Computers & Operations Research 142 (2022) 105730

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

# The Sample Analysis Machine Scheduling Problem: Definition and comparison of exact solving approaches

Miquel Bofill [a], Jordi Coll [b], Gerard Martín [a], Josep Suy [a,*], Mateu Villaret [a]

[a] University of Girona, Spain
[b] Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

ARTICLE INFO

*Keywords:*
Scheduling
RCPSP
Constraint Programming
SMT
MILP
Sample analysis

ABSTRACT

Nowadays there exists a wide variety of automatic machines that perform analytical tests on liquid samples, such as water, blood, urines, saliva, etc. A test can be modelled as a set of activities with given precedences and through sharing a set of limited resources. A scheduling process is therefore required to find a feasible or optimal execution of a set of tests. An important particularity of the machines performing tests are their storage areas of limited capacity. For instance, one activity may require a sample to be moved to an observation area, while another activity may later remove this sample from that observation area.

In this paper, we introduce a real problem encountered in the analytical instrument industry known as the Sample Analysis Machine Scheduling Problem (SAMSP). We show that the SAMSP is a particular case of the Resource Constrained Project Scheduling Problem with Cumulative Resources (RCPSP-Cu), and present a successful application of optimization techniques for it. We are interested in exact approaches, since the models presented will be used to prove the maximum throughput of the selected machine layouts. In particular, we compare the performance of approaches based on Constraint Programming (CP), Satisfiability Modulo Theories (SMT), and Mixed Integer Linear Programming (MILP), on real instances of SAMSP.

## 1. Introduction

When designing analytical test machines many aspects must be taken into account: the duration of the processes (activities), the quantity of resources available, the capacity of the storage areas, etc. For this reason, having a software prototype to simulate the throughput of several possible configurations of a machine using several test scenarios is extremely useful. The result of the simulations should be made available early on to machine designers to enable them to evaluate the tradeoff between the cost of technological improvement efforts (for example, decreasing the sample acquisition time by making a faster robotic arm) and the quantity of resources available in the machine (for example, allocating more space in some areas). Relevant information will only be obtained following an enormous number of simulations and the subsequent data analysis of the schedules acquired. For instance, some experiments may show that improving the speed of obtaining a sample is pointless unless the amount of room available in the heating area is greater than the threshold. Therefore, simulations need to be efficient and, where possible, provide a theoretical optimal.

We were commissioned by a company to develop a software program to help them design a new analytical test machine (a high level model of the machine is depicted in Fig. 2). The machine had to be able to perform several distinct types of tests simultaneously. Furthermore, all the tests consisted of several activities that had to be performed using limited resources. In particular, there were to be two robotic arms and a cuvette shuttle. A number of cuvettes were to contain the samples to be analysed; all of which would occupy a position at any time in a specific (limited) storage area. Because certain activities can only be performed in particular areas, this position had to be able to change during a specific test. For example, while dissolutions of samples can be performed in a cold storage area, a hot area not only allows for dissolutions, but incubations as well. Finally, there was to be an observation area (which is also a storage area with limited capacity) to evaluate the results of the tests on the samples. The activities making up the tests had to be scheduled without exceeding the capacity of any resource, while minimizing the global duration (makespan).

As said, when designing such machines many alternatives contemplating the duration of activities and resource capacities must be considered. Modelling the entire problem of obtaining an optimal design for the machine is not a realistic goal since, apart from complexity issues, there are many external limitations on the machine that cannot

* Corresponding author.
*E-mail addresses:* mbofill@imae.udg.edu (M. Bofill), jordi.coll@lis-lab.fr (J. Coll), gmartin@imae.udg.edu (G. Martín), suy@imae.udg.edu (J. Suy), villaret@imae.udg.edu (M. Villaret).

https://doi.org/10.1016/j.cor.2022.105730
Received 18 February 2021; Received in revised form 20 January 2022; Accepted 24 January 2022
Available online 8 February 2022

be easily known beforehand. However, having a software prototype to simulate the throughput of several possible configurations of the prototype machine is extremely useful and will hopefully guide the engineers to obtaining an accurate final design. While there are a number of studies in the literature that deal with particular sample analysis machines (Shin et al., 2010; You et al., 2017), we were unable to find any that could match the requirements of the company that approached us.

Our problem is related to the well-known Resource Constrained Project Scheduling Problem (RCPSP), which consists of scheduling a set of non-preemptive activities (or tasks) with predefined durations and demands on each of a set of renewable resources, subject to partial precedence constraints. Durations and precedence constraints between tasks imply minimum distances between activities (minimum time lags). Normally, the goal is to minimize the makespan, i.e., the schedule's end time. Many generalizations and specializations exist; for a survey on variants and extensions of the RCPSP see Brucker et al. (1999), Hartmann and Briskorn (2010). In particular, in the RCPSP/max (Bartusch et al., 1988), minimum and maximum time lags are considered. That is, a maximum delay between the start time of every two tasks can be specified, in addition to the minimum delays implied by precedences. An extension of the RCPSP/max is the RCPSP with cumulative resources (RCPSP-Cu), where storage facilities are added. The idea is that activities may require some intermediate product which is withdrawn from a storage facility, or they may manufacture products which are then put into a storage facility. A cumulative resource is given by the capacity and the minimum inventory level (or safety stock) of the storage area for this particular resource. See Neumann et al. (2005), Carlier et al. (2009), Hartmann and Briskorn (2010) and Chaleshtarti and Shadrokh (2011) for details. Note that the RCPSP-cu problem should not be confused with the Cumulative Scheduling Problem (CuSP), which is a special case of the RCPSP with one resource. It should also be noted that the term *cumulative resource* is used in some of the literature as a synonym for renewable resources, see Artigues et al. (2013).

With the problem at hand, we will have no manufactured products stored in some stock-keeping facility, but rather storage areas that can be occupied or vacated. An activity, in addition to having precedence restrictions and resource demands, can occupy or vacate a number $n \geq 0$ of positions of a certain storage area. Each storage area will have a limited number of positions (here we talk about cumulative demands resource or storage constraints). If an activity occupies positions, it begins to do so as soon as it starts and, on the contrary, if it vacates some position, it will do so at the end. This is just the opposite to the idea in the RCPSP-Cu. However, although the semantics are different, the way of solving the problem will essentially be the same.

We introduce the *Sample Analysis Machine Scheduling Problem (SAMSP)* as a special case of the RCPSP-Cu, and describe the methods applied and the experiments performed to solve this industrial problem in a real environment. Since in the design phase of the machine optimality of solutions is required, we are interested in exact solving methods. All known exact solvers for the RCPSP-Cu use branch-and-bound or MILP approaches (Neumann et al., 2005; Carlier et al., 2009; Chaleshtarti and Shadrokh, 2011). However, more recent research has shown that other approaches, namely Constraint Programming (CP) (Schutt et al., 2013; Vilím et al., 2015; Laborie, 2018; Lunardi et al., 2020) and Satisfiability Modulo Theories (SMT) (Bofill et al., 2016, 2017, 2020), are state-of-the-art for the particular case of the RCPSP/max and other similar scheduling problems.

In this paper we consider a model-and-solve approach to the SAMSP with CP, SAT/SMT and MILP. These technologies are well known for their efficiency as well as for their flexibility regarding problem definition modifications and/or additions of constraints. The purpose of this work is to present exact techniques to certify optimal solutions for the SAMSP in order to define the best configuration of the machine being designed. As shown in the experimental section, with this approach we were able to solve most of the instances proposed very efficiently and were able to certify optimality in most of the cases, thus allowing the best configuration to be determined. This knowledge is crucial for good decision making.

We consider the following technologies:

- Constraint Programming, in particular using the CP Optimizer (IBM, 2019; Laborie et al., 2018), the proprietary IBM constraint solver targeted at industrial scheduling problems, with excellent results on various RCPSP-like problems. The CP Optimizer consists of an exact algorithm which transparently embeds lots of metaheuristic searches. Its hybrid method exploits the flexibility of CP to integrate mathematical concepts, such as intervals, functions, sequences, etc., into the model, while using the good ideas of MILP solvers: model and run, conflict refinement, warm-starts, etc.
- SMT, which is an excellent candidate for model-and-solve approaches to RCPSP-like problems thanks to its good balance between efficiency and expressiveness in combining logic statements with arithmetic. The encodings that we present are inspired by the state-of-the-art encodings on RCPSP and some of their variants presented in Bofill et al. (2020), and we use the SMT solver Yices (Dutertre and de Moura, 2006) which has proven to be very competitive in solving scheduling problems.
- We also provide a pure MILP model using the techniques that have shown the best performance in scheduling problems, such as time index and disaggregated time formulations (Pritsker et al., 1969; Christofides et al., 1987; Artigues, 2017). We use the IBM MILP solver CPLEX (IBM, 2019).

The experiments show that the CP technology is clearly the best on the SAMSP, followed at a certain distance behind by SMT, and with MILP lagging far behind in performance.

The rest of the paper is organized as follows. In Section 2 we (re)define the RCPSP-Cu. In Section 3 we describe our particular real world industrial case, the SAMSP, and show how the SAMSP can be reduced to the RCPSP-Cu. In Section 4 we define the preprocessing and symmetry breaking used in all the formulations. In Section 5 we provide a CP optimizer formulation of the SAMSP, along with a variant of it. In Section 6 we provide an SMT encoding of the SAMSP, together with some refinements. In Section 7 we provide an MILP formulation, and also a variant of it. Section 8 is devoted to experiments, where we compare the performance of the different formulations and refinements on real SAMSP instances. We conclude in Section 9 by pointing out some future work.

## 2. The resource-constrained project scheduling problem with cumulative resources (RCPSP-Cu)

In the classic case of the RCPSP-Cu, production is considered to be carried out at the completion of the corresponding manufacturing activity, while consumption is always performed at the start of the activity. In our case, an activity occupying positions in the storage facility will begin to occupy them as soon as it starts and, on the contrary, if it vacates a position, it will do so when the activity has been completed (see Fig. 1).

This difference can be managed in the classical RCPSP-Cu by modelling occupancy as consumption of a *free space* resource and vacation as production of free space. However, since this is a bit unnatural, we redefine the RCPSP-Cu as follows.

The *Resource-Constrained Project Scheduling Problem with Cumulative Resources (RCPSP-Cu)* is a tuple $T = (\mathscr{V}, d, \mathscr{E}, \mathscr{R}, B, b, \mathscr{C}, P, c)$ where:

- $\mathscr{V} = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$ is a set of non-preemptive activities. Activities $A_0$ and $A_{n+1}$ are dummy activities representing, by convention, the start and the end of the schedule, respectively. The set of non-dummy activities is defined by $\mathscr{A} = \{A_1, \ldots, A_n\}$ with $n > 0$.

