An SMT Approach to Sample Analysis Machine Scheduling

Abstract. Currently there is a wide variety of automatic machines for performing analytical tests of liquid samples (water, blood, urine, saliva, etc.). These tests consist of activities with precedences, sharing a set of limited resources. Therefore a scheduling process is required. An important particularity of these machines are storage areas. For instance, an activity may move a sample to an observation area and another activity may remove later this sample from the observation area. In this work we present a successful application of optimization techniques to the industry of analytical machines. We introduce a real industrial problem, the Sample Analysis Machine Scheduling Problem (SAMSP) as an instance of the Resource Constrained Project Scheduling Problem with Cumulative Resources (RCPSP-Cu) and provide an efficient SMT formulation for a generic RCPSP-Cu and the particular case of SAMSP. Finally, we evaluate the performance of our system on real instances of the SAMSP.

Keywords: Scheduling · RCPSP · Cumulative Resources · SMT · Sample Analysis

1 Introduction

When designing machines for performing analytical tests, many decisions must be taken: duration of activities, amount of resources available, capacity of the storage areas, etc. Having a software prototype to simulate the throughput of several possible configurations of the machine with several tests scenarios is extremely useful. This information should be at hand of the designers in an early stage of the machine development in order to evaluate the balance between the cost of technological improvement efforts (for instance, improving sample acquisition time by making a faster arm) and the amount of resources available in the machine (for instance having more room in an observation area). Such information will be obtained after a huge amount of simulations and data analysis on the obtained schedules. For instance, some experiments may show that is useless to improve the speed of sample acquisition unless the amount of available room in a heating area is bigger than a threshold. Therefore the simulations need to be efficient and provide a theoretical optimal if possible.

A company has asked us for a software that can help them in the design of a new analytical test machine. Their machines perform several test of distinct types simultaneously. A test consists of several activities that must be performed with limited resources: two robotic arms and a cuvette shuttle. The cuvettes are occupying a position in a certain storage area at all times. This position will change during the execution of a test, since certain tasks can only be performed in particular storage areas. For example, dissolutions of samples can be performed in a cold storage area, whereas a hot area allows for dissolutions as well as incubations. Finally there is a storage area to observe the results. The activities of the tests have to be scheduled minimizing the global makespan.

As said, when designing such machines many alternatives about durations and capacities must be considered. Modelling the problem of obtaining an optimal design for the machine is not a realistic goal since, apart from complexity issues, there is a huge amount of external limitations on the machine design that are not easily known beforehand. Therefore, having a software prototype to simulate the throughput of several possible configurations of the machine is extremely useful and may guide the engineers to obtain an accurate design of the machine. There exist several works in the literature dealing with particular sample analysis machines [14,16], but we haven't found any that could match the requirements of the company.

The well-known Resource Constrained Project Scheduling Problem (RCPSP) consists in 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 minimal distances between activities (minimum time lags). Normally, the goal is to minimize the makespan, i.e., the end time of the schedule. Many generalizations and specializations exist; for a survey on variants and extensions of the RCPSP see [7,11]. In particular, in RCPSP/max [3] 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 the storage facility, or they may manufacture products which are put into the storage facility. A cumulative resource is given by the capacity and the minimum inventory level (or safety stock) of the storage area for this resource. See [12,8,9].

In our problem we do not have productions that should be stored in temporary warehouses, but we do have storage areas with a certain capacity that can be occupied or vacated. Therefore 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. The maximum number of positions in each storage area is limited (storage constraints). If an activity occupies positions it begins to occupy them as soon as it starts, and on the contrary, if it vacates some position, it will do so just after the activity is completed. This is just the opposite to the idea in the RCPSP-Cu. However, although the semantics is different, the way to solve the problem is the same as in the case of the RCPSP-Cu.

In this work we introduce the method we have used for solving our real-world industrial problem, the *Sample Analysis Machine Scheduling Problem (SAMSP)*, as a special case of the RCPSP-Cu. Since the RCPSP-Cu is an extension of the RCPSP/max, and in the design phase of the machine optimality of solutions is required, we are interested in the exact solving approaches for the

RCPSP/max. The most relevant recent exact solving approaches are Constraint Programming [15], Lazy Clause Generation (LCG) [13] and Satisfiability Modulo Theories (SMT) [5,6].

All known exact solvers for the RCPSP-Cu use branch-and-bound or MILP approaches. We believe that an SMT approach can work very well, and for this reason we have decided to use a model-and-solve approach with SMT for solving the SAMSP as a particular case of the RCPSP-Cu.

The rest of the paper is organized as follows. In Section 2 we formally redefine the RCPSP-Cu and propose an SMT encoding for it. In Section 3 we describe our particular real-world industrial case, the SAMSP. In Section 4 we provide several encoding refinements. In Section 5 we provide the results of our experiments comparing different variants of the encoding. We conclude in Section 6 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 performed always at the start of the activity. In our case, an activity occupying positions of the storage facility begins to occupy them as soon as it starts and, on the contrary, if it vacates some position, it will do so when the activity is completed (see Fig. 1).

