	1174.84	1481.15
	n=90	1.98	10.04	107.27	2.87	37.93	1236.3	5.19	1010.72	1463.55

Table 3: Runtimes of IRSA algorithm with replication.

For each combination of m, n and time lag ratio, there are three values representing 5th, 50th and 95th percentile respectively.

Runtimes of Simulated Annealing in seconds										
# time lags:		0.1 · n			0.2 · n			0.6 · n		
percentiles:		5th	50th	90th	5th	50th	90th	5th	50th	90th
m=2	n=10	0.07	0.19	2.34	0.07	0.33	1.72	0.4	1.92	10.34
	n=20	0.45	1.79	7.8	0.36	2.17	4.52	29.89	43.98	56.64
	n=30	2.21	11.09	26.92	7.96	19.49	35.29	162.21	178.33	198.11
	n=40	1.72	36.02	149.85	30.55	68.42	124.05	311.45	433.77	511.7
	n=50	53.67	175.15	317.48	141.42	175.0	270.53	549.18	570.64	608.93
	n=60	237.1	348.44	709.37	243.5	406.97	464.33	922.27	943.68	1118.43
	n=70	314.73	783.59	1253.82	477.27	620.01	721.86	1473.51	1533.04	1724.0
	n=80	590.44	1430.01	1765.31	895.98	1004.73	1089.57	2242.82	2343.06	2567.99
	n=90	1426.53	2413.12	3257.8	1459.99	1682.78	1844.18	3255.0	3436.63	3755.07
m=3	n=10	0.04	0.07	0.18	0.03	0.1	0.36	0.13	0.63	3.45
	n=20	0.28	1.8	5.48	0.75	2.79	11.59	16.47	27.06	42.1
	n=30	2.16	14.77	25.56	3.44	16.85	39.14	114.81	131.19	136.09
	n=40	23.87	69.42	144.75	18.57	76.0	112.27	309.46	316.73	347.74
	n=50	50.55	142.65	411.47	137.78	198.84	281.39	605.14	637.65	677.98
	n=60	250.17	492.71	681.61	307.43	430.65	557.75	1036.08	1065.32	1140.26
	n=70	319.11	1002.4	1301.44	593.57	784.63	921.06	1605.76	1718.09	1801.84
	n=80	520.88	1732.69	2603.52	933.17	1188.79	1450.41	2379.86	2466.6	2531.12
	n=90	1931.22	2764.64	3351.89	1653.24	1854.31	1984.31	3103.88	3575.54	3712.17
m=4	n=10	0.04	0.08	0.22	0.02	0.07	0.51	0.2	0.63	3.07
	n=20	0.21	1.24	6.44	0.43	1.23	7.31	9.08	23.52	36.63
	n=30	2.28	10.81	24.66	5.84	16.19	30.67	110.7	135.03	147.58
	n=40	16.98	79.87	157.62	22.2	61.07	128.02	324.19	349.05	375.64
	n=50	52.1	141.35	505.77	69.34	179.2	257.0	553.2	703.46	743.44
	n=60	83.33	361.15	709.56	267.68	364.17	490.69	919.09	936.63	958.29
	n=70	459.15	835.53	1344.62	442.86	728.6	901.04	1501.67	1519.99	1546.64
	n=80	641.77	1728.2	2461.49	726.34	985.28	1140.98	2206.45	2369.56	2653.58
	n=90	1922.18	2523.68	3419.96	1373.27	1536.5	1780.34	3828.48	4456.58	4564.08

Table 4: Runtimes of Simulated Annealing with replication.

For each combination of m, n and time lag ratio, there are three values representing 5th, 50th and 95th percentile respectively.

