Addressing storage time restrictions in the S-graph scheduling framework

Máté Hegyháti¹ · Tibor Holczinger² · Olivér Ósz¹

Received: 19 December 2019 / Revised: 24 July 2020 / Accepted: 24 July 2020 /
Published online: 4 August 2020
© The Author(s) 2020

Abstract

Storage planning and utilization are among the most important considerations of practical batch process scheduling. Modeling the available storage options appropriately can be crucial in order to find practically applicable solutions with the best objective value. In general, there are two main limitations on storage: capacity and time. This paper focuses on the latter and investigates different techniques to tackle limited storage time within the S-graph framework. The S-graph framework is a collection of combinatorial algorithms and a directed graph based model that has been introduced three decades ago and has been under development ever since. In this work, several options for addressing storage time limitations within the framework were implemented and tested for efficiency. The empirical results over a huge number of tests have unequivocally favored one of the approaches, which will be applied in later developments.

Keywords Scheduling · Limited-wait storage policy · S-graph framework

1 Introduction and literature review

In real life scheduling problems different rules may be present regarding the storage of intermediate materials between consecutive tasks. Constraints can be concerned with the capacity and the time limit of the storage operation. For capacity, we can

✉ Tibor Holczinger
holczinger.tibor@uni-pen.hu

Máté Hegyháti
hegyhati@sze.hu

Olivér Ósz
osz.oliver@sze.hu

¹ Department of Information Technology, Széchenyi István University, Egyetem tér 1., Győr 9026, Hungary

² Department of Applied Informatics, University of Pannonia, Zrínyi Miklós u. 18., Nagykanizsa 8800, Hungary

distinguish three main policies, unlimited- (UIS), finite- (FIS) and no intermediate storage (NIS) policy. Similarly, there are three common policies for limiting the time the intermediate material spends in storage, unlimited- (UW), limited- (LW) and zero-wait (ZW) policy.

Limited-wait storage policy often occurs in industry because of some time-sensitive physical or chemical properties of materials, e.g. temperature. This duration is usually specific to the material, i.e., different materials in the facility may have different limits. From the mathematical point of view, the starting time of the second task cannot be bigger than the completion time of the first task plus the allowed time limit. The ZW policy (or no-wait policy) is a specific case of the limited-wait strategy where the time limit is zero. In this case, the intermediates have to be processed in the next task without any delay. The completion time of the first task must be equal to the starting time of the second task. In the case of UW policy, there is no upper bound on the starting time of the receiving task for an intermediate posed by the completion time of the producing one.

Similarly to storage time limits, the storage capacity in the facility may differ for each intermediate material in the production. Note that if batch sizes are fixed a priori, and each intermediate is generated only once in the production, FIS policy reverts to either UIS or NIS based on the capacity of the storage. In some situations materials may also share the same storage unit (CIS, common intermediate storage policy). This option often raises additional challenges in planning such as appropriate cleaning.

In this paper we focus on the combination of NIS/UIS and UW/LW storage policies.

The first scheduling problems with no-wait constraints were published in the 70's, for example, Callahan (1971) presented a steel industry problem, and Salvador (1973) described an algorithm for a nylon polymerization flow. A lot of research papers have been published since then. Hall and Sriskandarajah (1996) and Allahverdi (2016) provided review papers about shop scheduling problems (flow shop, job shop, open shop) with no-wait constraints. These two reviews covered more than 400 publications which appeared between 1970 and 2016.

While shop scheduling and various aspects of the planning of batch processes have already been investigated for a couple of decades, the scheduling of industrial batch processes gained heightened interest in the 90's among engineers and optimization experts. Since then, researchers have presented numerous methods to solve problems (Méndez et al. 2006; Hegyháti and Friedler 2010; Allahverdi et al. 2018). Most papers proposed a Mixed Integer Linear Programming (MILP) model of the scheduling problem and solved it with a general-purpose solver like CPLEX. The main advantages of these approaches are their flexibility and extensibility but they also have disadvantages. Shaik and Floudas (2009) showed that the developed approaches may lead to suboptimal solutions, moreover, Ferrer-Nadal et al. (2008) and Hegyháti et al. (2009) reported that even infeasible schedules can be reported as optimal. Another group of solution methods is based on directed graphs and specialized branch-and-bound algorithms. The first graph-based approach was presented by Balas (1969) who showed a backtracking algorithm using disjunctive graphs. Sanmartí et al. (2002) published the S-graph framework and D'Ariano et al. (2007) showed a similar alternative graph

and mathematical methodology. A third notable direction in batch process scheduling is based on the state space enumeration of the production facility by either Linear Priced Timed Automata (Schoppmeyer et al. 2012) or Timed Place Petri Nets (Ghaeli et al. 2005). Using metaheuristics to find a good solution in a reliable time is a common approach nowadays because of the NP-hard nature of batch process scheduling. Some notable contributions of this sort involve genetic programming (Nguyen et al. 2019; Park et al. 2018), set partitioning (Grenouilleau et al. 2019) or ant colony optimization (Zhang and Wong 2018).