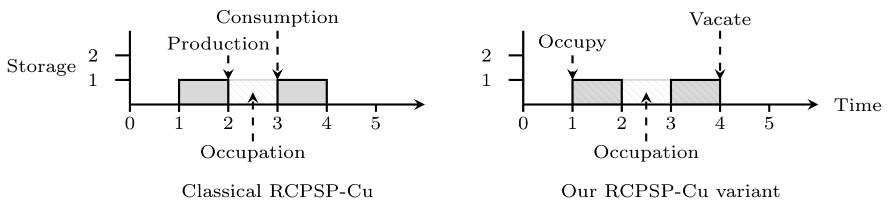

Fig. 1. Difference between classical and our RCPSP-Cu variant.

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

- $\mathcal{V} = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$ 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 $\mathcal{A} = \{A_1, \ldots, A_n\}$.

- d is a vector of $n+2$ naturals, being d_i the duration of activity i, where $d_0 = d_{n+1} = 0$, and $d_i \geq 0\ \forall A_i \in \mathcal{A}$.
- $\mathcal{E}$ 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 start time of activity A_i must precede that of activity A_j in at least $l_{i,j}$ units.[1]
  We assume that we are given a precedence activity-on-node (AON) graph $G(\mathcal{V}, \mathcal{E})$ without cycles, since otherwise the precedence relation would be inconsistent. We also assume that $\mathcal{E}$ is such that A_0 is a predecessor of all other activities and A_{n+1} is a successor of all other activities.
- $\mathcal{R} = \{R_1, \ldots, R_{|\mathcal{R}|}\}$ is a set of renewable resources.
- B is a vector of $|\mathcal{R}|$ naturals, being B_r the available amount of each resource R_r.
- b is a matrix of $(n+2)\times|\mathcal{R}|$ naturals corresponding to the resource demands of activities. Therefore, $b_{i,k}$ represents the amount of resource R_k used during each unit of time of the execution of activity A_i. Note that $b_{0,k} = 0$ and $b_{n+1,k} = 0, \forall k \in \{1, \ldots, |\mathcal{R}|\}$.
- $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$ is a set of cumulative resources (storage areas).
- P is a vector of $|\mathcal{C}|$ naturals, being P_s the available amount of cumulative resource C_s.
- c is a matrix of $(n+2)\times|\mathcal{C}|$ integers corresponding to the amount of positions occupied or vacated by each activity, namely, $c_{i,s} > 0$ represents that activity A_i occupies $c_{i,s}$ positions of storage area C_s and $c_{i,s} < 0$ represents that activity A_i vacates $|c_{i,s}|$ positions of storage area C_s. Any activity occupying positions will begins 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 is completed. Note that $c_{0,s} = 0$ and $c_{n+1,s} = 0, \forall s \in \{1, \ldots, |\mathcal{C}|\}$.

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

(i) for each renewable resource, the amount of that resource required by all he activities running at that time is smaller than the given resource availability.
(ii) for each storage area, its occupancy amount is between zero and its given capacity.

2.1 Encoding

The objective of this problem is to obtain a schedule of the activities of all projects (tests in the particular case of the SAMSP) minimizing the global makespan. We will first describe some preprocessing steps, then define the variables of the model and provide an SMT formulation, and finally describe the optimization process.

[1] Note that while positive $d_{i,j}$ values represent a minimum delay between the start of the two activities, negative values represent a maximum delay.

Preprocessing. We compute the transitive closure $\mathcal{E}^*$ of $\mathcal{E}$ considering only positive lags; in other words, $\mathcal{E}^*$ will contain all triplets $(A_i, A_j, l_{i,j}) \in \mathcal{E}$ where $l_{i,j}$ is negative, as well as triplets $(A_i, A_j, l_{i,j})$ where $l_{i,j}$ is the maximum length of any path from A_i to A_j in $G(\mathcal{V}, \mathcal{E})$ considering only positive lags. We obtain the trivial upper bound

$$UB = \sum_{A_i \in \mathcal{A}} \max(d_i, \max_{(A_i, A_j, l_{i,j}) \in \mathcal{E}} l_{i,j})$$

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

By $TW(A_i)$ be denote the time window of activity A_i, i.e., the range of time instants in which A_i can be running according to some preprocessed information [2]. The overall idea of the encoding is to capture, for each activity $A_i \in \mathcal{V}$, and for each discrete time instant $t \in TW(A_i)$, whether A_i is running at time t. Then, the encoding enforces the constraints on the resources at all time instants $t \in [0 \dots H]$, where H is a large enough time horizon to ensure the feasibility of the projects. From $TW(A_i)$ and d_i we obtain the earliest ES_i and the latest LS_i time instants at which A_i can start, as well as the earliest EC_i and the latest LC_i time instants at which A_i can end.

Variables. 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, $\forall A_i \in \mathcal{V}$.
$X_{i,t}$	integer variables that are 1 if A_i is running at time t, and 0 otherwise. $\forall A_i \in \mathcal{A}, \forall t \in [ES_i \dots LC_i - 1]$.
$Y_{i,t}$	integer variables that are 1 iff A_i has already started at time t, and 0 otherwise, $\forall A_i \in \mathcal{A}, \forall t \in [ES_i \dots H]$.