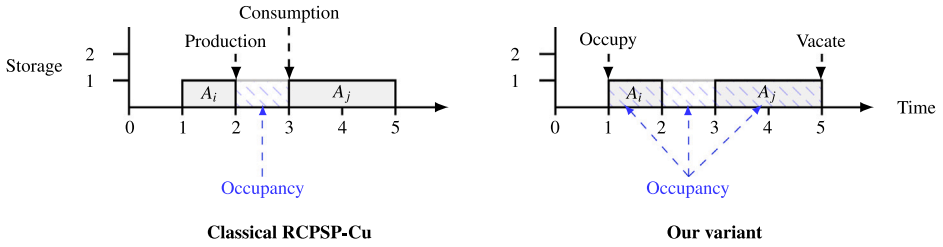


**Fig. 1.** Difference between the classical RCPSP-Cu and our variant on storage occupancy between two activities.

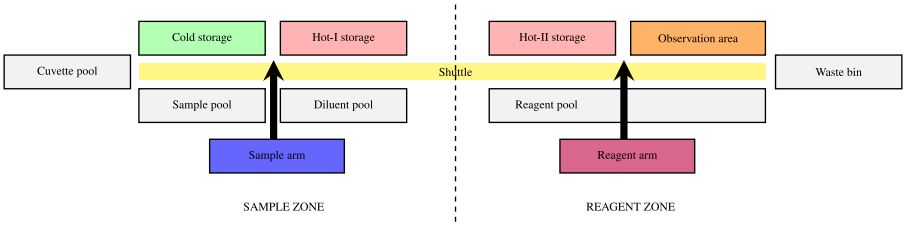


**Fig. 2.** Schematic machine architecture.

- $d$ is a vector of $n+2$ naturals, $d_i$ being the duration of activity $i$, where $d_0 = d_{n+1} = 0$, and $d_i \geq 0 \; \forall A_i \in \mathscr{A}$.
- $\mathscr{E}$ is a set of triplets of the form $(A_i, A_j, l_{i,j})$ which represent precedence relations between activities with a (positive or negative) time lag, meaning that the difference between the start of $A_j$ and $A_i$ is at least $l_{i,j}$.[1]
  We assume that we are given a precedence activity-on-node (AON) graph $G(\mathscr{V}, \mathscr{E})$ without cycles of a positive sum of time lags, since otherwise the precedence relation would be inconsistent. We also assume that $\mathscr{E}$ is such that $A_0$ is a predecessor of all other activities and $A_{n+1}$ is a successor of all other activities.
- $\mathscr{R} = \{R_1, \ldots, R_r\}$ is a set of renewable resources.
- $B$ is a vector of $r$ naturals, $B_i$ being the available amount of each resource $R_i$.
- $b$ is a matrix of $(n+2)\times r$ naturals corresponding to the renewable resource demands of activities. That is, $b_{i,j}$ represents the amount of resource $R_j$ used during each unit of time of the execution of activity $A_i$. Note that $b_{0,j} = 0$ and $b_{n+1,j} = 0 \;\; \forall j \in \{1, \ldots, r\}$.
- $\mathscr{C} = \{C_1, \ldots, C_{cr}\}$ is a set of cumulative resources (storage areas).
- $P$ is a vector of $cr$ naturals, $P_i$ being the available amount of cumulative resource $C_i$.
- $c$ is a matrix of $(n+2) \times cr$ integers corresponding to the number of positions occupied or vacated by each activity; namely, $c_{i,j} > 0$ represents that activity $A_i$ occupies $c_{i,j}$ positions in storage area $C_j$ and $c_{i,j} < 0$ represents that activity $A_i$ vacates $\left|c_{i,j}\right|$ positions in storage area $C_j$. Any activity occupying positions will begin to occupy them as soon as it starts and, on the contrary, if it vacates some positions, it will do so just after the activity has been completed. Note that $c_{0,j} = 0$ and $c_{n+1,j} = 0 \;\; \forall j \in \{1, \ldots, cr\}$.

The goal of the RCPSP-Cu is to find a schedule (start time) for all activities, such that the global duration (makespan) is minimized and the schedule is feasible with respect to the precedences, and the renewable and cumulative resource constraints. This means that, for each time instant:

(i) for each renewable resource, the amount of that resource required by all the activities running at that time is not greater than the given resource's availability, and

(ii) for each storage area, its occupancy is between zero and the given capacity.

## 3. The Sample Analysis Machine Scheduling Problem (SAMSP)

The industrial problem that we consider consists of scheduling a set of tests in a sample analysis machine, minimizing the makespan. The problem starts with a set of samples; each one located in a different recipient of a sample pool. Multiple tests can be asked to be performed on each sample. There are several distinct test types, each one consisting of a particular collection of activities with their corresponding precedences, durations, and renewable and cumulative resource demands. The possible steps of a test are as follows:

1. A cuvette is acquired from a cuvette pool and moved to a storage facility.
2. A sample from the sample pool is aspirated and deposited into that cuvette.
3. Once the sample is in the cuvette, several diluents from a diluents pool can be added.
4. The previous actions can be performed both in cold or hot storage areas. If the cuvette is already in a hot storage area then incubation starts, otherwise it will be moved to the hot storage area to start incubation.
5. During incubation, several reagents from a reagent pool can be added to the sample.
6. The recipient is moved to an observation area, where other reagents can also be added, and the observation process starts. This observation process produces the test outcome.
7. Finally, the cuvette is thrown into the waste bin.

A *Sample Analysis Machine* specification is composed of a set of resources with their respective available amounts and a set of storage areas with their respective capacities. Several test types can be specified according to the characteristics of the machine where they are to be performed. A *test type* specification for a particular sample analysis machine is composed of a set of activities (e.g., sample aspiration, cuvette transport between storage areas, addition of diluents or reagents, etc.) that the machine must perform to accomplish that specific test type. There is a generalized precedence relation between the activities of a test type: minimum and maximum time lags. Each activity also has a duration, a set of renewable resource demands (a robotic arm to acquire samples, a shuttle to move cuvettes, etc.) and a number of occupied or vacated positions in storage areas (cold, hot, etc.).

[1] Positive $l_{i,j}$ values are used to enforce a minimum delay of the start of $A_j$ w.r.t. $A_i$, and negative values are used to set a maximum delay of $|l_{i,j}|$ of the start of $A_i$ w.r.t. $A_j$.

Then, the *Sample Analysis Machine Scheduling Problem (SAMSP)* consists of, a given sample analysis machine specification, a set of test type specifications for that machine, the number of tests of each test type to be performed, and a schedule generated of the activities that minimizes the completion time and fulfils all precedences, renewable resource and storage constraints. This problem is similar to those considered in Shin et al. (2010) and You et al. (2017) albeit with different machine architecture.

### 3.1. The Sample Analysis Machine architecture

As an example, the specific sample analysis machine that we consider is structured into two zones: the sample zone and the reagent zone. Each zone is only accessible by its own robotic arm: the sample arm and reagent arm, respectively. A shuttle allows cuvettes to be transported between the different storage areas allocated in the machine, thus connecting the two zones (see Fig. 2). The sample zone is composed of two cuvette storage areas (Cold and Hot-I), and three pools (Cuvette, Sample and Diluent pools). The reagent zone is also composed of two cuvette storage areas (Hot-II and Observation), plus a Reagent pool and a Waste bin.

The possible operations that can be performed in this machine are the following:

- Shuttle movements:
  - Move a cuvette from the cuvette pool to any storage area.
  - Move a cuvette between storage areas.
  - Move a cuvette from a storage area to the waste bin.
- Sample arm operations:
  - Aspirate a sample and dispense it to a cuvette located in the Cold or Hot-I storage areas.
  - Aspirate from a cuvette located in the Cold or Hot-I storage areas and dispense the contents to another cuvette of the same zone.
- Reagent arm operations:
  - Aspirate a reagent and dispense it to a cuvette located in the Hot-II or Observation areas.

The activities making up the tests will correspond to the operations described above, with their appropriate durations, precedences, renewable resource (shuttle and arms) demands and cumulative resource (Cold, Hot-I, Hot-II storages and Observation area) demands and supplies.

#### 3.1.1. A test type example for the machine architecture considered

A test type example for the machine architecture described above is depicted in Fig. 3. The meaning of each activity is as follows:

1. A cuvette (cuvette 1) is transported from the cuvette pool to the cold storage area by the shuttle.
2. A cuvette (cuvette 2) is transported from the cuvette pool to the cold storage area by the shuttle.
3. A sample is aspirated and deposited in cuvette 1 by the sample arm.
4. A diluent is aspirated and deposited in cuvette 1 by the sample arm.
5. A diluent is aspirated and deposited in cuvette 2 by the sample arm.
6. A diluent is aspirated and deposited in cuvette 2 by the sample arm.
7. The mixture in cuvette 2 is aspirated and deposited in cuvette 1 by the sample arm.
8. Cuvette 2 is transported from the cold storage area to the waste bin by the shuttle.
9. Cuvette 1 is transported from the cold storage area to the hot-II storage area by the shuttle.
10. A reagent is aspirated and deposited in cuvette 1 by the reagent arm.
11. Cuvette 1 is transported from the hot-II storage area to the observation area by the shuttle.
12. Cuvette 1 is transported from the observation area to the waste bin by the shuttle.

### 3.2. The SAMSP as an RCPSP-Cu instance

In this subsection we provide an example of a particular SAMSP with its machine configuration and one test type, and show how this becomes an RCPSP-Cu instance (cf. Section 2).

We assume we have a machine with the same architecture described in Section 3.1 with three renewable resources $\mathscr{R} = \{\text{Shuttle}, \text{Sample Arm}, \text{Reagent Arm}\}$. All three resources have only one unit available; therefore $B = \langle 1, 1, 1\rangle$. The machine has four cumulative resources (storage areas) $\mathscr{C} = \{\text{Cold}, \text{Hot-I}, \text{Hot-II}, \text{Observation}\}$. The capacities of the storage areas are: four positions for cold, hot-I and hot-II storage and two positions for observation area, i.e., $P = \langle 4, 4, 4, 2\rangle$.

To turn an SAMSP instance into an RCPSP-Cu instance we need to obtain the global set of precedence relations $\mathscr{E}_g$ by merging all precedence relations of the different tests to be performed. However, we will consider all initial activities as the same global initial activity $A_0$ and all the final activities as the same global final activity $A_{n+1}$, with $n$ being the number of non-dummy activities of all joined tests. We denote by $\mathscr{V}_g$ the resulting set with all activities. These activities have their corresponding durations $\mathfrak{d}$, resource demands $\mathfrak{b}$ and occupancy/vacation amounts $\mathfrak{c}$. The set of non-dummy activities is then defined as $\mathscr{A}_g = \mathscr{V}_g \setminus \{A_0, A_{n+1}\}$. Finally, we denote by $T_g = (\mathscr{V}_g, \mathfrak{d}, \mathscr{E}_g, \mathscr{R}, B, \mathfrak{b}, \mathscr{C}, P, \mathfrak{c})$ the RCPSP-Cu instance resulting from the union of all the tests.

Fig. 4 shows the solution found for an RCPSP-Cu consisting of two test instances of the test type depicted in Fig. 3.

Note that the SAMSP cannot be directly encoded as a traditional RCPSP. One could think of simply joining a storage occupancy task and its corresponding storage vacancy task into a single task, and consider storage as a renewable resource. Unfortunately this is not possible in traditional RCPSP because tasks have a fixed duration. Moreover, the occupancy and the vacancy tasks use a different renewable resource (shuttle) only during their execution, which again makes it impossible to merge them into a single activity.

## 4. Preprocessing and symmetry breaking

In this section we describe some preprocessing steps and the symmetry breaking considerations that will be used in the different approaches to the problem.

### 4.1. Preprocessing

We compute the transitive closure $\mathscr{E}_g^*$ of $\mathscr{E}_g$, and as a result $\mathscr{E}_g^*$ contains a triplet $(A_i, A_j, l_{i,j})$ for any pair of activities $(A_i, A_j)$ connected by a path, where $l_{i,j}$ is the maximum length of any path from $A_i$ to $A_j$ in $G(\mathscr{V}_g, \mathscr{E}_g)$.

We obtain the trivial upper bound

$$UB = \sum_{A_i \in \mathscr{A}_g} \max(\mathfrak{d}_i, \max_{(A_i, A_j, l_{i,j}) \in \mathscr{E}_g} l_{i,j})$$

and the trivial lower bound $LB = l_{0,n+1}$, where $(A_0, A_{n+1}, l_{0,n+1}) \in \mathscr{E}_g^*$.

Also, we can compute the time windows for the different activities, represented by their earliest ($ES_i$) and latest ($LS_i$) start time, and earliest ($EC_i$) and latest ($LC_i$) close time. They are computed, given an upper bound $UB$, based on the time lag of each activity w.r.t. activities $A_0$ and $A_{n+1}$: $ES_i = l_{0,i}$, $LS_i = UB - l_{i,n+1}$, $EC_i = ES_i + \mathfrak{d}_i$, $LC_i = LS_i + \mathfrak{d}_i$.

