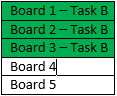
**Problem description**

**Environment**

Creating hardware processor requires a lot of computational resources and time. Additionally when several projects like this are running in parallel. A system is built, in order to share between compute resources, for different task in different projects. Specifically we are interested in scheduling tasks for costly resources called emulators.

Each emulator has five boards. A task sent by user can request one or more boards, the boards must be occupied in sequential order. For example if our current state is that we have a task that runs on emulator on board 3,

|  |
| --- |
| Board 1 |
| Board 2 |
| Board 3 – Task A |
| Board 4 |
| Board 5 |

and a new task comes in, that need 3 boards, it cannot run in the same emulator unit as the previous task, because of the constraint that all boards must be occupied sequentially. So we will use two emulators.

|  |
| --- |
| Board 1 |
| Board 2 |
| Board 3 – Task A |
| Board 4 |
| Board 5 |

A task can require more than five boards, in this case the board of the next closest emulator will be occupied. Because of this property we can model the problem as a long ordered line of boards, ignoring the existence of emulators.

A user of the system can make three different request types:

1. Provide possible time window for the task and its size.
2. Provide exact start time of the task he wants to run.
3. Request extension to already running task.

After a user make a request, the system tries to schedule the new request, and reply the user whether it was able to do so.

When doing the scheduling there are three types of tasks:

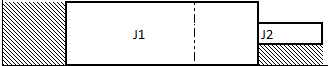
1. Task at progress, this tasks are already running, they occupy specific boards, due time is known. When doing re-scheduling this tasks will remain fixed.
2. Tasks that are promised to run in time window. For this tasks we know the earliest time they can start running, and the latest due date. When doing re-schedule if there is a need to, we can change their start time or boards being used, as long as we meet promised due date.
3. Tasks that are coming from new user request. For this tasks we know their availability, due date, and number of needed boards. There is a need to schedule in the given due date.

**Goal**

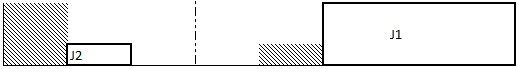
To create correct scheduling we followed guidelines as given by the business need:

1. Complete as most tasks as possible in their due date is the most important
2. For all jobs that cannot complete before or at their due date, it is preferable that:
   1. That total jobs lateness will be as small as possible.
   2. Lateness will be evenly distributed between them.
3. For all jobs that can meet their due date it is preferable that:
   1. Their earliness will be evenly distributed among them.

When scheduling two jobs, we have three possible schedules: both of the jobs will meet their due date; only one of the jobs will meet due date; none of the jobs will meet due date. Since first guideline is to prefer as most task as possible to be completed before their due date, we will always prefer first case over second, and the second over the third. For example, if we have two jobs which we can schedule in two alternative ways, both jobs with small latency, or we can make a schedule where one of the jobs will finish on time, while the other one will have a bigger latency. We will prefer alternative where we have at least one completed task.



Alternative 1, not jobs completes on due date. The dashed line is due date of both of the jobs.



Alternative 2, one job completes on due date. The dashed line is due date of both of the jobs.

If both of the jobs are not going to meet their schedule, our alternatives look like: (1) both jobs end at the same time, and their total latency is t­s; (2) jobs complete on different times and their total latency is smaller than ts; (3) jobs complete on different times and their total latency is bigger than ts;(4) jobs complete on different times and their total latency is equal ts; According to second guideline we would prefer alternative (2) over all others, then alternative (1), afterwards alternative (4) and at the end alternative (3).

**Mathematical modeling**

Let M = {M1, M2, …, Mm} be a set of distinct processors (processors and boards are used interchangeably) that can execute only one job at a time. Processors reside along a straight line were processor M2 is located immediately after M1, processor M3 immediately after M2 and so forth. Let J = {J1, J2, …, Jn} be a set of jobs that are needed to be scheduled on the processors. Each job is defined by: pj – processing time of job j; rj – ready time of job j; dj – due date of job j; sj – size of job j, number of needed sequential processors.

We present three different formulations where each formulation has the following parameters:

And below decision variables, are common to all:

**MIP Formulation 1**

This is a relative formulation taken from [Guan, 2004], in our case all weights are equal 1. Model parameters are the same as in formulation 1.

Decision Variables:

(1)

Subject to:

(2)

(3)

(4)

(5)

(6)

(7)

(8)

(9)

(10)

(11)

(13)

(14)

(15)

**MIP Formulation 2**

This formulation is also relative formulation, taken from [Heragu].

Decision variables:

(1)

Subject to:

(16)

(17)

(18)

(19)

(20)

(21)

(22)

(23)

(13)

(14)

**MIP Formulation 3**

Decision Variables:

(1)

Subject to:

(24)

(25)

(26)

(27)

(28)

(13)

(14)

Constraint (2) forces that all the jobs will start process at some period of time. (3) Enforces that at time *t*, and on board m only one job *j* can run. (4) Ensures that all jobs will not start running before their availability time. (5) Assign the completion time of job *j* to variable. (6) Calculates unit penalty, if job is late will equal 1 otherwise 0.

**Experiments**

In order to understand in which cases it is better to use different formulation, we created data sets to test the formulations on them. Our test includes following independent variables:

1. Number of jobs {5,10,15}

2. Due date {[pj-17],[pj-25]}

3. Number of machines {10,15}

Job process time and job size are randomly selected, from fixed interval. The interval for both is [1-10]. For each permutation 10 instances created.