SMT Encoding. The following set of SMT constraints encode the existence of an schedule within a given time horizon H.

$$S_0 = 0 \tag{1}$$

$$(ES_i \leq S_i) \land (S_i \leq LS_i) \qquad \forall A_i \in \mathcal{V} \tag{2}$$

$$S_j - S_i \geq l_{i,j} \qquad \forall (A_i, A_j, l_{i,j}) \in \mathcal{E}^* \tag{3}$$

$$(0 \leq X_{i,t}) \land (X_{i,t} \leq 1) \qquad \forall A_i \in \mathcal{A}, \forall t \in [ES_i \dots LC_i - 1] \tag{4}$$

$$X_{i,t} = 1 \leftrightarrow ((S_i \leq t) \land (t < S_i + d_i)) \qquad \forall A_i \in \mathcal{A}, \forall t \in [ES_i \dots LC_i - 1] \tag{5}$$

$$(0 \leq Y_{i,t}) \land (Y_{i,t} \leq 1) \qquad \forall A_i \in \mathcal{A}, \forall t \in [ES_i \dots H] \tag{6}$$

$$Y_{i,t} = 1 \leftrightarrow (S_i \leq t) \qquad \forall A_i \in \mathcal{A}, \forall t \in [ES_i \dots LS_i - 1] \tag{7}$$

$$Y_{i,t} = 1 \qquad \forall A_i \in \mathcal{A}, \forall t \in [LS_i, H] \tag{8}$$

$$\left(\sum_{\substack{A_i \in \mathcal{A} \text{ s.t.} \\ t \in [ES_i \dots LC_i], \\ b_{i,r} > 0}} b_{i,r} \cdot X_{i,t} \right) \leq B_r \qquad \forall R_r \in \mathcal{R}, \forall t \in [0 \dots H] \tag{9}$$

$$0 \leq \left(\sum_{\substack{A_i \in \mathcal{A} \text{ s.t.} \\ t \in [ES_i \ldots H], \\ c_{i,s} > 0}} c_{i,s} \cdot Y_{i,t} \right) + \left(\sum_{\substack{A_i \in \mathcal{A} \text{ s.t.} \\ t \in [EC_i \ldots H], \\ c_{i,s} < 0}} c_{i,s} \cdot Y_{i,(t-d_i)} \right) \leq P_s$$

$$\forall C_s \in \mathcal{C}, \forall t \in [0 \ldots H] \quad (10)$$

Constraints (2) restrict the start times of each activity.[2] Constraints (3) are precedence constraints. Constraints (4) and (5) enforce pseudo-Boolean variables $X_{i,t}$ to be 1 if and only if activity A_i is running at time t. Constraints (6), (7) and (8) 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. Constraints (9) and (10) are the 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 $c_{i,s}$) 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 $c_{i,s}$) this also counts for each time instant ranging from the end time of the activity to H.

2.2 Optimization

Optimization is performed by means of successive queries of schedule feasibility using the previous encoding and given a time horizon. In particular, we use a dichotomic 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 to take advantage of the learning between successive calls.

3 The Sample Analysis Machine Scheduling Problem

The industrial problem that we consider consists in scheduling a set of tests in a sample analysis machine, minimizing the total 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 in a particular collection of activities with their corresponding precedences, durations, and renewable and cumulative resource demands. Tests may follow the following steps:

(i) A test recipient named cuvette is acquired from a cuvette pool and moved to an storage facility.
(ii) A sample from the sample pool is aspirated and disposed into that cuvette.
(iii) Once the sample is in the cuvette, many diluents from a diluents pool can be added.

[2] Note that this same constraints asks also S_{n+1} to be smaller than LS_{n+1} that is in fact H.

(iv) Previous actions can be performed both in cold or in hot storage areas. If the cuvette is in a hot storage area then incubation starts, otherwise it will be moved to the hot storage area to start incubation.
(v) During incubation, many reagents from a reagent pool can be added to the sample.
(vi) The recipient is moved to an observation storage area, where other reagents can also be added, and the observation process starts. This observation process produces the test outcome.
(vii) Finally, the cuvette is trashed.

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, cuvettes transport between storage areas, diluents and reagents addition, . . .) that the machine must perform to accomplish a test of that type. There is a generalized precedence relation between activities of a test type: minimal and maximal 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, . . .) and a number of occupied or vacated positions in storage areas (cold, hot, . . .).[3]

Then, the *Sample Analysis Machine Scheduling Problem (SAMSP)* consists in, given a sample analysis machine specification, a set of test type specifications for that machine, and a number of tests of each test type to be performed, generate a schedule of the activities that minimizes the completion time and fulfils all precedence, renewable resource and storage constraints.

This problem is similar to the ones considered in [14,16] but under a different machine architecture.

3.1 The Sample Analysis Machine Architecture

Our sample analysis machine is structured in 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 to transport cuvettes between the different storage areas allocated in the machine, i.e., it connects the two machine zones (see Figure 2).