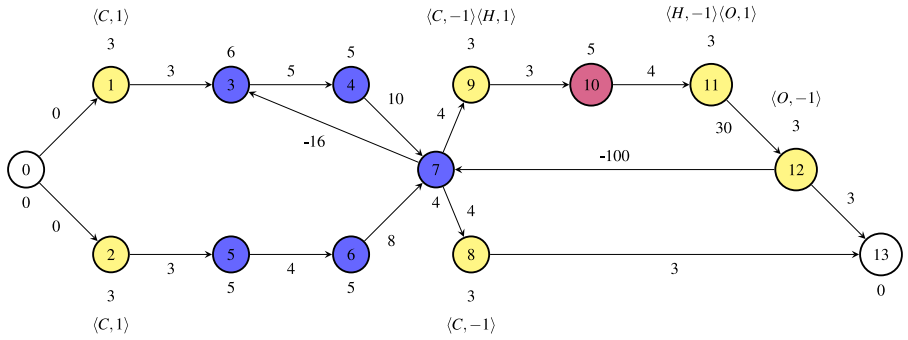


**Fig. 3.** Graph of precedences of a test type. Each activity is labelled by its duration and the number of positions occupied/vacated in the corresponding storage area (C:Cold; H:Hot-II; O:Observation) if the activity includes this. Shuttle actions are depicted in yellow, sample arm actions in blue and reagent arm actions in purple. The label of the edges are the requested time lags between activities.

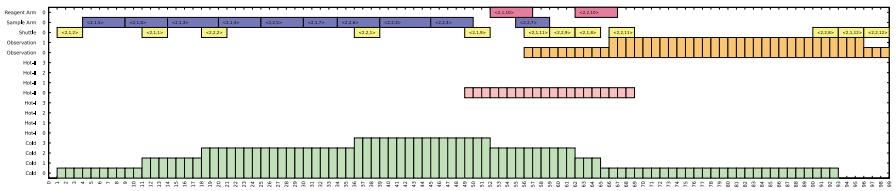

**Fig. 4.** Solution for two instances of the test type depicted in Fig. 3. In each activity the triplet $\langle tt, itt, na\rangle$ represents: $tt$ the test type, $itt$ the instance number of the test and $na$ the activity number in the test type.

### 4.2. Symmetry breaking

If we have several tests of the same type, the relative order between them is indifferent, so we can fix some relative order without losing optimality. To enforce this relative order, we incorporate two precedence relations (1) and (3) in $\mathscr{E}_g$ (see below) before computing the extended precedence graph $\mathscr{E}_g^*$. Recall that inequality (14) transforms each precedence relation from $\mathscr{E}_g^*$ into a precedence constraint. These additional symmetry breaking precedence constraints allow windows to be significantly reduced.

To define the new precedence relations we introduce a function $\sigma(A_i) = \langle tt_i, itt_i, na_i\rangle$ which, for each activity $A_i \in \mathscr{A}_g$, returns its test type $tt_i$, its instance number $itt_i$ within its test type, and the activity number $na_i$ in its test type. The added precedences are the following:

- *Resource*. We can enforce a relative order between all identical activities of test instances of the same test type.

$$(A_i, A_j, o) \qquad \begin{array}{c}\forall A_i, A_j \in \mathscr{A}_g, \text{ s.t.}\\ tt_i = tt_j, na_i = na_j, itt_i = itt_j - 1\end{array} \tag{1}$$

  where $\sigma(A_i) = \langle tt_i, itt_i, na_i\rangle$, $\sigma(A_j) = \langle tt_j, itt_j, na_j\rangle$ and $o = 0$ if $A_i$ (equivalently $A_j$) does not use any resources, or $o = \mathfrak{d}_i$ if the activity uses some resource (note that in our setting all resources are unary).
- *Storage*. These precedences only apply to activities occupying storage areas. Since each storage area has a certain capacity, we can bound the number of activities with the same number and the same test type that can be running in parallel. This amount will depend on the storage capacity and the units occupied by each activity. The bound can be enforced until the storage area is vacated by, at least, one unit. We need to compute the minimum amount of time that a certain storage area will be definitely be occupied by such activities (2). These time lags will be computed considering only test types of equal type $tt$. Using (3) we compute a set of new precedences that will replace the existing ones in $\mathscr{E}_g^*$ when they

$$pt_{i,s,k} = \min\Big(\infty, \min_{\substack{A_j \in \mathscr{A}_k, \text{ s.t.}\\ \mathfrak{c}_{i,s} \neq 0,\\ (A_i, A_j, l_{i,j}) \in \mathscr{E}_k^*, l_{i,j} > 0}} l_{i,j} + \mathfrak{d}_j\Big)$$

$$\forall A_i \in \mathscr{A}_k, \forall C_s \in \mathscr{C}, \forall T_m \tag{2}$$

$$(A_i, A_j, pt_{na_i,s,k}) \qquad \begin{array}{c}\forall A_i, A_j \in \mathscr{A}_g, \forall C_s \in \mathscr{C}, \forall T_m, \text{ s.t.}\\ pt_{na_i,s,k} > 0, pt_{na_i,s,k} < \infty,\\ tt_i = tt_j = k, na_i = na_j, itt_i = itt_j - \lfloor \frac{P_s}{\mathfrak{c}_{i,s}} \rfloor\end{array} \tag{3}$$

where $\sigma(A_i) = \langle tt_i, itt_i, na_i\rangle$, $\sigma(A_j) = \langle tt_j, itt_j, na_j\rangle$. In (3), activities $A_i$ and $A_j$ have the same test type and activity number, and the distance between instances of that test type due to the capacity of storage area $C_s$ and the storage area demand of $A_i$ is $\lfloor \frac{P_s}{\mathfrak{c}_{i,s}} \rfloor$. For example, if the size of a storage area $C_s$ is $P_s = 2$ and the demand for positions of an activity $A_i$ is $\mathfrak{c}_{i,s} = 1$, we will add precedence relations between all activities with the same activity number as $A_i$ in instance tests 1 and 3, 2 and 4, …

## 5. CP formulation for the SAMSP

We use the CP Optimizer system, with a set of variables *tasks* declared as *IntervalVarArray tasks(nbTasks)* which is a vector of variables of type interval. These type of variables group all the information (duration, resource consumption, storage consumption, start, end, etc.) of an activity. In our case, all the information specified in the instance, such as durations and resource consumption, are preassigned. The number of activities is *nbTasks* and is equal to the cardinality of $\mathscr{V}_g$.

The preprocessing and the symmetry breaking exposed in Section 4, have also been used in CP encoding with a notable reduction in resolution time.

Since the CP models are not time indexed, in contrast with the SMT and MILP models, we do not need to provide an upper bound for the makespan. Also, we have observed that constraining the start times of the activities to be within their time windows does not result in any improvement in performance.

### 5.1. CP formulation

$$Objective: \quad Minimize(EndOf(tasks[nbTasks - 1])) \tag{4}$$

$$StartBeforeStart(task[i], tasks[j], l_{i,j}) \quad \forall (A_i, A_j, l_{i,j}) \in \mathscr{E}_g^* \tag{5}$$

For the renewable resource constraints, we can declare *CumulFunctionExpr resources(nbResources)* as an array of size $nbResources$ of cumulative functions representing the sum of individual contributions of interval variables.

$$resources[r] \leq B_r \qquad \forall R_r \in \mathscr{R} \tag{6}$$

where

$$resources[r] = \sum_{A_i \in \mathscr{A}_g, \mathfrak{b}_{i,r}>0} Pulse(task[i], \mathfrak{b}_{i,r})$$

For the storage constraints, we can declare *CumulFunctionExpr storages(nbStorages)* which is, again, an array of size *nbStorages* of cumulative functions.

$$storages[s] \leq C_s \qquad \forall C_s \in \mathscr{C} \tag{7}$$

where

$$storages[s] = \sum_{\substack{A_i \in \mathscr{A}_g, \\ \mathfrak{c}_{i,s} > 0}} StepAtStart(task[i], \mathfrak{c}_{i,s}) - \sum_{\substack{A_i \in \mathscr{A}_g, \\ \mathfrak{c}_{i,s} < 0}} StepAtEnd(task[i], -\mathfrak{c}_{i,s})$$

The objective described in (4) is to minimize the makespan. Constraints (5) are the precedence constraints. Constraints (6) are the renewable resource constraints and Constraints (7) are the cumulative constraints. In (6) we use *Pulse*, which is an elementary cumulative function expression representing the contribution of an individual interval variable (activity) when the activity is running. In (7) we use *StepAtStart* and *StepAtEnd* which are elementary cumulative function expressions representing the contribution of an individual interval variable (activity) when the activity is started or ended respectively, up to our horizon.

### 5.2. Refinements

We have considered two refinements. The first consists of using the $NoOverlap$ function to model the renewable resources given that they are all one unit in size. The second consists of using the $Pulse$ function to model cumulative resources, since whenever we place a cuvette in a warehouse it will, in accordance with our instructions, end up leaving the warehouse.

#### 5.2.1. Using NoOverlap for renewable resources

As stated earlier, in the machine architecture considered all renewable resources are unary ($B_r = 1$) and demands on them are also unary ($\mathfrak{b}_{i,r} = 1$). Therefore, we can replace renewable resource constraints (6) by employing a disjunction between the execution of activities requiring the same resource.

The *resources* array is not declared, but instead we declare an *IntervalVarArray over* for each resource $r$, which is an array of interval variables. We will place in it the activities that cannot be overlapped for each resource.

$$NoOverlap(over_r) \quad \forall R_r \in \mathscr{R} \tag{8}$$

where

$$over_r = [\ tasks[i] \quad | \quad A_i \in \mathscr{A}_g,\ \text{s.t. } \mathfrak{b}_{i,r} > 0]$$

#### 5.2.2. Using pulse for cumulative resources

Every test has a description which says when a cuvette is to be placed in a particular location and when it is to be removed from said location, i.e., one task puts a cuvette in a specific storage area and another removes this cuvette from that the storage point.

At preprocessing time we compute the set

$$\mathscr{O}_g = \bigcup_{\forall C_s \in \mathscr{C}} \mathscr{O}_s$$

where each $\mathscr{O}_s$ is the same as $\mathscr{O}_g$ restricted to a storage $C_s$. Each $o_i \in \mathscr{O}_g$ is a virtual task that starts at the same time as the task with number $o_i^s (A_{o_i^s} \in \mathscr{A}_g)$ (put a cuvette in a storage area) and ends at the same time as the task with number $o_i^e (A_{o_i^e} \in \mathscr{A}_g)$ (remove a cuvette from a storage area). The $o_i^s$ and $o_i^e$ associated with $o_i$ are also computed at the preprocessing time, depending on a description of a test.

To implement the set $\mathscr{O}_g$, we use an *IntervalVarArray* called $occupancy$. The constraints linking $o_i$ with their respective $A_{o_i^s}$ and $A_{o_i^e}$ are:

$$StartatStart(occupancy[i], tasks[o_i^s]) \quad \forall o_i \in \mathscr{O}_g \tag{9}$$

$$EndatEnd(occupancy[i], tasks[o_i^e]) \quad \forall o_i \in \mathscr{O}_g \tag{10}$$

The execution of each element of occupancy determines the occupancy of a position in a storage area, so Constrains (7) are replaced by:

$$storages[s] \leq C_s \qquad \forall C_s \in \mathscr{C} \tag{11}$$

where

$$storages[s] = \sum_{o_i \in \mathscr{O}_s} Pulse(occupancy[i], 1)$$

we describe the variables used in the encoding and then the encoding itself.

## 6. SMT encoding of the SAMSP

First, we describe the variables used in the encoding and then the encoding itself. In the encoding, we use the preprocessing and symmetry breaking described in Section 4. Furthermore, we define the optimization method used, as well as some refinements made to the initial encoding.

### 6.1. Variables

Below we define the following variables that will be used in our constraints.

- $S_i$ integer variables that denote the start time of activity $A_i$, $\forall A_i \in \mathscr{V}$.
- $X_{i,t}$ integer variables that are 1 if $A_i$ is running at time $t$, or 0 otherwise. $\forall A_i \in \mathscr{A}, \forall t \in [ES_i \dots LC_i - 1]$.
- $Y_{i,t}$ integer variables that are 1 iff $A_i$ has already started at time $t$, or 0 otherwise, $\forall A_i \in \mathscr{A}, \forall t \in [ES_i \dots LS_i - 1]$, s.t.$\exists C_s \in \mathscr{C}, \mathfrak{c}_{i,s} \neq 0$.