Lots of batch plants do not allow intermediate storage, nevertheless, problems with NIS and mixed wait policies have received less attention than those with UIS/UW. There are some papers containing MILP models (Jung et al. 1994; Liu and Karimi 2008; Gicquel et al. 2012) but most of the publications use heuristics to solve problems featuring ZW or LW policies. Moon et al. (1996) presented a MILP model for multiproduct batch plants with ZW policy using a heuristic preprocessing step to solve it. Condotta and Shakhlevich (2012) proposed a tabu search based algorithm to solve two-stage job shop problems where a single machine is available for each job and has a certain time interval between the two stages. An et al. (2016) examined a two-machine flow shop scheduling problem with LW policy and developed a B&B algorithm which uses heuristics to accelerate the search. Zhou et al. (2018) showed a swarm optimization based algorithm to solve a battery production plant where there is a LW policy before the formation process.

Limited-wait storage policy often appears in case studies of specific problem classes, which made it necessary to extend the capabilities of the S-graph framework in this direction, too. The first proposed method relied on arcs with interval weights (Hegyháti et al. 2011) and a modified feasibility function. There are, however, several other ways to address this feature, which were also briefly investigated (Hegyháti 2015).

The goal of this paper is twofold. First, it aims to provide a comprehensive review and description of approaches that are available to tackle this task, either published or unpublished. Secondly, it provides the results of an extensive empirical comparison to reveal the technique that proves to be the most efficient in practice. The approaches are expatiated on scheduling problems whose production constraints adhere to Flexible Job Shop rules, which are often referred to as Multipurpose Recipes in the chemical batch process scheduling literature. The objective is the minimization of makespan, processing steps are non-preemptive, and any combination of UIS/NIS and UW/LW policies may apply to each intermediate material. The empirical tests were carried out on this specific class of scheduling problems, however, it has to be noted, that the presented approaches are not limited by these constraints, and they may be applied for much wider classes of scheduling problems, that the S-graph framework can address otherwise.

2 Motivational example

Limitations on the storage time can have significant effect on the objective function or, in some cases, even feasibility. In this section, a simple motivational example is presented to highlight this effect. The recipe for the example is shown

Fig. 1 Motivational example

Fig. 2 Optimal schedule for the motivational example in case of UW policy

in Fig. 1 in the form of a block diagram. The problem features three products, A, B, and C, each having a sequential production recipe with 3, 2, and 1 production steps, respectively. These steps can be carried out with 3 units and only a single unit is suitable for any task. It is also assumed that each intermediate can have dedicated storage units with sufficiently large capacity as the goal of this example is to focus on time limitations only.

From the optimization point of view, the most relaxed case for storage time is the UW policy, thus, the optimal UW makespan provides a lower bound on the makespan for cases with more restricting policies. The UW optimal schedule is shown in Fig. 2 with the makespan of 13 h.

This schedule requires the intermediates in the production of A to be stored for 2 and 3 h, so this schedule is definitely not feasible if the storage time for all intermediates is limited to 2 h. Another schedule with different sequencing decision might still provide the same 13 h optimum but for this problem that is not the case. In the LW optimal schedule the sequencing decisions remain the same and makespan is increased to 14 h by delaying the tasks $i1$, $i2$ and $i6$ by 1 h, as shown in Fig. 3.

As UW is the most relaxed policy, ZW is the other end of the spectrum with being the most restrictive policy on storage timing. This also means that the optimal makespan with ZW policy provides an upper bound on the optimal makespan of 17 h as shown in Fig. 4. Note, that in this case, the sequencing decisions are also altered as the sequencing of the previous cases would result in a worse makespan of 18 h with ZW policy.

Fig. 3 Optimal schedule for the motivational example in case of LW policy with 2 h of maximal storage time

Fig. 4 Optimal schedule for the motivational example in case of ZW policy

This small example showed that time limitation on storage operations can affect the objective. As it will be shown in Sect. 6, it can also influence the computational needs of approaches to find the optimal solution, which was observed for MILP formulations for the cyclic scheduling of robotic cells (Papp et al. 2018).