The sample zone is composed by two cuvette storage areas, named *Cold* and *Hot-I* storage, and three pools: cuvettes pool, samples pool and diluents pool.

The reagent zone is also composed by two cuvette storage areas, named *Hot-II* and *Observation* storage, plus a reagents pool and a trash.

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

[3] Notice that a sample analysis machine specification with a test type specification is an instance of an RCPSP-cu.

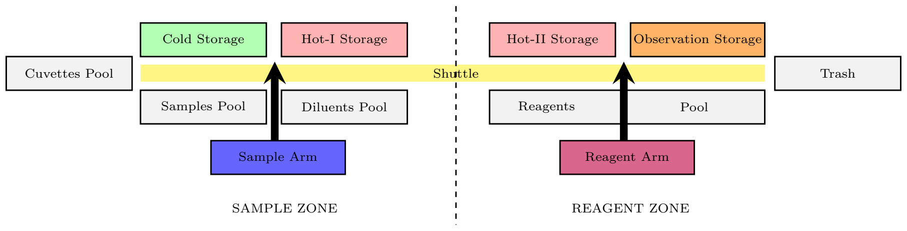

Fig. 2. Schematic Machine Architecture.

– Shuttle movements:
  - Move a cuvette from the cuvettes pool to any cuvette storage area.
  - Move a cuvette between cuvette storage areas.
  - Move a cuvette a from cuvette storage area to the trash.
– Sample arm operations:
  - Aspirate a sample and dispense it to a cuvette located in Cold or Hot-I storage areas.
  - Aspirate a diluent and dispense it to a cuvette located in Cold or Hot-I storage areas.
  - Aspirate from a cuvette located in 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 Hot-II or Observation storage areas.

The activities of the tests will correspond to the operations described above with their appropriate durations, precedences, renewable resource (shuttle and arms) demands and storage demands and supplies.

A test type example in our machine architecture. A test type example for our machine is depicted in Figure 3. The meaning of every activity is the following:

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 Trash 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 Storage Area by the shuttle.
12. Cuvette 1 is transported from the Observation Storage Area to the Trash by the shuttle.

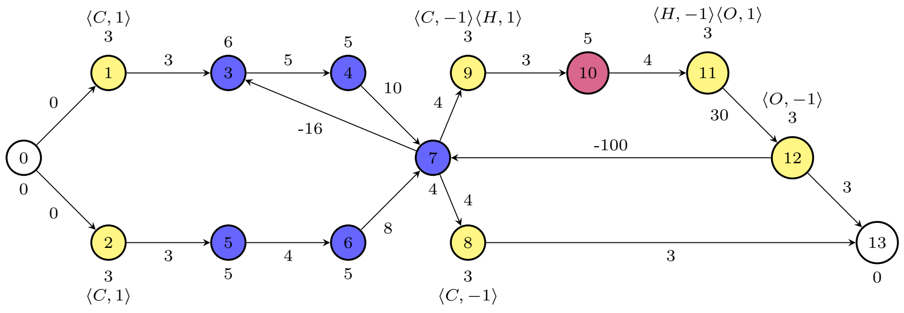

Fig. 3. Graph of precedences of a Test Type. Each activity is labelled by its duration and the number of positions occupied/vacated in the corresponding storage area (C:Cold; H:Hot-II; O:Observation) if the activity has some. Shuttle actions are depicted in yellow, sample arm actions in blue and reagent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

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

To turn a SAMSP instance into an RCPSP-Cu instance we need to obtain the global set of precedence relations $\mathcal{E}_g$ by joining 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}, being n the number of non-dummy activities of all joined tests. We denote by $\mathcal{V}_g$ the resulting set with all activities. These activities have their corresponding durations d_g, resource demands b_g and occupancy/vacation amounts c_g. The set of non-dummy activities is then defined as $\mathcal{A}_g = \mathcal{V}_g \setminus \{A_0, A_{n+1}\}$. Finally, we denote by $T_g = (\mathcal{V}_g, d_g, \mathcal{E}_g, \mathcal{R}, B, b_g, \mathcal{C}, P, c_g)$ the RCPSP-Cu instance resulting of the union of all the tests.

Due to the large number of activities and the great level of parallelism that can be achieved it is very important to find a good Upper Bound. To do this we check the feasibility of the problem by setting as Upper Bound the trivial Lower Bound. If we get a positive answer then we have an Upper Bound (and hence an optimal solution because the upper bound would be the same as the lower bound). Otherwise we double the value of this Lower Bound and repeat the process until we get a positive answer and hence an Upper Bound. We call this process inverse dichotomic UB search. We also calculate the transitive closure $\mathcal{E}_g^*$ of $\mathcal{E}^*$ as described in Section 2.1, and use the optimization process described in Section 2.2.