### 6.2. SMT encoding

The following set of SMT constraints encode a schedule within a given time horizon $H$.

$$S_0 = 0 \tag{12}$$

$$(ES_i \leq S_i) \wedge (S_i \leq LS_i) \qquad \forall A_i \in \mathscr{V} \tag{13}$$

$$S_j - S_i \geq l_{i,j} \qquad \forall (A_i, A_j, l_{i,j}) \in \mathscr{E}^* \tag{14}$$

$$(0 \leq X_{i,t}) \wedge (X_{i,t} \leq 1) \qquad \forall A_i \in \mathscr{A}, \forall t \in [ES_i \dots LC_i - 1] \tag{15}$$

$$X_{i,t} = 1 \leftrightarrow ((S_i \leq t) \wedge (t < S_i + \mathfrak{d}_i)) \qquad \forall A_i \in \mathscr{A}, \forall t \in [ES_i \dots LC_i - 1] \tag{16}$$

$$(0 \leq Y_{i,t}) \wedge (Y_{i,t} \leq 1) \qquad \forall A_i \in \mathscr{A}, \forall t \in [ES_i \dots LS_i - 1],\ \text{s.t. } \exists C_s \in \mathscr{C}, \mathfrak{c}_{i,s} \neq 0 \tag{17}$$

$$Y_{i,t} = 1 \leftrightarrow (S_i \leq t) \qquad \forall A_i \in \mathscr{A}, \forall t \in [ES_i \dots LS_i - 1],\ \text{s.t. } \exists C_s \in \mathscr{C}, \mathfrak{c}_{i,s} \neq 0 \tag{18}$$

$$\left(\sum_{\substack{A_i \in \mathscr{A},\ \text{s.t.}\\ t \in [ES_i \dots LC_i - 1],\\ \mathfrak{b}_{i,r} > 0}} \mathfrak{b}_{i,r} \cdot X_{i,t}\right) \leq B_r \quad \forall R_r \in \mathscr{R}, \forall t \in [0 \dots H] \tag{19}$$

$$0 \leq PS_{s,t} + \left(\sum_{\substack{A_i \in \mathscr{A},\ \text{s.t.}\\ t \in [ES_i \dots LS_i - 1],\\ \mathfrak{c}_{i,s} > 0}} \mathfrak{c}_{i,s} \cdot Y_{i,t}\right) + \left(\sum_{\substack{A_i \in \mathscr{A},\ \text{s.t.}\\ t \in [EC_i \dots LC_i - 1],\\ \mathfrak{c}_{i,s} < 0}} \mathfrak{c}_{i,s} \cdot Y_{i,(t-\mathfrak{d}_i)}\right) \leq P_s$$
$$\forall C_s \in \mathscr{C}, \forall t \in [0 \dots H] \tag{20}$$

where

$$PS_{s,t} = \sum_{\substack{A_j \in \mathscr{A},\ \text{s.t.}\\ LS_j \leq t,\ \mathfrak{c}_{j,s} > 0}} \mathfrak{c}_{j,s} + \sum_{\substack{A_j \in \mathscr{A},\ \text{s.t.}\\ LC_j \leq t,\ \mathfrak{c}_{j,s} < 0}} \mathfrak{c}_{j,s} \tag{21}$$

Constraints (13) restrict the start times of each activity. Constraints (14) are precedence constraints. Constraints (15) and (16) enforce pseudo-Boolean variables $X_{i,t}$ to be 1 if and only if activity $A_i$ is running at time $t$. Constraints (17) and (18) enforce pseudo-Boolean variables $Y_{i,t}$ to be 1 if and only if activity $A_i$ has already started to run at time $t$ and occupy/vacate a storage area. Constraints (19) and (20) are the renewable resource and storage constraints, respectively. It can be observed that when an activity occupies some positions in a storage area (an activity having a positive value of $\mathfrak{c}_{i,s}$) this occupancy counts for each time instant ranging from the start time of the activity to $H$. On the other hand, when an activity vacates some positions in a storage area (an activity having a negative value of $\mathfrak{c}_{i,s}$) this also counts for each time instant ranging from the end time of the activity to $H$. We can see that if the test is well constructed, then the non-negative restriction in (20) is not necessary since a storage area will always be occupied before being vacated, and there is a precedence between occupying and vacating actions. Eq. (21) describes how we pre-compute the number $PS_{s,t}$ for all cumulative resources and for each time instance. This number is the aggregate of the occupancy and vacation of storage $s$ due to the activities that have already started or finished at time $t$ according to their time windows.

### 6.3. Optimization

Since we use a time indexed formulation, the number of variables and clauses of our SMT encoding is proportional to the given upper bound of the makespan. Therefore, it is very important to find a good upper bound. To do this, we have implemented a procedure based on satisfiability queries to the SMT solver. We start with checking the feasibility of the problem by setting as upper bound the trivial lower bound. If we obtain a positive answer then we have an upper bound (and hence an optimal solution because the upper bound is the same as the lower bound). Otherwise, we keep doubling the value of this upper bound and repeating the process until we obtain a positive answer, and hence a valid upper bound. This technique is based on the destructive lower bounds technique introduced in Klein and Scholl (1999).

Once we have obtained a valid upper bound, optimization is performed using a binary search between the $UB$ and the $LB$, until the difference between these limits is less than 10 units of time. Then, we carry out a top-down sequential search avoiding re-encoding the problem from scratch at each satisfiability call but instead simply adding clauses to decrease the upper bound. In doing this, we are able to take advantage of the lemmas learnt by the SMT solver between successive calls.

### 6.4. Refinements

We made the following refinements to the encoding in Section 6.2:

1. **Resource Constraints**. In the machine architecture considered, all renewable resources are unary ($B_r = 1$) and the demands for them are also unary ($\mathfrak{b}_{i,r} = 1$). Therefore, we can replace the renewable resource constraints (19) in the basic encoding in Section 6.2 in several alternative ways:

   - We can enforce a disjunction between the execution of activities requiring the same resource:

   $$(S_j \geq S_i + \mathfrak{d}_i) \vee (S_i \geq S_j + \mathfrak{d}_j)$$
   $$\forall A_i, A_j \in \mathscr{A}_g, i < j,\ \text{s.t.}$$
   $$[ES_i \dots LC_i - 1] \cap [ES_j \dots LC_j - 1] \neq \emptyset,$$
   $$\exists R_r \in \mathscr{R}, \mathfrak{b}_{i,r} > 0, \mathfrak{b}_{j,r} > 0 \tag{22}$$

   In this case, it is not necessary to introduce variables $X_{i,t}$ nor constraints (15) and (16).
   - Since integer variables $X_{i,t}$ are in fact pseudo-Boolean variables, we can easily replace them by Boolean variables $x_{i,t}$ that are true iff $A_i$ is running at time $t$, $\forall A_i \in \mathscr{A}, \forall t \in [ES_i \dots LC_i - 1]$, i.e., replacing constraints(15) and (16) with:

   $$x_{i,t} \leftrightarrow ((S_i \leq t) \wedge (t < S_i + \mathfrak{d}_i)) \quad \forall A_i \in \mathscr{A}, \forall t \in [ES_i \dots LC_i - 1] \tag{23}$$

   Then (19) can be replaced by an SAT encoding of an at-most-one (AMO) constraint:

   $$AMO_{\substack{A_i \in \mathscr{A}_g,\ \text{s.t.}\\ t \in [ES_i \dots LC_i - 1],\\ \mathfrak{b}_{i,r} > 0}} (x_{i,t})$$
   $$\forall R_r \in \mathscr{R}, \forall t \in [0, H] \tag{24}$$

   In particular, we use two encodings for constraints(24): regular/ladder encoding (see Biere et al., 2009), and the pairwise encoding, which introduces the follow clauses:

   $$\overline{x_{i,t}} \vee \overline{x_{j,t}} \quad \begin{array}{c}\forall A_i, A_j \in \mathscr{A}_g, i < j,\\ \forall t \in [ES_i \dots LC_i - 1] \cap [ES_j \dots LC_j - 1],\ \text{s.t.}\\ \exists R_r \in \mathscr{R}, \mathfrak{b}_{i,r} > 0, \mathfrak{b}_{j,r} > 0\end{array} \tag{25}$$

   Notice that if $(A_i, A_j, l_{i,j}) \in \mathscr{E}_g^*$ and $l_{i,j} > 0$ or $(A_j, A_i, l_{j,i}) \in \mathscr{E}_g^*$ and $l_{j,i} > 0$ then:

   - If $l_{i,j} > \mathfrak{d}_i$ or $l_{j,i} > \mathfrak{d}_j$ it is impossible that these two activities are simultaneously executed, therefore constraints (22) and the pairwise encoding (25) do not introduce clauses for such pairs of activities.
   - Otherwise, there is a relative order between these activities (w.l.o.g. $\mathfrak{d}_i \geq l_{i,j} > 0$) and only one part of the disjunction of (22) is needed (i.e. $(S_j \geq S_i + \mathfrak{d}_i)$).

2. **Cumulative Constraints**. As with renewable resource constraints, we can use Boolean variables $y_{i,t}$ instead of integer variables $Y_{i,t}$ to deal with cumulative constraints, i.e., $y_{i,t}$ Boolean variables that are true iff $A_i$ has already started at time $t$, $\forall A_i \in \mathscr{A}_g, \forall t \in [ES_i \dots LS_i - 1]$, thus replacing constraints (17) and (18) with the following:

   $$y_{i,t} \leftrightarrow (S_i \leq t) \quad \begin{array}{c}\forall A_i \in \mathscr{A}_g, \forall t \in [ES_i \dots LS_i - 1],\ \text{s.t.},\\ \exists C_s \in \mathscr{C}, \mathfrak{c}_{i,s} \neq 0\end{array} \tag{26}$$

   Using Boolean variables $y_{i,t}$ and the SAT encoding of cardinality constraints presented in Abío et al. (2013), we can replace

**Table 1**
Test Type description.

| Test | Act. | Prec. | LB | Renewable resources | | | Storages areas | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Shuttle | S. Arm | R. Arm | Cold | Hot-I | Hot-II | Obser. |
| Type 1 | 7 | 8 | 25 | 3 | 1 | 1 | 0 | 1 | 0 | 1 |
| Type 2 | 14 | 17 | 62 | 6 | 5 | 1 | 2 | 0 | 1 | 1 |
| Type 3 | 9 | 11 | 44 | 4 | 1 | 2 | 0 | 1 | 1 | 1 |
| Type 4 | 9 | 10 | 38 | 4 | 1 | 2 | 1 | 0 | 1 | 1 |

constraints (20) with:

$$PS_{s,t} + \left( \sum_{\substack{A_i \in \mathscr{A}_g,\ \text{s.t.} \\ t \in [ES_i \dots LS_i], \\ \mathfrak{c}_{i,s} > 0}} y_{i,t} \right) + \left( \sum_{\substack{A_i \in \mathscr{A}_g,\ \text{s.t.} \\ t \in [EC_i \dots LC_i - 1], \\ \mathfrak{c}_{i,s} < 0}} (\overline{y_{i,(t-\mathfrak{d}_i)}} - 1) \right) \leq P_s$$

$$\forall C_s \in \mathscr{C}, \forall t \in [0 \dots H] \tag{27}$$

## 7. MILP formulation for the SAMSP

Our formulation is based on the Disaggregated Time-Indexed (DDT) MILP Formulation for RCPSP. This was formulated by Christofides et al. (1987) for RCPSP, and is very similar to the Time-Indexed (DT) Formulation of Pritsker et al. (1969). The two formulations mainly differ in how they formulate the precedence constraints. This is the best formulation in most cases; as can be seen in Artigues (2017). It only uses a set of binary decision variables $\mathfrak{x}_{i,t} \in \{0, 1\}$ that are 1 if $A_i$ starts at time $t$, or 0 otherwise, $\forall A_i \in \mathscr{V}_g, \forall t \in [ES_i \dots LS_i]$. Our formulation is based on Chaleshtarti and Shadrokh (2011).

The preprocessing and the symmetry breaking detailed in Section 6.4 have also been used in the MILP formulation, with a notable reduction in encoding size and resolution time.