3 Problem definition

Limited-wait storage policy can be present in a scheduling problem regardless of many other features of the problem, e.g., the objective function. In this paper, the proposed approaches are introduced (Sect. 5) and compared (Sect. 6) via makespan minimization problems. As the methods of the S-graph framework for maximizing throughput (Majozzi and Friedler 2006; Holczinger et al. 2007) and other objectives (Holczinger et al. 2012, 2019) rely on makespan minimization as a subroutine, the approaches can be used in those methods without any or just minor alterations. The aim of this paper is to illustrate the approaches in their simplest form, i.e., any parameters which are *independent* from the limit on the storage time or constraint are disregarded. The inclusion of such aspects would only make the presentation in Sect. 5 more obscure without adding any scientific value.

In the investigated scheduling problem class the makespan has to be minimized for a given product quantity with the available equipment units of the facility. All of these units are assumed to be operated in batch mode and preemption is not allowed

during production steps. The required production quantity is given as a number of batches for each product in consideration, i.e., batch sizes of each product are fixed a-priori. The processing steps of a product can have any non-cyclic dependency network and they are referred to as tasks in the rest of the paper. These networks are assumed to be disjoint and have a single task that produces the corresponding final product. Several units may be capable to carry out a task with possibly different processing times, however, task assignment has to be unique, i.e., in the final schedule a task has to be executed on exactly one unit.

Each dependency between tasks is associated with an intermediate material, which has to be stored until the consecutive task in the production is ready to be executed. Storage time may be limited and a dedicated storage unit is either available or not. In the latter case, the intermediate may be stored in the unit that is assigned to the task producing it. This storage in the processing unit contributes to the storage time of the intermediate and keeps the unit unavailable for other tasks. If a task produces several intermediates required by different tasks, these intermediates may be transferred to the assigned units of these subsequent tasks at different times. Storing of input intermediates of a task in the processing unit before executing it is not allowed and shared storage units are not considered. Products are assumed to be shipped as soon as they are produced, i.e., their storage is not a concern in the investigated problem class.

Naturally, feasible schedules have to adhere to the timing of production dependencies and a unit may not be assigned to more than one task at a time. The only timing parameter considered is the processing time, that is given for any suitable task-unit pair. Other timing related parameters, such as cleaning-, transfer-, changeover- and arrival times, due dates, etc. have no relation to how the presented approaches tackle storage time limitations. Addressing them can be done easily and independently in the S-graph framework, as presented in several papers in the literature (Adonyi et al. 2008; Hegyháti 2015; Holczinger et al. 2019), thus, they are disregarded in the rest of the paper.

3.1 Formal problem data

The input data is formally given by the following sets and parameters. In order to keep the notation as simple as possible, it is assumed—without the loss of generality—that only a single batch is to be produced from each product. This can be achieved simply by having sufficient number of copies of a product.¹ Moreover, several redundant set definitions are derived to further simplify later notations.

$\mathcal{P}$ is a finite set of products
 $\mathcal{I}_p$ is the finite set of tasks needed to be carried out to produce a single batch of product $p \in \mathcal{P}$, where $\mathcal{I}_p \cap \mathcal{I}_{p'} = \emptyset$ if $p \neq p'$

¹ This does not reflect the actual implementation, as many S-graph algorithms apply the acceleration technique proposed for multiple batches of a single product (Holczinger et al. 2002).