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

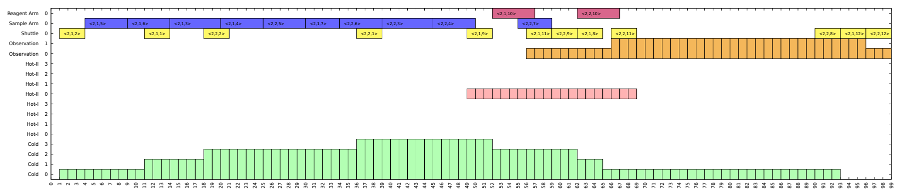

Fig. 4. Solution for two instances of the test type depicted in Figure 3. In each activity the triplet $\langle tt, itt, na\rangle$ represents: tt the test type, itt the instance number of the test and na the activity number in the test type.

4 Encoding Refinements

We have made the following refinements to the encoding of Section 2.1:

1. **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 (11) and (13) in $\mathcal{E}_g$ (see below) before computing the extended precedence graph $\mathcal{E}_g^*$. Recall that inequation (3) transforms each precedence relation from $\mathcal{E}_g^*$ into a precedence constraint. These additional symmetry breaking precedence constraints allow to significantly reduce activity time windows.
   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 $A_i \in \mathcal{A}_g$, 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.

$$(A_i, A_j, o) \qquad \begin{array}{c} \forall A_i, A_j \in \mathcal{A}_g \text{ s.t.} \\ tt_i = tt_j, na_i = na_j, itt_i = itt_j - 1 \end{array} \tag{11}$$

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 and $o = d_i$ 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 minimal amount of time that, for sure, a certain storage area will be occupied by such activities (12). These time lags will be computed considering only test types of equal type T_k. Using (13) we compute a set of new precedences that will replace the existing ones in $\mathcal{E}_g^*$ when they are longer.

$$pt_{i,s,k} = \min\Big(\infty, \min_{\substack{\forall A_j \in \mathcal{A}_k, s.t. \\ c_{i,s} > 0, c_{j,s} < 0, \\ (A_i, A_j, l_{i,j}) \in \mathcal{E}_k^*, l_{i,j} > 0}} l_{i,j} + d_j\Big) \qquad \forall A_i \in \mathcal{A}_k, \forall C_s \in \mathcal{C}, \forall T_k \tag{12}$$

$$(A_i, A_j, pt_{na_i,s,k}) \qquad \begin{array}{c} \forall A_i, A_j \in \mathcal{A}_g, \forall C_s \in \mathcal{C}, \forall T_k \text{ s.t.} \\ pt_{na_i,s,k} > 0, pt_{na_i,s,k} < \infty, \\ tt_i = tt_j = k, na_i = na_j, itt_i = itt_j - \lfloor \frac{P_s}{c_{i,s}} \rfloor \end{array} \tag{13}$$

where $\sigma(A_i) = \langle tt_i, itt_i, na_i \rangle$, $\sigma(A_j) = \langle tt_j, itt_j, na_j \rangle$. In (13) 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 storage area C_s capacity and storage area demand of A_i is $\lfloor \frac{P_s}{c_{i,s}} \rfloor$.

For example, if the size of an storage area C_s is $P_s = 2$ and the demand of positions of an activity A_i is $c_{i,s} = 1$, 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, ...

2. **Resource Constraints**. In the machine architecture considered all non-cumulative resources are unary ($B_r = 1$) and the demands over them are also unary ($b_{i,r} = 1$). Therefore, we can replace resource constraints (9) in the basic encoding of Section 2.1 in several alternative ways:

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

$$(S_j \geq S_i + d_i) \vee (S_i \geq S_j + d_j) \qquad \begin{array}{c} \forall A_i, A_j \in \mathcal{A}_g, i < j \text{ s.t.} \\ [ES_i \dots LC_i] \cap [ES_j \dots LC_j] \neq \emptyset, \\ \exists R_r \in \mathcal{R}, b_{i,r} > 0, b_{j,r} > 0 \end{array} \tag{14}$$

  In this case, it is not necessary to introduce variables $X_{i,t}$ neither constraints (4) and(5).
- 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, $\forall A_i \in \mathcal{A}, \forall t \in [ES_i \dots LC_i - 1]$, i.e., replacing constraints(4) and(5) by:

$$x_{i,t} \leftrightarrow ((S_i \leq t) \wedge (t < S_i + d_i)) \quad \forall A_i \in \mathcal{A}, \forall t \in [ES_i \dots LC_i - 1] \tag{15}$$

  Then (9) can be replaced by a SAT encoding of an at-most-one (AMO) constraint:

$$AMO_{\substack{A_i \in \mathcal{A}_g \text{ s.t.} \\ t \in [ES_i \dots LC_i], \\ b_{i,r} > 0}} (x_{i,t}) \qquad \forall R_r \in \mathcal{R}, \forall t \in [0, H] \tag{16}$$

  In particular we use two encodings for constraints(16): regular/ladder encoding (see[4]), and the pairwise encoding, which introduces the follow clauses:

$$\overline{x_{i,t}} \vee \overline{x_{j,t}} \qquad \begin{array}{c} \forall A_i, A_j \in \mathcal{A}_g, i < j, \\ \forall t \in [ES_i \dots LC_i] \cap [ES_j \dots LC_j] \text{ s.t.} \\ \exists R_r \in \mathcal{R}, b_{i,r} > 0, b_{j,r} > 0 \end{array} \tag{17}$$