### 7.1. MILP formulation

$$\min \sum_{t \in [ES_{n+1} \dots LS_{n+1}]} t \cdot \mathfrak{x}_{n+1,t} \tag{28}$$

s.t.

$$\sum_{t \in [ES_i \dots LS_i]} \mathfrak{x}_{i,t} = 1 \quad \forall A_i \in \mathscr{V}_g \tag{29}$$

$$\sum_{t \in [ES_j \dots LS_j]} t \cdot \mathfrak{x}_{j,t} - \sum_{t \in [ES_i \dots LS_i]} t \cdot \mathfrak{x}_{i,t} \geq l_{i,j} \quad \forall (A_i, A_j, l_{i,j}) \in \mathscr{E}_g^* \tag{30}$$

$$\left( \sum_{\substack{A_i \in \mathscr{A}_g,\ \text{s.t.} \\ t \in [ES_i \dots LC_i - 1], \\ \mathfrak{b}_{i,r} > 0}} \left( \sum_{z=\max(ES_i, t-p_i+1)}^{t} \mathfrak{b}_{i,r} \cdot \mathfrak{x}_{i,z} \right) \right) \leq B_r$$

$$\forall R_r \in \mathscr{R}, \forall t \in [0 \dots H] \tag{31}$$

$$PS_{s,t} + \left( \sum_{\substack{A_i \in \mathscr{A}_g,\ \text{s.t.} \\ t \in [ES_i \dots LS_i - 1], \\ \mathfrak{c}_{i,s} > 0}} \left( \sum_{z=ES_i}^{t} \mathfrak{c}_{i,s} \cdot \mathfrak{x}_{i,z} \right) \right) + \left( \sum_{\substack{A_i \in \mathscr{A}_g,\ \text{s.t.} \\ t \in [EC_i \dots LC_i - 1], \\ \mathfrak{c}_{i,s} < 0}} \left( \sum_{z=EC_i}^{t} \mathfrak{c}_{i,s} \cdot \mathfrak{x}_{i,(z-\mathfrak{d}_i)} \right) \right) \leq P_s$$

$$\forall C_s \in \mathscr{C}, \forall t \in [0 \dots H] \tag{32}$$

The objective is (28), minimize the makespan. Constraints (29) state that each activity has to be started exactly once in the scheduling horizon. Constraints (30) are the precedence constraints. Constraints (31) are the renewable resource constraints and Constraints (32) are the cumulative resource constraints.

### 7.2. Refinements

As stated above, in the machine architecture considered, all renewable resources are unary ($B_r = 1$) and demands on them are also unary ($\mathfrak{b}_{i,r} = 1$). Therefore, we can replace renewable resource constraints (31) with a disjunction between the execution of activities requiring the same resource:

$$\mathfrak{x}_{i,t} + \sum_{z=\max(ES_i, t-d_j+1)}^{\min(t+d_i-1, LS_j)} \mathfrak{x}_{j,z} \leq 1 \qquad \begin{array}{l} \forall A_i, A_j \in \mathscr{A}_g, i < j, \forall t \in [ES_i \dots LS_i], \text{ s.t.} \\ [ES_i \dots LC_i - 1] \cap [ES_j \dots LC_j - 1] \neq \emptyset, \\ \exists R_r \in \mathscr{R}, \mathfrak{b}_{i,r} > 0, \mathfrak{b}_{j,r} > 0 \end{array} \tag{33}$$

The disjunction between activities $A_i$ and $A_j$ is enforced by forbidding $A_j$ to run when $A_i$ starts. For this purpose, we consider all time instants $z$ such that if activity $A_j$ starts at time $z$, it will be running at time $t$. Time windows are also taken into account when considering the ranges of times $t$ and $z$.

## 8. Experiments

We ran our experiments on an 8 GB Intel® Xeon® E3-1220v2 machine at 3.10 GHz. For our experiments, we consider four different test types for the actual machine layout described in Section 3.1. In Sections 8.1–8.4, we experiment with SMT, CP and MILP with the first machine configuration proposal from the company itself, which is $\langle 4, 4, 4, 2 \rangle$, i.e., there are four positions for Cold, Hot-I and Hot-II Storage areas, and two positions for the Observation Area. In Section 8.5, we experiment with different configurations of the machine.

In Table 1 we summarize the main characteristics of the test types that we consider for our experiments. We recall that these test types are based on the ones that our client was using. These test types involve activities of short duration ranging from three to seven time units (corresponding to fast shuttle or arm operations), but with larger time lag, of up to 30 time units (corresponding to the durations of the different chemical processes), between them. For each test type, we indicate the number of (non-dummy) activities, the total number of precedences between activities, the initial LB, the number of activities requiring each renewable resource (shuttle, sample arm, reagent arm) and the number of activities that occupy storage resources. Recall that, in our layout an activity corresponds to either a shuttle movement or an arm action and therefore only one renewable resource is required by each activity. Furthermore, in our test types, for each activity that occupies a storage position, there is another activity that vacates that position. Notice, therefore, that a single test of a particular type by itself is not a hard problem to solve, since in most cases the activities requiring the resource must be executed in a certain order defined by the precedence relations. Nevertheless, the instances we consider are comprised of several tests, and therefore many activities may require the same resource.

We name the instances following the nomenclature $I_L$, where $L = [\langle i, j \rangle]$ is a list of pairs in which $i$ is a test type identifier and $j$ is the number of required tests of test type $i$. The instances are divided in two sets: Set I contains basic instances and Set II contains more complex

**Table 2**
Set I Instances Description.

| Instance | Opt. | Act. | Precedences | | Act. demanding | | LB initial | |
|---|---|---|---|---|---|---|---|---|
| | | | w/o SB | w SB | Renewable | Storage | w/o SB | w SB |
| $I_{[\langle 1,5\rangle]}$ | 59 | 27 | 131 | 341 | 25 | 10 | 25 | 59 |
| $I_{[\langle 1,10\rangle]}$ | 102 | 52 | 261 | 1206 | 50 | 20 | 25 | 100 |
| $I_{[\langle 1,20\rangle]}$ | 192 | 102 | 521 | 4511 | 100 | 40 | 25 | 185 |
| $I_{[\langle 2,5\rangle]}$ | 173 | 62 | 541 | 1501 | 60 | 20 | 62 | 128 |
| $I_{[\langle 2,10\rangle]}$ | 297 | 122 | 1081 | 5401 | 120 | 40 | 62 | 200 |
| $I_{[\langle 2,20\rangle]}$ | 548 | 242 | 2161 | 20401 | 240 | 80 | 62 | 365 |
| $I_{[\langle 3,5\rangle]}$ | 89 | 37 | 251 | 681 | 35 | 15 | 44 | 88 |
| $I_{[\langle 3,10\rangle]}$ | 145 | 72 | 501 | 2436 | 70 | 30 | 44 | 139 |
| $I_{[\langle 3,20\rangle]}$ | 264 | 142 | 1001 | 9171 | 140 | 60 | 44 | 249 |
| $I_{[\langle 4,5\rangle]}$ | 69 | 37 | 206 | 546 | 35 | 15 | 38 | 66 |
| $I_{[\langle 4,10\rangle]}$ | 124 | 72 | 411 | 1941 | 70 | 30 | 38 | 101 |
| $I_{[\langle 4,20\rangle]}$ | 244 | 142 | 821 | 7281 | 140 | 60 | 38 | 171 |
| $I_{[\langle 1,5\rangle,\langle 2,5\rangle]}$ | 197 | 87 | 671 | 1891 | 85 | 30 | 62 | 128 |
| $I_{[\langle 1,10\rangle,\langle 2,10\rangle]}$ | 357 | 172 | 1341 | 6606 | 170 | 60 | 62 | 200 |
| $I_{[\langle 1,5\rangle,\langle 3,5\rangle]}$ | 122 | 62 | 381 | 1021 | 60 | 25 | 44 | 88 |
| $I_{[\langle 1,10\rangle,\langle 3,10\rangle]}$ | 224 | 122 | 761 | 3641 | 120 | 50 | 44 | 139 |
| $I_{[\langle 1,5\rangle,\langle 4,5\rangle]}$ | 109 | 62 | 336 | 866 | 60 | 25 | 38 | 66 |
| $I_{[\langle 1,10\rangle,\langle 4,10\rangle]}$ | 214 | 122 | 671 | 3146 | 120 | 50 | 38 | 101 |
| $I_{[\langle 2,5\rangle,\langle 3,5\rangle]}$ | 205 | 97 | 791 | 2131 | 95 | 35 | 62 | 128 |
| $I_{[\langle 2,10\rangle,\langle 3,10\rangle]}$ | 365 | 192 | 1581 | 7836 | 190 | 70 | 62 | 200 |
| $I_{[\langle 2,5\rangle,\langle 4,5\rangle]}$ | 201 | 97 | 746 | 2056 | 95 | 35 | 62 | 128 |
| $I_{[\langle 2,10\rangle,\langle 4,10\rangle]}$ | 361 | 192 | 1491 | 7341 | 190 | 70 | 62 | 191 |
| $I_{[\langle 3,5\rangle,\langle 4,5\rangle]}$ | 124 | 72 | 456 | 1226 | 70 | 30 | 44 | 88 |
| $I_{[\langle 3,10\rangle,\langle 4,10\rangle]}$ | 244 | 142 | 911 | 4376 | 140 | 60 | 44 | 139 |

**Table 3**
Set II Instances Description.

| Instance | Opt. | Act. | Precedences | | Act. demanding | | LB initial | |
|---|---|---|---|---|---|---|---|---|
| | | | w/o SB | w SB | Renewable | Storage | w/o SB | w SB |
| $I_{[\langle 1,60\rangle]}$ | 552 | 302 | 1561 | 38731 | 300 | 120 | 25 | 525 |
| $I_{[\langle 1,120\rangle]}$ | 1092 | 602 | 3121 | 153061 | 600 | 240 | 25 | 1035 |
| $I_{[\langle 2,60\rangle]}$ | 1548 | 722 | 6481 | 176401 | 720 | 240 | 62 | 1025 |
| $I_{[\langle 2,120\rangle]}$ | 3048 | 1442 | 12961 | 698401 | 1440 | 480 | 62 | 2015 |
| $I_{[\langle 3,60\rangle]}$ | unk | 422 | 3001 | 79111 | 420 | 180 | 44 | 689 |
| $I_{[\langle 3,120\rangle]}$ | unk | 842 | 6001 | 313021 | 840 | 360 | 44 | 1349 |
| $I_{[\langle 4,60\rangle]}$ | 724 | 422 | 2461 | 62641 | 420 | 180 | 38 | 451 |
| $I_{[\langle 4,120\rangle]}$ | 1444 | 842 | 4921 | 247681 | 840 | 360 | 38 | 871 |
| $I_{[\langle 1,20\rangle,\langle 2,20\rangle]}$ | 677 | 342 | 2681 | 24911 | 340 | 120 | 62 | 365 |
| $I_{[\langle 1,20\rangle,\langle 3,20\rangle]}$ | 432 | 242 | 1521 | 13681 | 240 | 100 | 44 | 249 |
| $I_{[\langle 1,20\rangle,\langle 4,20\rangle]}$ | 424 | 242 | 1341 | 11791 | 240 | 100 | 38 | 185 |
| $I_{[\langle 2,20\rangle,\langle 3,20\rangle]}$ | 685 | 382 | 3161 | 29571 | 380 | 140 | 62 | 365 |
| $I_{[\langle 3,20\rangle,\langle 4,20\rangle]}$ | 681 | 382 | 2981 | 27681 | 380 | 140 | 62 | 365 |
| $I_{[\langle 3,20\rangle,\langle 4,20\rangle]}$ | 484 | 282 | 1821 | 16451 | 280 | 120 | 44 | 249 |
| $I_{[\langle 1,5\rangle,\langle 2,5\rangle,\langle 3,5\rangle]}$ | 232 | 122 | 921 | 2521 | 120 | 45 | 62 | 128 |
| $I_{[\langle 1,5\rangle,\langle 2,5\rangle,\langle 4,5\rangle]}$ | 232 | 122 | 876 | 2386 | 120 | 45 | 62 | 128 |
| $I_{[\langle 1,5\rangle,\langle 3,5\rangle,\langle 4,5\rangle]}$ | 169 | 97 | 586 | 1566 | 95 | 40 | 44 | 88 |
| $I_{[\langle 2,5\rangle,\langle 3,5\rangle,\langle 4,5\rangle]}$ | 236 | 132 | 996 | 2726 | 130 | 50 | 62 | 128 |
| $I_{[\langle 1,10\rangle,\langle 2,10\rangle,\langle 3,10\rangle]}$ | 427 | 242 | 1841 | 9041 | 240 | 90 | 62 | 200 |
| $I_{[\langle 1,10\rangle,\langle 2,10\rangle,\langle 4,10\rangle]}$ | 427 | 242 | 1751 | 8546 | 240 | 90 | 62 | 200 |
| $I_{[\langle 1,10\rangle,\langle 3,10\rangle,\langle 4,10\rangle]}$ | 334 | 192 | 1171 | 5581 | 190 | 80 | 44 | 139 |
| $I_{[\langle 2,10\rangle,\langle 3,10\rangle,\langle 4,10\rangle]}$ | 431 | 262 | 1991 | 9776 | 260 | 100 | 62 | 200 |
| $I_{[\langle 1,5\rangle,\langle 2,5\rangle,\langle 3,5\rangle,\langle 4,5\rangle]}$ | 267 | 157 | 1126 | 3066 | 155 | 60 | 62 | 128 |
| $I_{[\langle 1,10\rangle,\langle 2,10\rangle,\langle 3,10\rangle,\langle 4,10\rangle]}$ | unk | 312 | 2251 | 10981 | 310 | 120 | 62 | 200 |

instances. Set II is only run with CP, which is the only setting able to solve all instances of Set I with small running times.