Solution quality for IRSA algorithm with and without replication							
# time lags:		0.1 · n		0.2 · n		0.6 · n	
replication used:		yes	no	yes	no	yes	no
m=2	n=10	10.0	9.14	9.92	9.34	9.33	9.33
	n=20	19.48	17.62	19.64	18.08	19.29	18.77
	n=30	28.11	26.37	28.92	26.81	28.61	27.99
	n=40	36.43	33.86	36.51	34.83	37.17	36.5
	n=50	44.96	41.68	44.34	42.65	45.01	44.46
	n=60	51.2	50.07	51.0	49.76	52.81	52.6
	n=70	59.85	58.03	60.32	59.01	60.74	60.39
	n=80	68.07	65.87	66.49	65.33	69.69	69.34
	n=90	73.86	73.86	74.25	74.25	78.06	78.06
m=3	n=10	10.0	9.46	10.0	9.73	9.46	9.41
	n=20	19.77	17.91	19.6	18.36	19.67	19.13
	n=30	28.73	26.36	28.66	27.05	28.57	28.24
	n=40	37.56	34.91	36.53	35.0	37.01	36.56
	n=50	44.71	42.69	45.3	43.82	45.86	45.04
	n=60	52.21	50.28	53.75	52.04	53.87	53.87
	n=70	60.86	58.92	58.87	58.71	61.41	61.34
	n=80	67.96	66.68	70.19	68.94	68.96	68.96
	n=90	74.54	72.98	75.97	75.9	77.32	76.42
m=4	n=10	10.0	9.69	10.0	9.85	9.92	9.81
	n=20	19.67	18.14	19.72	18.46	19.46	19.35
	n=30	29.16	26.35	28.92	26.97	28.83	28.25
	n=40	36.89	35.12	36.09	34.87	37.03	36.92
	n=50	45.79	42.94	45.05	43.24	45.2	44.59
	n=60	55.88	51.45	52.6	51.35	54.58	54.31
	n=70	60.84	59.72	60.69	59.15	62.01	62.01
	n=80	69.38	66.99	69.55	68.44	70.73	70.73
	n=90	78.06	76.07	76.71	76.41	78.0	77.65

Table 5: Mean solution quality of IRSA with and without replication

Solution quality for Simulated Annealing with and without replication							
# time lags:		0.1 · n		0.2 · n		0.6 · n	
replication used:		yes	no	yes	no	yes	no
m=2	n=10	10.0	10.0	10.0	10.0	9.5	9.5
	n=20	20.0	20.0	20.0	20.0	19.85	19.85
	n=30	30.0	30.0	30.0	29.98	28.41	24.79
	n=40	40.0	39.81	40.0	39.62	31.74	31.74
	n=50	50.0	49.4	50.0	49.06	12.33	9.13
	n=60	60.0	58.85	60.0	58.25	2.83	0.0
	n=70	70.0	67.79	69.98	66.81	3.49	3.15
	n=80	80.0	77.31	79.92	75.06	0.0	0.0
	n=90	90.0	86.41	80.85	74.79	0.0	0.0
m=3	n=10	10.0	10.0	10.0	10.0	10.0	10.0
	n=20	20.0	20.0	20.0	20.0	20.0	20.0
	n=30	30.0	29.95	30.0	30.0	29.96	28.34
	n=40	40.0	39.68	40.0	39.6	37.8	36.11
	n=50	50.0	48.94	50.0	48.95	29.46	22.7
	n=60	60.0	58.43	60.0	57.99	8.72	4.98
	n=70	70.0	67.35	70.0	66.81	0.0	0.0
	n=80	80.0	76.35	79.97	74.45	0.0	0.0
	n=90	90.0	84.58	89.87	83.15	0.0	0.0
m=4	n=10	10.0	10.0	10.0	10.0	10.0	10.0
	n=20	20.0	20.0	20.0	20.0	20.0	20.0
	n=30	30.0	30.0	30.0	29.98	28.5	28.38
	n=40	40.0	39.69	40.0	39.59	39.92	38.68
	n=50	50.0	49.03	50.0	49.01	44.23	44.23
	n=60	60.0	58.1	60.0	57.86	23.67	16.41
	n=70	70.0	67.33	70.0	66.89	24.46	24.46
	n=80	80.0	75.84	79.98	75.64	3.7	3.7
	n=90	90.0	84.89	89.93	84.33	0.0	0.0

Table 6: Mean solution quality of Simulated Annealing algorithm with and without replication

7 Conclusion

In this work, we introduced the problem of scheduling of F-shaped tasks with replication to maximize execution probability. We surveyed the related literature and highlighted the similarities and dissimilarities with this problem. We have formulated the formal definition of this scheduling problem. This required a generalization of existing notation and introduction of new concepts that allowed us to address this scheduling problem. We also addressed the interpretation of relative temporal constraints in schedules with replicated jobs.

Since the existing method for computation of execution probability was not applicable on schedules with replicated jobs and computing the probability from definition is not tractable even for relatively small instances, we have analyzed the problem and discovered which subset of replicated jobs can influence execution probability of other jobs. We have introduced the concept of blanket for such sets and defined a new way for evaluation of execution probability.