Notice that if $(A_i, A_j, l_{i,j}) \in \mathcal{E}_g^*$ and $l_{i,j} > 0$ or $(A_j, A_i, l_{j,i}) \in \mathcal{E}_g^*$ and $l_{j,i} > 0$ then:
- If $l_{i,j} > d_i$ or $l_{j,i} > d_j$ it is impossible that these two activities are simultaneously executed, therefore constraints (14) and the pairwise encoding (17) do not introduce clauses for such pair of activities.
- Otherwise, there exists a relative order between these activities (w.l.o.g. $d_i \geq l_{i,j} > 0$) and only one part of the disjunction of (14) is needed (i.e. $(S_j \geq S_i + d_i)$).

3. **Cumulative Constraints**. Similarly to non-cumulative 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, $\forall A_i \in \mathcal{A}, \forall t \in [ES_i \dots H]$, thus replacing constraints (6), (7) and(8) by the following:

$$y_{i,t} \leftrightarrow (S_i \leq t) \qquad \forall A_i \in \mathcal{A}, \forall t \in [ES_i \dots LS_i - 1] \tag{18}$$

$$y_{i,t} \qquad \forall A_i \in \mathcal{A}, \forall t \in [LS_i \dots H] \tag{19}$$

Using Boolean variables $y_{i,t}$ and the SAT encoding of cardinality constraints presented in [1], we can replace constraints (10) by:

$$0 \leq \left(\sum_{\substack{A_i \in \mathcal{A} \text{ s.t.} \\ t \in [ES_i \ldots H], \\ c_{i,s} > 0}} y_{i,t} \right) + \left(\sum_{\substack{A_i \in \mathcal{A} \text{ s.t.} \\ t \in [EC_i \ldots H], \\ c_{i,s} < 0}} (\overline{y_{i,(t-d_i)}} - 1) \right) \leq P_s \qquad \forall C_s \in \mathcal{C}, \forall t \in [0 \ldots H] \quad (20)$$

4. **Redundant Channelling Constraints**. Finally, we can also add the following redundant constraint to make explicit to the SAT solver the channelling between variables $x_{i,t}$ (or $X_{i,t}$) and $y_{i,t}$ (or $Y_{i,t}$):

$$\overline{x_{i,t}} \vee y_{i,t} \qquad \forall A_i \in \mathcal{A}_g, t \in [ES_i \ldots LC_i] \quad (21)$$

We must substitute in (21) the terms $x_{i,t}$ (similarly $y_{i,t}$) for $X_{i,t} = 1$ (similarly $Y_{i,t} = 1$) when we use the integer variables instead of Boolean variables.

5 Experiments

We have run our experiments on a 8GB Intel® Xeon® E3-1220v2 machine at 3.10 GHz. In all experiments we have used Yices 2.6.1 [10] as the core SMT solver.[4] There are four test types. We have set the nomenclature for the set of instances as I_L, where $L = [\langle i, j \rangle]$ is a list of pairs where i is a test type identifier and j is the number of required tests of test type i. We have considered the following incremental combinations of encodings:

- E: the basic encoding provided in Section 2.1.
- E_{sb1}: E with the resource symmetry breaking derived from (11).
- E_{sb2}: E with the storage symmetry breaking derived from (13).
- E_{sb}: E with both kinds of symmetry breaking.
- E_{amo}: E_{sb} with AMO constraints for resource constraints (16) with a regular/ladder encoding.
- E_b: E_{sb} with a Boolean pairwise encoding (17) of resource constraints.
- E_{mu}: E_{sb} with mutexes for resources constraints (14).
- $E_{b,card}$: E_b with cardinality constraints for storage constraints (20).
- $E_{b,red}$: E_b with redundant channeling constraints, i.e., with constraints (21) being $\overline{x_{i,t}} \vee (Y_{i,t} = 1)$.

5.1 Results

Table 1 shows that the basic encoding E struggles to solve instances with only 10 tests. In fact, it only solves 4 instances out of 24. With respect to symmetry

[4] Our solvers, benchmarks and detailed results will be make publicly available.

Table 1. Performance comparison of encoding variants. Time in seconds (time limit 600).