The different instances that we consider capture the most common and interesting test sets according to our customer's requirements. A commonly used benchmarking setting is to test the throughput of an analysis machine given a certain number of tests of the same type, such as instances $I_{[\langle 1,20\rangle]}$ or $I_{[\langle 1,60\rangle]}$. Also, it is interesting to consider instances with many combinations of different test types, like $I_{[\langle 3,10\rangle,\langle 4,10\rangle]}$ or $I_{[\langle 1,5\rangle,\langle 2,5\rangle,\langle 3,5\rangle,\langle 4,5\rangle]}$, in order to detect patterns in the optimal solutions in distinct scenarios. A common case is to receive a set of samples and to perform the same test types on all of them, hence having the same number of tests for each type.

Tables 2 and 3 describe the sets of instances: the column *Instance* contains the instance name; column *Opt.* the optimum makespan of the instances (or – if unknown); column *Act.* the total number of activities; columns *Precedences* report the total number of extended precedence relations without and with the symmetry breaking precedences explained in Section 4.2; columns *Act. demanding* report the number of activities demanding a renewable resource (column *Renewable*) and the number of activities that occupy a storage position (column *Storage*); columns *LB initial* contain the initial LB of the makespan, computed from the extended precedence graph without and with the symmetry breaking precedences. The number of activities in the considered instances range from 27 to 242 in Set I, and from 97 to 1442 in Set II. It can be observed that the number of precedence relations grows significantly when using symmetry breaking, especially in the instances where there is a large number of tests of a same type. This increase in the number of precedences results in an improvement in the computed LB of the makespan. In fact, when symmetry breaking is not used, the LB

**Table 4**
Set I. Performance comparison of CP encoding variants. Time in seconds (time limit 600).

| Instance | Opt. | $CP$ | $CP_{sb}$ | $CP_{no}$ | $CP_{sbp}$ | $CP_{nop}$ |
|---|---|---|---|---|---|---|
| $I_{\{1,5\}}$ | 59 | 300 | <1 | <1 | <1 | <1 |
| $I_{\{1,10\}}$ | 102 | 334 | <1 | <1 | <1 | <1 |
| $I_{\{1,20\}}$ | 192 | – | <1 | <1 | 1 | <1 |
| $I_{\{2,5\}}$ | 173 | <1 | <1 | <1 | <1 | <1 |
| $I_{\{2,10\}}$ | 298 | 2 | <1 | <1 | <1 | <1 |
| $I_{\{2,20\}}$ | 548 | 103 | <1 | <1 | <1 | 1 |
| $I_{\{3,5\}}$ | 89 | 1 | <1 | <1 | <1 | <1 |
| $I_{\{3,10\}}$ | 145 | – | <1 | <1 | <1 | <1 |
| $I_{\{3,20\}}$ | 264 | – | 3 | 4 | 3 | 5 |
| $I_{\{4,5\}}$ | 69 | <1 | <1 | <1 | <1 | <1 |
| $I_{\{4,10\}}$ | 124 | 17 | <1 | <1 | <1 | <1 |
| $I_{\{4,20\}}$ | 244 | – | <1 | <1 | <1 | <1 |
| $I_{\{1,5\},\{2,5\}}$ | 197 | 5 | <1 | <1 | 1 | 1 |
| $I_{\{1,10\},\{2,10\}}$ | 357 | – | 7 | 5 | 7 | 6 |
| $I_{\{1,5\},\{3,5\}}$ | 122 | 191 | 2 | 1 | 1 | 2 |
| $I_{\{1,10\},\{3,10\}}$ | 224 | – | 27 | 23 | 40 | 38 |
| $I_{\{1,5\},\{4,5\}}$ | 109 | 3 | <1 | <1 | <1 | <1 |
| $I_{\{1,10\},\{4,10\}}$ | 214 | – | 1 | 1 | 2 | 2 |
| $I_{\{2,5\},\{3,5\}}$ | 205 | 4 | 1 | 1 | 1 | 2 |
| $I_{\{2,10\},\{3,10\}}$ | 365 | 14 | 13 | 17 | 13 | 29 |
| $I_{\{2,5\},\{4,5\}}$ | 201 | 9 | 1 | 1 | 5 | 4 |
| $I_{\{2,10\},\{4,10\}}$ | 361 | – | 15 | 18 | 21 | 17 |
| $I_{\{3,5\},\{4,5\}}$ | 124 | 4 | 1 | 1 | 2 | 1 |
| $I_{\{3,10\},\{4,10\}}$ | 244 | 262 | 2 | 4 | 9 | 6 |
| $TOTALS$ | 24 | 16 | 24 | 24 | 24 | 24 |

**Table 5**
Set II. Performance comparison of CP encoding variants for difficult instances. Time in seconds (time limit 600).

| Instance | Opt. | $CP$ | $CP_{sb}$ | $CP_{no}$ | $CP_{sbp}$ | $CP_{nop}$ |
|---|---|---|---|---|---|---|
| $I_{\{1,60\}}$ | 552 | – | 9 | 9 | 7 | 7 |
| $I_{\{1,120\}}$ | 1092 | – | 103 | 75 | 83 | 97 |
| $I_{\{2,60\}}$ | 1548 | – | 2 | 2 | 10 | 13 |
| $I_{\{2,120\}}$ | 3048 | – | 11 | 11 | 70 | 63 |
| $I_{\{3,60\}}$ | unk | – | – | – | – | – |
| $I_{\{3,120\}}$ | unk | – | – | – | – | – |
| $I_{\{4,60\}}$ | 724 | – | 1 | 2 | 2 | 2 |
| $I_{\{4,120\}}$ | 1444 | – | 10 | 10 | 12 | 9 |
| $I_{\{1,20\},\{2,20\}}$ | 677 | – | 53 | 50 | 78 | 76 |
| $I_{\{1,20\},\{3,20\}}$ | 432 | – | 495 | – | – | – |
| $I_{\{1,20\},\{4,20\}}$ | 424 | – | 8 | 7 | 26 | 13 |
| $I_{\{2,20\},\{3,20\}}$ | 685 | – | 130 | 57 | 56 | 96 |
| $I_{\{2,20\},\{4,20\}}$ | 681 | – | 309 | 157 | 328 | 271 |
| $I_{\{3,20\},\{4,20\}}$ | 484 | – | 12 | 12 | 19 | 17 |
| $I_{\{1,5\},\{2,5\},\{3,5\}}$ | 232 | 67 | 6 | 4 | 6 | 11 |
| $I_{\{1,5\},\{2,5\},\{4,5\}}$ | 232 | – | 9 | 13 | 12 | 11 |
| $I_{\{1,5\},\{3,5\},\{4,5\}}$ | 169 | 19 | 3 | 3 | 8 | 4 |
| $I_{\{2,5\},\{3,5\},\{4,5\}}$ | 236 | 194 | 8 | 6 | 11 | 10 |
| $I_{\{1,10\},\{2,10\},\{3,10\}}$ | 427 | – | 123 | 44 | 79 | 93 |
| $I_{\{1,10\},\{2,10\},\{4,10\}}$ | 427 | – | 260 | 184 | 373 | 167 |
| $I_{\{1,10\},\{3,10\},\{4,10\}}$ | 334 | – | 28 | 15 | 28 | 20 |
| $I_{\{2,10\},\{3,10\},\{4,10\}}$ | 431 | – | 232 | 240 | 340 | 285 |
| $I_{\{1,5\},\{2,5\},\{3,5\},\{4,5\}}$ | 267 | – | 69 | 70 | 109 | 84 |
| $I_{\{1,10\},\{2,10\},\{3,10\},\{4,10\}}$ | unk | – | – | – | – | – |
| $TOTALS$ | 24 | 3 | 21 | 20 | 20 | 20 |

corresponds to the largest LB among the different test types included in the instance.

### 8.1. CP experiments

In the CP experiments we have used the IBM ILOG CP Optimizer V12.9.0 (IBM, 2019) as the CP engine and we have considered the following encodings:

- $CP$: the basic encoding provided in Section 5.1, consisting of a *Pulse* formulation for the renewable resource constraints. Symmetry breaking is not applied.
- $CP_{sb}$: $CP$ plus symmetry breaking.
- $CP_{no}$: the encoding refinement provided in Section 5.2, where no-overlap formulation is used for the renewable resource constraints. Symmetry breaking is also applied.
- $CP_{sbp}$: $CP_{sb}$ using *Pulse* function for cumulative resources.
- $CP_{nop}$: $CP_{no}$ using *Pulse* function for cumulative resources.

Table 4 reports on the utilization of CP Optimizer for our problem for the instances in Set I. The first thing to observe is that using symmetry breaking makes a huge difference in the performance of the solver, since both $CP_{sb}$ and $CP_{no}$ are able to optimally solve all the instances and, in most cases, in less than one second. On the contrary, if symmetry breaking is not used, the solver is unable to certify the optimality of eight instances. Moreover, among the optimally certified instances, it requires more than 100 s in some cases.

**Table 6**
Set I. Performance comparison of SMT encoding variants. Time in seconds (time limit 600).

| Instance | *Opt* | $SMT$ | $SMT_{sb}$ | Resources | | | $SMT_{amo,card}$ |
|---|---|---|---|---|---|---|---|
| | | | | $SMT_{amo}$ | $SMT_b$ | $SMT_{dj}$ | |
| $I_{[\langle 1,5\rangle]}$ | 59 | <1 | <1 | <1 | <1 | <1 | <1 |
| $I_{[\langle 1,10\rangle]}$ | 102 | – | 1 | 1 | 1 | 1 | 2 |
| $I_{[\langle 1,20\rangle]}$ | 192 | – | 7 | 5 | 6 | 6 | 14 |
| $I_{[\langle 2,5\rangle]}$ | 173 | 20 | 3 | 2 | 3 | 2 | 4 |
| $I_{[\langle 2,10\rangle]}$ | 298 | – | 42 | 17 | 25 | 54 | 36 |
| $I_{[\langle 2,20\rangle]}$ | 548 | – | – | 152 | 202 | – | 427 |
| $I_{[\langle 3,5\rangle]}$ | 89 | 1 | 1 | 1 | 1 | <1 | 1 |
| $I_{[\langle 3,10\rangle]}$ | 145 | – | 2 | 2 | 2 | 2 | 5 |
| $I_{[\langle 3,20\rangle]}$ | 264 | – | 130 | 92 | 91 | 255 | 96 |
| $I_{[\langle 4,5\rangle]}$ | 69 | 1 | <1 | <1 | <1 | <1 | 1 |
| $I_{[\langle 4,10\rangle]}$ | 124 | – | 4 | 3 | 3 | 14 | 6 |
| $I_{[\langle 4,20\rangle]}$ | 244 | – | 162 | 148 | 152 | – | 139 |
| $I_{[\langle 1,5\rangle,\langle 2,5\rangle]}$ | 197 | – | 16 | 11 | 12 | 237 | 15 |
| $I_{[\langle 1,10\rangle,\langle 2,10\rangle]}$ | 357 | – | – | – | – | – | – |
| $I_{[\langle 1,5\rangle,\langle 3,5\rangle]}$ | 122 | – | 4 | 3 | 4 | 8 | 5 |
| $I_{[\langle 1,10\rangle,\langle 3,10\rangle]}$ | 224 | – | – | – | – | – | – |
| $I_{[\langle 1,5\rangle,\langle 4,5\rangle]}$ | 109 | – | 75 | 51 | 57 | 489 | 81 |
| $I_{[\langle 1,10\rangle,\langle 4,10\rangle]}$ | 214 | – | – | – | – | – | – |
| $I_{[\langle 2,5\rangle,\langle 3,5\rangle]}$ | 205 | – | 23 | 14 | 16 | – | 23 |
| $I_{[\langle 2,10\rangle,\langle 3,10\rangle]}$ | 365 | – | – | – | – | – | – |
| $I_{[\langle 2,5\rangle,\langle 4,5\rangle]}$ | 201 | – | 24 | 12 | 15 | 374 | 26 |
| $I_{[\langle 2,10\rangle,\langle 4,10\rangle]}$ | 361 | – | – | – | – | – | – |
| $I_{[\langle 3,5\rangle,\langle 4,5\rangle]}$ | 124 | – | 57 | 64 | 81 | 240 | 54 |
| $I_{[\langle 3,10\rangle,\langle 4,10\rangle]}$ | 244 | – | – | – | – | – | – |
| $TOTALS$ | 24 | 4 | 17 | 18 | 18 | 15 | 18 |