Furthermore, we proved the problem to be strongly \mathcal{NP} -hard and we have proposed two heuristic algorithms for its solution. These algorithms were implemented and described. We have evaluated these algorithms on a dataset of randomly generated instances for various parameters. Results of this evaluation were provided and discussed. It was shown that Simulated Annealing fails on instances with dense time lag graphs. Therefore we implemented IRSA to solve more complex instances and to provide a faster alternative to Simulated Annealing. The measured results show that the implemented IRSA algorithm can solve instances with up to 90 jobs within reasonable amount of time and demonstrated that the replication has a positive impact on the probability of execution for less critical jobs.

References

- [1] Sanjoy Baruah, Haohan Li, and Leen Stougie. Towards the design of certifiable mixed-criticality systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, pages 13–22. IEEE, 2010.
- [2] James C Bean, John R Birge, John Mittenthal, and Charles E Noon. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3):470–483, 1991.
- [3] Alan Burns and Robert Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, 2013.
- [4] Louis-Claude Canon and Emmanuel Jeannot. Evaluation and optimization of the robustness of dag schedules in heterogeneous environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):532–546, 2010.
- [5] Petr Cincibus. Algoritmy pro rozvrhování úloh s různými stupni kritičnosti a relativními časovými omezeními. 2015.
- [6] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1-3):61–95, 1991.
- [7] Christoph Dürr, Zdeněk Hanzálek, Christian Konrad, Yasmina Seddik, René Sitters, Óscar C Vásquez, and Gerhard Woeginger. The triangle scheduling problem. *Journal of Scheduling*, pages 1–8, 2017.
- [8] Na Fu, Pradeep Varakantham, and Hoong Chuin Lau. Robust partial order schedules for rcpsp/max with durational uncertainty. 2016.
- [9] Alain Girault, Hamoudi Kalla, Mihaela Sighireanu, and Yves Sorel. An algorithm for automatically obtaining distributed and fault-tolerant static schedules. In *Proceeding of International Conference on Dependable Systems and Networks*, pages 165–190. IEEE, 2003.
- [10] Zdeněk Hanzálek, Tomáš Tunys, and Přemysl Šůcha. An analysis of the non-preemptive mixed-criticality match-up scheduling problem. *Journal of Scheduling*, 19(5):601–607, 2016.
- [11] Zdeněk Hanzálek and Přemysl Sucha. Time symmetry of project scheduling with time windows and take-give resources. 11 2017.
- [12] Mathieu Jan, Lilia Zaourar, and Maurice Pitel. Maximizing the execution rate of low-criticality tasks in mixed criticality system. *Proc. WMC, RTSS*, pages 43–48, 2013.

REFERENCES

- [13] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.
- [14] Kenli Li, Xiaoyong Tang, Bharadwaj Veeravalli, and Keqin Li. Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. *IEEE Transactions on Computers*, 64(1):191–204, 2015.
- [15] Sean Luke. *Essentials of metaheuristics*, volume 113. Lulu Raleigh, 2009.
- [16] LG Mitten. Sequencing n jobs on two machines with arbitrary time lags. *Management science*, 5(3):293–298, 1959.
- [17] Yingfeng Oh and Sang H Son. Scheduling real-time tasks for dependability. *Journal of the Operational Research Society*, 48(6):629–639, 1997.
- [18] Samantha Ranaweera and Dharma P Agrawal. A task duplication based scheduling algorithm for heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, pages 445–450. IEEE, 2000.
- [19] Yasmina Seddik and Zdenek Hanzálek. Match-up scheduling of mixed-criticality jobs: Maximizing the probability of jobs execution. *European Journal of Operational Research*, 262(1):46–59, 2017.
- [20] Dvir Shabtay, Nufar Gaspar, and Moshe Kaspi. A survey on offline scheduling with rejection. *Journal of Scheduling*, 16(1):3–28, 2013.
- [21] Hang Su and Dakai Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 147–152. EDA Consortium, 2013.
- [22] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239–243. IEEE, 2007.

Appendix A CD Content

In Table 7 are listed names of all root directories on CD.

Directory name	Description
thesis	Bachelor's thesis in pdf format.
thesis_sources	latex source codes
data	generated instances and computed results in the raw format
code	Scala project containing source codes

Table 7: CD Content

Appendix B List of abbreviations

In Table 8 are listed abbreviations used in this thesis.

Abbreviation	Meaning
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
IRSA	Iterative Resource Scheduling Algorithm
MOEA	Multi-Objective Evolutionary Algorithm
MILP	Mixed-Integer Linear Program

Table 8: Lists of abbreviations