Instance	Opt	E	Symmetry			Resources			$E_{b,card}$	$E_{b,red}$
			E_{sb1}	E_{sb2}	E_{sb}	E_{amo}	E_b	E_{mu}		
$I_{[\langle 1,5\rangle]}$	59	1	1	1	1	1	1	1	1	1
$I_{[\langle 1,10\rangle]}$	102	-	1	1	1	2	1	1	2	1
$I_{[\langle 1,20\rangle]}$	192	-	18	21	11	16	6	6	16	1
$I_{[\langle 2,5\rangle]}$	173	28	9	12	8	12	2	6	13	2
$I_{[\langle 2,10\rangle]}$	298	-	473	571	227	106	98	212	106	16
$I_{[\langle 2,20\rangle]}$	548	-	-	-	-	-	-	-	-	218
$I_{[\langle 3,5\rangle]}$	89	1	1	1	1	2	1	1	1	1
$I_{[\langle 3,10\rangle]}$	145	-	6	11	5	10	1	6	12	1
$I_{[\langle 3,20\rangle]}$	264	-	326	-	217	224	175	461	253	68
$I_{[\langle 4,5\rangle]}$	69	1	1	1	1	1	1	1	1	1
$I_{[\langle 4,10\rangle]}$	124	-	11	597	6	9	5	13	13	2
$I_{[\langle 4,20\rangle]}$	244	-	296	-	278	319	166	-	223	180
$I_{[\langle 1,5\rangle,\langle 2,5\rangle]}$	197	-	54	363	43	31	20	230	35	8
$I_{[\langle 1,10\rangle,\langle 2,10\rangle]}$	-	-	-	-	-	-	-	-	-	-
$I_{[\langle 1,5\rangle,\langle 3,5\rangle]}$	122	-	7	21	7	10	5	11	9	2
$I_{[\langle 1,10\rangle,\langle 3,10\rangle]}$	-	-	-	-	-	-	-	-	-	-
$I_{[\langle 1,5\rangle,\langle 4,5\rangle]}$	109	-	48	-	69	50	41	245	57	41
$I_{[\langle 1,10\rangle,\langle 4,10\rangle]}$	-	-	-	-	-	-	-	-	-	-
$I_{[\langle 2,5\rangle,\langle 3,5\rangle]}$	205	-	82	-	73	53	33	355	45	13
$I_{[\langle 2,10\rangle,\langle 3,10\rangle]}$	-	-	-	-	-	-	-	-	-	-
$I_{[\langle 2,5\rangle,\langle 4,5\rangle]}$	201	-	473	-	96	58	29	310	59	10
$I_{[\langle 2,10\rangle,\langle 4,10\rangle]}$	361	-	-	-	-	-	-	-	-	487
$I_{[\langle 3,5\rangle,\langle 4,5\rangle]}$	124	-	47	-	75	152	53	261	68	58
$I_{[\langle 3,10\rangle,\langle 4,10\rangle]}$	-	-	-	-	-	-	-	-	-	-
$TOTAL$	24	4	17	11	17	17	17	16	17	19

breaking we can observe that both E_{sb1} and E_{sb2} improve the results substantially, and that their union, i.e. E_{sb}, achieves better results: a total of 17 instances are solved with much better times. Symmetry breaking can only be applied if there are several tests of the same test type in the test set to schedule. Fortunately, this is the case in most of the test sets. Usually there are a few test types that are required to be performed on most of the samples.

Regarding the variants of the constraints on renewable resources (9), the best one is the Boolean pairwise encoding E_b. On the other hand, for the constraints on cumulative resources the best variant is the initial one (10). Transforming this constraint to a cardinality constraint (20) does not provide any improvement.

Finally, we can observe that the redundant channelling constraint (21) really improves the performance, allowing to solve two more instances than without it. Among all, the $E_{b,red}$ encoding variant is the best one.

Table 2 reports on another type of experiments. In these experiments we illustrate how the company designing the machine is using our software. What they do is to consider several machine configurations and study which is their throughput for several representative scenarios. Each additional resource has a significant cost and it is mandatory to be able to evaluate the interest of adding them or not. In the experiments we use the encoding variant $E_{b,red}$ since it is the one that has reported the best results, as shown in Table 1. 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 $\langle 6,6,6,4\rangle$, respectively. The first configuration is, in fact, the one used in Table 1. It can be observed that extending this configuration with two more Observation storage positions produces a little improvement on the throughput of the machine, i.e., most of the makespans are reduced by a 10%. On the other hand, configuration $\langle 6, 6, 6, 4\rangle$ does not provide any benefit regarding makespan; moreover, it takes much longer to solve most of the instances, probably due to the increase in the encoding size.

Table 2. Comparison of different machine configurations. Time in seconds (time limit 600).

Instance	$\langle 4, 4, 4, 2\rangle$		$\langle 4, 4, 4, 4\rangle$		$\langle 6, 6, 6, 4\rangle$	
	Opt.	Time	Opt.	Time	Opt.	Time
$I_{[\langle 1,5\rangle]}$	59	1	53	1	53	1
$I_{[\langle 1,10\rangle]}$	102	1	92	1	92	1
$I_{[\langle 1,20\rangle]}$	192	1	182	7	182	12
$I_{[\langle 2,5\rangle]}$	173	2	173	2	173	2
$I_{[\langle 2,10\rangle]}$	298	16	298	20	298	26
$I_{[\langle 2,20\rangle]}$	548	218	548	243	548	359
$I_{[\langle 3,5\rangle]}$	89	1	73	1	73	1
$I_{[\langle 3,10\rangle]}$	145	1	127	3	127	3
$I_{[\langle 3,20\rangle]}$	264	68	-	-	-	-
$I_{[\langle 4,5\rangle]}$	69	1	69	1	69	1
$I_{[\langle 4,10\rangle]}$	124	2	120	3	120	4
$I_{[\langle 4,20\rangle]}$	244	180	-	-	-	-