We observe very similar results for $CP_{sb}$ and $CP_{no}$, and there appears to be no significant difference when using the *Pulse* function for cumulative resources in the $CP_{sbp}$ and $CP_{nop}$ encodings. As the results for these four encodings are extraordinarily good, we have carried out tests with more difficult instances, corresponding to Set II, see Table 5. In these difficult instances the need of symmetry breaking is still more evident, since only three optimums are certified if not used. We also observe that, although $CP_{sb}$ optimally certifies one more instance, the times of $CP_{no}$ and $CP_{nop}$ are a bit better in general. The $CP_{sbp}$ encoding seems to be the one that exhibits the worst solving times in more instances. Finally, $CP_{sb}$, $CP_{no}$, $CP_{sbp}$ and $CP_{nop}$ are able to find some solution for all instances of Set II.

### 8.2. SMT experiments

In the SMT experiments we have used Yices 2.6.1 (Dutertre and de Moura, 2006) as the core SMT solver. We have considered the following incremental combinations of encodings:

- $SMT$: the basic encoding provided in Section 6.2, without symmetry breaking.
- $SMT_{sb}$: $S$ plus symmetry breaking.
- $SMT_{amo}$: $SMT_{sb}$ with AMO constraints for resource constraints (24) with a regular/ladder encoding.
- $SMT_b$: $SMT_{sb}$ with a Boolean pairwise encoding (25) of renewable resource constraints.
- $SMT_{dj}$: $SMT_{sb}$ with disjunction for resources constraints (22).
- $SMT_{amo,card}$: $SMT_{amo}$ with cardinality constraints for storage constraints (27).

Table 6 contains the results for the different SMT encodings. In SMT, using symmetry breaking is also decisive since $SMT$ only certifies four optimums while $SMT_{sb}$ certifies 17. Regarding the variants of the constraints on renewable resources (19), $SMT_{amo}$ and $SMT_b$ are the only variants that really improve solving times compared to $SMT_{sb}$. From between these two, the one with best solving times is $SMT_{amo}$. Configuration $SMT_{dj}$ is clearly worse than $SMT_{sb}$. On the other hand, we can observe that the use of SAT encodings of cardinality constraint (27) to deal with cumulative resource constraints in setting $SMT_{amo,card}$ does not improve $SMT_{amo}$, which uses the LIA theory.

**Table 7**
Set I. Performance comparison of MILP encoding variants. Time in seconds (time limit 600).

| Instance | Opt. | $MILP$ | $MILP_{sb}$ | $MILP_d$ |
|---|---|---|---|---|
| $I_{[\langle 1,5\rangle]}$ | 59 | 17 | <1 | <1 |
| $I_{[\langle 1,10\rangle]}$ | 102 | – | 2 | 7 |
| $I_{[\langle 1,20\rangle]}$ | 192 | – | 243 | – |
| $I_{[\langle 2,5\rangle]}$ | 173 | – | – | – |
| $I_{[\langle 2,10\rangle]}$ | 298 | – | – | – |
| $I_{[\langle 2,20\rangle]}$ | 548 | – | – | – |
| $I_{[\langle 3,5\rangle]}$ | 89 | 436 | 1 | 1 |
| $I_{[\langle 3,10\rangle]}$ | 145 | – | 135 | – |
| $I_{[\langle 3,20\rangle]}$ | 264 | – | – | – |
| $I_{[\langle 4,5\rangle]}$ | 69 | 213 | 1 | <1 |
| $I_{[\langle 4,10\rangle]}$ | 124 | – | 96 | – |
| $I_{[\langle 4,20\rangle]}$ | 244 | – | – | – |
| $I_{[\langle 1,5\rangle,\langle 2,5\rangle]}$ | 197 | – | – | – |
| $I_{[\langle 1,10\rangle,\langle 2,10\rangle]}$ | 357 | – | – | – |
| $I_{[\langle 1,5\rangle,\langle 3,5\rangle]}$ | 122 | – | – | – |
| $I_{[\langle 1,10\rangle,\langle 3,10\rangle]}$ | 224 | – | – | – |
| $I_{[\langle 1,5\rangle,\langle 4,5\rangle]}$ | 109 | – | 525 | – |
| $I_{[\langle 1,10\rangle,\langle 4,10\rangle]}$ | 214 | – | – | – |
| $I_{[\langle 2,5\rangle,\langle 3,5\rangle]}$ | 205 | – | – | – |
| $I_{[\langle 2,10\rangle,\langle 3,10\rangle]}$ | 365 | – | – | – |
| $I_{[\langle 2,5\rangle,\langle 4,5\rangle]}$ | 201 | – | – | – |
| $I_{[\langle 2,10\rangle,\langle 4,10\rangle]}$ | 361 | – | – | – |
| $I_{[\langle 3,5\rangle,\langle 4,5\rangle]}$ | 124 | – | – | – |
| $I_{[\langle 3,10\rangle,\langle 4,10\rangle]}$ | 244 | – | – | – |
| $TOTALS$ | 24 | 3 | 8 | 4 |

We have observed that Set II is too hard for any of the SMT encodings. The best encoding, $E_{amo}$, is only able to certify the optimality of the instance $I[\langle 1,60\rangle]$ and find a solution for nine instances more.

### 8.3. MILP experiments

In the MILP experiments we have used the IBM ILOG CPLEX V12.9.0 (IBM, 2019) as the MILP solver. We have considered the following encodings:

- $MILP$: the basic encoding provided in Section 7.1, without symmetry breaking.

**Table 8**
Set I Comparison CP/SMT/MILP. Time in seconds (time limit 600).

| Instance | Opt. | CP | | | | SMT | | | | MILP | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Time | LB | UB | Gap | Time | LB | UB | Gap | Time | LB | UB | Gap |
| $I_{[\langle 1,5\rangle]}$ | 59 | <1 | 59 | 59 | 0.0 | <1 | 59 | 59 | 0.0 | <1 | 59 | 59 | 0.0 |
| $I_{[\langle 1,10\rangle]}$ | 102 | <1 | 102 | 102 | 0.0 | 1 | 102 | 102 | 0.0 | 2 | 102 | 102 | 0.0 |
| $I_{[\langle 1,20\rangle]}$ | 192 | <1 | 192 | 192 | 0.0 | 5 | 192 | 192 | 0.0 | 243 | 192 | 192 | 0.0 |
| $I_{[\langle 2,5\rangle]}$ | 173 | <1 | 173 | 173 | 0.0 | 2 | 173 | 173 | 0.0 | – | 172 | 175 | 1.7 |
| $I_{[\langle 2,10\rangle]}$ | 298 | <1 | 298 | 298 | 0.0 | 17 | 298 | 298 | 0.0 | – | – | – | – |
| $I_{[\langle 2,20\rangle]}$ | 548 | <1 | 548 | 548 | 0.0 | 152 | 548 | 548 | 0.0 | – | – | – | – |
| $I_{[\langle 3,5\rangle]}$ | 89 | <1 | 89 | 89 | 0.0 | 1 | 89 | 89 | 0.0 | 1 | 89 | 89 | 0.0 |
| $I_{[\langle 3,10\rangle]}$ | 145 | <1 | 145 | 145 | 0.0 | 2 | 145 | 145 | 0.0 | 135 | 145 | 145 | 0.0 |
| $I_{[\langle 3,20\rangle]}$ | 264 | 4 | 264 | 264 | 0.0 | 92 | 264 | 264 | 0.0 | – | – | – | – |
| $I_{[\langle 4,5\rangle]}$ | 69 | <1 | 69 | 69 | 0.0 | <1 | 69 | 69 | 0.0 | <1 | 69 | 69 | 0.0 |
| $I_{[\langle 4,10\rangle]}$ | 124 | <1 | 124 | 124 | 0.0 | 3 | 124 | 124 | 0.0 | 96 | 124 | 124 | 0.0 |
| $I_{[\langle 4,20\rangle]}$ | 244 | <1 | 244 | 244 | 0.0 | 148 | 244 | 244 | 0.0 | – | – | – | – |
| $I_{[\langle 1,5\rangle,\langle 2,5\rangle]}$ | 197 | <1 | 197 | 197 | 0.0 | 11 | 197 | 197 | 0.0 | – | – | – | – |
| $I_{[\langle 1,10\rangle,\langle 2,10\rangle]}$ | 357 | 5 | 357 | 357 | 0.0 | – | 351 | 357 | 1.7 | – | – | – | – |
| $I_{[\langle 1,5\rangle,\langle 3,5\rangle]}$ | 122 | 1 | 122 | 122 | 0.0 | 3 | 122 | 122 | 0.0 | – | 112 | 142 | 21.1 |
| $I_{[\langle 1,10\rangle,\langle 3,10\rangle]}$ | 224 | 23 | 224 | 224 | 0.0 | – | 218 | 225 | 3.2 | – | – | – | – |
| $I_{[\langle 1,5\rangle,\langle 4,5\rangle]}$ | 109 | <1 | 109 | 109 | 0.0 | 14 | 109 | 109 | 0.0 | 525 | 109 | 109 | 0.0 |
| $I_{[\langle 1,10\rangle,\langle 4,10\rangle]}$ | 214 | 1 | 214 | 214 | 0.0 | – | – | – | – | – | – | – | – |
| $I_{[\langle 2,5\rangle,\langle 3,5\rangle]}$ | 205 | 1 | 205 | 205 | 0.0 | 12 | 205 | 205 | 0.0 | – | – | – | – |
| $I_{[\langle 2,10\rangle,\langle 3,10\rangle]}$ | 365 | 17 | 365 | 365 | 0.0 | – | 351 | 375 | 6.8 | – | – | – | – |
| $I_{[\langle 2,5\rangle,\langle 4,5\rangle]}$ | 201 | 1 | 201 | 201 | 0.0 | 64 | 201 | 201 | 0.0 | – | – | – | – |
| $I_{[\langle 2,10\rangle,\langle 4,10\rangle]}$ | 361 | 18 | 361 | 361 | 0.0 | – | 358 | 363 | 1.4 | – | – | – | – |
| $I_{[\langle 3,5\rangle,\langle 4,5\rangle]}$ | 124 | 1 | 124 | 124 | 0.0 | 18 | 124 | 124 | 0.0 | – | – | – | – |
| $I_{[\langle 3,10\rangle,\langle 4,10\rangle]}$ | 244 | 4 | 244 | 244 | 0.0 | – | 209 | 277 | 32.5 | – | – | – | – |
| *TOTALS* | 24 | 24 | 24 | 24 | 0 | 18 | 23 | 23 | 5 | 8 | 10 | 10 | 2 |

- $MILP_{sb}$: $MILP$ plus symmetry breaking.
- $MILP_{d}$: $MILP_{sb}$ using the refinement provided in Section 7.2.

We have observed in our preliminary experiments that a pure MILP model provided an extremely poor performance in the SAMSP. Therefore, in order to confirm that SAMSP is beyond this approach, we use an upper bound highly adjusted to the optimal makespan. In particular, for each instance we consider the value of the known $optimum + 20$, as initial upper bound.

Table 7 reports on the utilization of CPLEX (IBM, 2019) on our problem. Using symmetry breaking allowed the system to solve five more instances than not using it did. It can also be observed that the results for $MILP_{sb}$, corresponding to our adaptation of the classical DDT encoding, are better than those for $MILP_{d}$.

With any of the three encodings considered, the number of solved instances is very small. We have also observed that no encoding is able to solve any instance of Set II.

### 8.4. Comparison experiments

In this subsection we provide a comparison between the best encodings of CP, SMT and MILP. That is:

- $CP_{no}$ for CP, which corresponds to the *Pulse* formulation for the renewable resource constraints.
- $SMT_{amo}$ for SMT, which is the ladder encoding for renewable resource constraints and LIA encoding for cumulative resource constraints.
- $MILP_{sb}$ for MILP, which corresponds to our DDT adaptation.

Table 8 reports on the results obtained. For each approach and each instance we report the following: the time in seconds required to obtain and certify an optimal solution (or – if the optimal solution has not been certified in less than 600 s); the best LB and UB; and the optimality gap. We can see that the CP approach is clearly the best, and that MILP is much worse than SMT and CP. MILP only finds and certifies the optimum for eight instances out of 24, while SMT certifies 18 instances with much better times and CP certifies the optimum of all 24 instances with very little time. In addition, MILP only finds a suboptimal solution for two more instances, and SMT for five instances more. In total, CP finds solution for the 24 instances, SMT for 23 instances and MILP for 10 instances.

We can conclude that the CP approach is clearly better, followed at a certain distance by SMT, while MILP lags far behind them in performance. In the problem studied, all test types have an activity that occupies a position in the observation storage, which is a scarce resource. As a result, the instances tend to have large makespans, which penalizes time-indexed formulations such as the SMT and MILP ones.

### 8.5. Machine design experiments

Table 9 reports on another type of experiment in which we illustrate how the company designing the machine is using our software. What they do is to consider several machine configurations and study what their throughput is for several representative scenarios. Each additional resource has a significant cost and it is mandatory to be able to evaluate whether adding them or not is of interest. We consider the following machine configurations with respect to the number of cold, hot-I, hot-II and observation storage positions: $\langle 4,4,4,2\rangle$, $\langle 4,4,4,4\rangle$ and $\langle 6,6,6,4\rangle$. It can be observed that extending the initial configuration with two more observation storage positions produces a small improvement on the throughput of the machine, i.e., most of the makespans are reduced by 10%. On the other hand, configuration $\langle 6,6,6,4\rangle$ does not provide any benefit regarding makespan.

## 9. Conclusions and further work

We have described a successful application of model-and-solve techniques with which to tackle a real world industrial scheduling problem: the Sample Analysis Machine Scheduling Problem (SAMSP). We have provided CP, SMT and MILP formulations for the problem and shared some preprocessing and symmetry breaking ideas.

The instances obtained from this industrial application have interesting characteristics. First, each activity uses only one renewable resource. Second, the durations of the activities are short compared to the values of the time lags. Third, although there is a large number of activities, these activities are grouped into tests of a particular type, which allow us to apply symmetry breaking techniques based on pre-established execution orders.

**Table 9**
Set I. Comparison of different machine configurations, reporting the optimal makespan with each configuration.

| Instance | ⟨4, 4, 4, 2⟩ | ⟨4, 4, 4, 4⟩ | ⟨6, 6, 6, 4⟩ |
|---|---|---|---|
| $I_{[\langle 1,5\rangle]}$ | 59 | 53 | 53 |
| $I_{[\langle 1,10\rangle]}$ | 102 | 92 | 92 |
| $I_{[\langle 1,20\rangle]}$ | 192 | 182 | 182 |
| $I_{[\langle 2,5\rangle]}$ | 173 | 173 | 173 |
| $I_{[\langle 2,10\rangle]}$ | 298 | 298 | 298 |
| $I_{[\langle 2,20\rangle]}$ | 548 | 548 | 548 |
| $I_{[\langle 3,5\rangle]}$ | 89 | 73 | 73 |
| $I_{[\langle 3,10\rangle]}$ | 145 | 127 | 127 |
| $I_{[\langle 3,20\rangle]}$ | 264 | 247 | 247 |
| $I_{[\langle 4,5\rangle]}$ | 69 | 69 | 69 |
| $I_{[\langle 4,10\rangle]}$ | 124 | 120 | 120 |
| $I_{[\langle 4,20\rangle]}$ | 244 | 240 | 240 |
| $I_{[\langle 1,5\rangle,\langle 2,5\rangle]}$ | 197 | 183 | 183 |
| $I_{[\langle 1,10\rangle,\langle 2,10\rangle]}$ | 357 | 343 | 343 |
| $I_{[\langle 1,5\rangle,\langle 3,5\rangle]}$ | 122 | 107 | 107 |
| $I_{[\langle 1,10\rangle,\langle 3,10\rangle]}$ | 224 | 212 | 212 |
| $I_{[\langle 1,5\rangle,\langle 4,5\rangle]}$ | 109 | 105 | 105 |
| $I_{[\langle 1,10\rangle,\langle 4,10\rangle]}$ | 214 | 210 | 210 |
| $I_{[\langle 2,5\rangle,\langle 3,5\rangle]}$ | 205 | 202 | 202 |
| $I_{[\langle 2,10\rangle,\langle 3,10\rangle]}$ | 365 | 362 | 362 |
| $I_{[\langle 2,5\rangle,\langle 4,5\rangle]}$ | 201 | 197 | 197 |
| $I_{[\langle 2,10\rangle,\langle 4,10\rangle]}$ | 361 | 357 | 357 |
| $I_{[\langle 3,5\rangle,\langle 4,5\rangle]}$ | 124 | 120 | 120 |
| $I_{[\langle 3,10\rangle,\langle 4,10\rangle]}$ | 244 | 240 | 240 |

The CP formulation, using the IBM ILOG CP Optimizer, produced some truly impressive results. In fact, this approach was by far the best, followed by SMT in second place, while the pure MILP approach was a distant third.

Thanks to this study, our partner company has been able to use an exact tool to experimentally evaluate the goodness of different machine configurations, and enhance the machine designing process. We have observed that the real bottleneck component of the machines with the given architecture is the shuttle. Since introducing an extra shuttle is not affordable, the shuttle moving packs of several cuvettes at the same time has been suggested as a way to compensate this. Dealing with such a complex configuration, however, is beyond the scope of this study and will form part of future work, along with incorporating new symmetry breaking techniques for similar test types, and a better heuristic procedure to compute an initial upper bound of the makespan.

## CRediT authorship contribution statement

**Miquel Bofill:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Supervision, Validation, Writing – review & editing. **Jordi Coll:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Gerard Martín:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation. **Josep Suy:** Conceptualization, Formal analysis, Investigation, Methodology, Software, Supervision, Validation, Writing – review & editing. **Mateu Villaret:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Supervision, Validation, Writing – original draft, Writing – review & editing.

## Acknowledgements

This work was supported by MICINN/FEDER, UE (grant number RTI2018-095609-B-I00) and partially funded by the French Agence Nationale de la Recherche, reference ANR-19-CHIA-0013-01.

## References

Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., 2013. A parametric approach for smaller and better encodings of cardinality constraints. In: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming. CP, Springer, pp. 80–96.

Artigues, C., 2017. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. Oper. Res. Lett. 45 (2), 154–159.

Artigues, C., Lopez, P., Haït, A., 2013. The energy scheduling problem: Industrial case-study and constraint propagation techniques. Int. J. Prod. Econ. 143 (1), 13–23.

Bartusch, M., Mohring, R.H., Radermacher, F.J., 1988. Scheduling project networks with resource constraints and time windows. Ann. Oper. Res. 16, 201–240.

Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (Eds.), 2009. Handbook of Satisfiability. In: Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press, p. 980.

Bofill, M., Coll, J., Suy, J., Villaret, M., 2016. Solving the multi-mode resource-constrained project scheduling problem with SMT. In: 28th IEEE International Conference on Tools with Artificial Intelligence. ICTAI, pp. 239–246.

Bofill, M., Coll, J., Suy, J., Villaret, M., 2017. Compact MDDs for pseudo-boolean constraints with at-most-one relations in resource-constrained scheduling problems. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. IJCAI 2017, ijcai.org, pp. 555–562.

Bofill, M., Coll, J., Suy, J., Villaret, M., 2020. SMT encodings for resource-constrained project scheduling problems. Comput. Ind. Eng. 149, 106777. http://dx.doi.org/10.1016/j.cie.2020.106777.

Brucker, P., Drexl, A., Mohring, R., Neumann, K., Pesch, E., 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. Eur. J. Oper. Res. 112 (1), 3–41.

Carlier, J., Moukrim, A., Xu, H., 2009. The project scheduling problem with production and consumption of resources: A list-scheduling based algorithm. Discrete Appl. Math. 157 (17), 3631–3642, Sixth International Conference on Graphs and Optimization 2007.

Chaleshtarti, A.S., Shadrokh, S., 2011. Branch and bound algorithms for resource constrained project scheduling problem subject to cumulative resources. In: 2011 International Conference on Information Management, Innovation Management and Industrial Engineering, Vol. 1. pp. 147–152.

Christofides, N., Alvarez-Valdes, R., Tamarit, J., 1987. Project scheduling with resource constraints: A branch and bound approach. Eur. J. Oper. Res. 29 (3), 262–273.

Dutertre, B., de Moura, L., 2006. The Yices SMT Solver. Tech. Rep., Computer Science Laboratory, SRI International, Available at http://yices.csl.sri.com.

Hartmann, S., Briskorn, D., 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. Eur. J. Oper. Res. 207 (1), 1–14.

IBM, 2019. IBM ILOG CPLEX optimization studio V12.9.0. URL https://www.ibm.com/support/knowledgecenter/es/SSSA5P.

Klein, R., Scholl, A., 1999. Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. Eur. J. Oper. Res. 112 (2), 322–346.

Laborie, P., 2018. An update on the comparison of MIP, CP and hybrid approaches for mixed resource allocation and scheduling. In: van Hoeve, W.J. (Ed.), Integration of Constraint Programming, Artificial Intelligence, and Operations Research. CPAIOR 2018, Delft, the Netherlands, June 26-29, 2018, In: Lecture Notes in Computer Science, vol. 10848, Springer, pp. 403–411.

Laborie, P., Rogerie, J., Shaw, P., Vilím, P., 2018. IBM ILOG CP optimizer for scheduling - 20+ years of scheduling with constraints at IBM/ILOG. Constraints Int. J. 23 (2), 210–250.

Lunardi, W.T., Birgin, E.G., Laborie, P., Ronconi, D.P., Voos, H., 2020. Mixed integer linear programming and constraint programming models for the online printing shop scheduling problem. Comput. Oper. Res. 123, 105020.

Neumann, K., Schwindt, C., Trautmann, N., 2005. Scheduling of continuous and discontinuous material flows with intermediate storage restrictions. Eur. J. Oper. Res. 165 (2), 495–509, Project Management and Scheduling.

Pritsker, A.A.B., Watters, L.J., Wolfe, P.M., 1969. Multiproject scheduling with limited resources: A zero-one programming approach. Manage. Sci. 16 (1), 93–108.

Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G., 2013. Solving RCPSP/max by lazy clause generation. J. Sched. 16 (3), 273–289.

Shin, S.H., Choi, B.J., Ryew, S.M., Kim, J.W., Kim, D.S., Chung, W.K., Choi, H.R., Koo, J.C., 2010. Development of an improved scheduling algorithm for lab test operations on a small-size bio robot platform. JALA: J. Assoc. Lab. Autom. 15 (1), 15–24.

Vilím, P., Laborie, P., Shaw, P., 2015. Failure-directed search for constraint-based scheduling. In: Integration of AI and OR Techniques in Constraint Programming. Springer, pp. 437–453.

You, W.S., Choi, B.J., Moon, H., Koo, J.C., Choi, H.R., 2017. Robotic laboratory automation platform based on mobile agents for clinical chemistry. Intell. Serv. Robot. 10 (4), 347–362.