Instance	$\langle 4, 4, 4, 2\rangle$		$\langle 4, 4, 4, 4\rangle$		$\langle 6, 6, 6, 4\rangle$	
	Opt.	Time	Opt.	Time	Opt.	Time
$I_{[\langle 1,5\rangle,\langle 2,5\rangle]}$	197	8	183	6	183	6
$I_{[\langle 1,10\rangle,\langle 2,10\rangle]}$	-	-	-	-	-	-
$I_{[\langle 1,5\rangle,\langle 3,5\rangle]}$	122	2	107	90	107	92
$I_{[\langle 1,10\rangle,\langle 3,10\rangle]}$	-	-	-	-	-	-
$I_{[\langle 1,5\rangle,\langle 4,5\rangle]}$	109	41	105	65	105	62
$I_{[\langle 1,10\rangle,\langle 4,10\rangle]}$	-	-	-	-	-	-
$I_{[\langle 2,5\rangle,\langle 3,5\rangle]}$	205	13	202	11	202	13
$I_{[\langle 2,10\rangle,\langle 3,10\rangle]}$	-	-	-	-	-	-
$I_{[\langle 2,5\rangle,\langle 4,5\rangle]}$	201	10	197	10	197	16
$I_{[\langle 2,10\rangle,\langle 4,10\rangle]}$	361	487	357	484	-	-
$I_{[\langle 3,5\rangle,\langle 4,5\rangle]}$	124	58	120	383	120	486
$I_{[\langle 3,10\rangle,\langle 4,10\rangle]}$	-	-	-	-	-	-

6 Conclusions and Further Work

We have reported a successful application of an SMT model-and-solve technique to tackle a real-world industrial scheduling problem: the Sample Analysis Machine Scheduling Problem (SAMSP). After providing a first encoding for this problem, we have proposed different refinements such as improved precedence constraints, modelling alternatives for resource and storage constraints, symmetry breaking constraints, and redundant channelling constraints. We have seen that the best alternative is to use pure Boolean encodings for AMO-like resource constraints, linear integer arithmetic expressions for storage constraints, and to apply symmetry breaking and redundant channelling constraints.

Thanks to the system that we have present presented, 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, it has been suggested the possibility of the shuttle moving packs of several cuvettes at the same time. Dealing with such complex configuration is left as further work. We also left as further work the incorporation of further symmetry breaking techniques for similar test types, and a better heuristic procedure to compute an initial upper bound of the makespan.

References

1. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: 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). pp. 80–96. Springer (2013)
2. Artigues, C., Demassey, S., Neron, E.: Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. ISTE Ltd. (2007)
3. Bartusch, M., Mohring, R.H., Radermacher, F.J.: Scheduling Project Networks with Resource Constraints and Time Windows. Annals of Operations Research **16**, 201–240 (January 1988)
4. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (February 2009)
5. Bofill, M., Coll, J., Suy, J., Villaret, M.: 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 (2016)
6. Bofill, M., Coll, J., Suy, J., Villaret, M.: 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. pp. 555–562. ijcai.org (2017)
7. Brucker, P., Drexl, A., Mohring, R., Neumann, K., Pesch, E.: Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. European Journal of Operational Research **112**(1), 3 – 41 (1999)
8. Carlier, J., Moukrim, A., Xu, H.: The project scheduling problem with production and consumption of resources: A list-scheduling based algorithm. Discrete Applied Mathematics **157**(17), 3631 – 3642 (2009), sixth International Conference on Graphs and Optimization 2007
9. Chaleshtarti, A.S., Shadrokh, S.: 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 (2011)
10. Dutertre, B., de Moura, L.: The Yices SMT Solver. Tech. rep., Computer Science Laboratory, SRI International (2006), available at `http://yices.csl.sri.com`
11. Hartmann, S., Briskorn, D.: A Survey of Variants and Extensions of the Resource-Constrained Project Scheduling Problem. European Journal of Operational Research **207**(1), 1 – 14 (2010)
12. Neumann, K., Schwindt, C., Trautmann, N.: Scheduling of continuous and discontinuous material flows with intermediate storage restrictions. European Journal of Operational Research **165**(2), 495 – 509 (2005), project Management and Scheduling
13. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Solving rcpsp/max by lazy clause generation. J. Scheduling **16**(3), 273–289 (2013)
14. Shin, S.H., Choi, B.J., Ryew, S.M., Kim, J.W., Kim, D.S., Chung, W.K., Choi, H.R., Koo, J.C.: Development of an improved scheduling algorithm for lab test operations on a small-size bio robot platform. JALA: Journal of the Association for Laboratory Automation **15**(1), 15–24 (2010)
15. Vilím, P., Laborie, P., Shaw, P.: Failure-directed search for constraint-based scheduling. In: Integration of AI and OR Techniques in Constraint Programming, pp. 437–453. Springer (2015)

16. You, W.S., Choi, B.J., Moon, H., Koo, J.C., Choi, H.R.: Robotic laboratory automation platform based on mobile agents for clinical chemistry. Intelligent Service Robotics **10**(4), 347–362 (Oct 2017)