Fig. 5 Illustrative example

- $\mathcal{I}$ = $\bigcup_{p \in \mathcal{P}} \mathcal{I}_p$ is a derived notation for the set of all tasks, of all products
 $\mathcal{I}_p^- \subset \mathcal{I}_p$ is the finite set of tasks that are a prerequisite of task $i \in \mathcal{I}_p$, $p \in \mathcal{P}$
 $\mathcal{I}_i^+ = \{i^+ \in \mathcal{I}_p \mid i \in \mathcal{I}_p^-\}$ is a derived notation for the finite set of tasks that depend on task $i \in \mathcal{I}_p$, $p \in \mathcal{P}$
 $i_p^* \in \mathcal{I}_p$ is the task that produces the final product, the only task $i \in \mathcal{I}_p$ for which $\mathcal{I}_i^+ = \emptyset$
 $\mathcal{S} = \{(i, i^+) \mid i \in \mathcal{I}_p^-, i^+ \in \mathcal{I}\}$ is a derived notation used for the finite set of intermediates that may need storage, represented by the pair of tasks that produce and consume it
 $\mathcal{S}^{LW} \subseteq \mathcal{S}$ is the set of intermediates that have limitations on their storage time
 $st_{i,i^+}^{max} \in [0, \infty[$ is the upper time limit on storing the intermediate between the tasks $(i, i^+) \in \mathcal{S}^{LW}$
 $\mathcal{S}^{UW} = \mathcal{S} \setminus \mathcal{S}^{LW}$ is a derived notation for the set of intermediates that have no limitation on their storage time
 $\mathcal{S}^{UIS} \subseteq \mathcal{S}$ is the set of intermediates that have dedicated storage unit
 $\mathcal{S}^{NIS} = \mathcal{S} \setminus \mathcal{S}^{UIS}$ is a derived notation for the set of intermediates that have no dedicated storage unit
 $\mathcal{J}$ is the finite set of processing units available in the facility
 $\mathcal{J}_i$ is the non-empty finite set of units that can carry out the task $i \in \mathcal{I}$
 $\mathcal{I}_j = \{i \in \mathcal{I} \mid j \in \mathcal{J}_i\}$ is a derived notation for the finite set of tasks that can be carried out by the unit $j \in \mathcal{J}$
 $pt_{i,j} \in [0, \infty[$ is the processing time of task $i \in \mathcal{I}$ in unit $j \in \mathcal{J}_i$

To help in understanding the notations, an illustrative example is presented. Figure 5 shows the recipe of this example, which is a modified version of the motivational example of Sect. 2. It also contains 3 products (A, B, C), and products A and B has the same tasks like in the motivation example but task $i1$ can be performed by 2 processing units in 4 h. Furthermore, the recipe of product C has 4 tasks and it is not sequential. Task $i6$ generates 2 intermediates and task $i9$ has 2 prerequisite tasks ($i7$, $i8$). Moreover, task $i6$ can be performed by two units ($j1, j2$) and the processing time of task $i6$ is 8 and 9 h using $j1$ and $j2$, respectively. Figure 5 does not contain the storage information of the system. There are different storage policies for production of different products. The policies are NIS/LW with 2 h limit, NIS/ZW, and UIS/

Table 1 Formal description of the illustrative example shown in Fig. 5

Notation	Value
$\mathcal{P}$	$\{A, B, C\}$
$\mathcal{I}_p$	$\mathcal{I}_A = \{i1, i2, i3\}, \mathcal{I}_B = \{i4, i5\}, \mathcal{I}_C = \{i6, i7, i8, i9\}$
$\mathcal{I}$	$\{i1, i2, i3, i4, i5, i6, i7, i8, i9\}$
$\mathcal{I}_i^-$	$\mathcal{I}_{i1}^- = \emptyset, \mathcal{I}_{i2}^- = \{i1\}, \mathcal{I}_{i3}^- = \{i2\}, \mathcal{I}_{i4}^- = \emptyset, \mathcal{I}_{i5}^- = \{i4\}, \mathcal{I}_{i6}^- = \emptyset, \mathcal{I}_{i7}^- = \{i6\}, \mathcal{I}_{i8}^- = \{i6\}, \mathcal{I}_{i9}^- = \{i7, i8\}$
$\mathcal{I}_i^+$	$\mathcal{I}_{i1}^+ = \{i2\}, \mathcal{I}_{i2}^+ = \{i3\}, \mathcal{I}_{i3}^+ = \emptyset, \mathcal{I}_{i4}^+ = \{i5\}, \mathcal{I}_{i5}^+ = \emptyset, \mathcal{I}_{i6}^+ = \{i7, i8\}, \mathcal{I}_{i7}^+ = \{i9\}, \mathcal{I}_{i8}^+ = \{i9\}, \mathcal{I}_{i9}^+ = \emptyset$
i_p^*	$i_A^* = i3, i_B^* = i5, i_C^* = i9$
$\mathcal{S}$	$\{(i1, i2), (i2, i3), (i4, i5), (i6, i7), (i6, i8), (i7, i9), (i8, i9)\}$
$\mathcal{S}^{LW}$	$\{(i1, i2), (i2, i3), (i4, i5)\}$
st_{i,i^+}^{max}	$st_{i1,i2}^{max} = st_{i2,i3}^{max} = 2, st_{i4,i5}^{max} = 0$
$\mathcal{S}^{UW}$	$\{(i6, i7), (i6, i8), (i7, i9), (i8, i9)\}$
$\mathcal{S}^{UIS}$	$\{(i6, i7), (i6, i8), (i7, i9), (i8, i9)\}$
$\mathcal{S}^{NIS}$	$\{(i1, i2), (i2, i3), (i4, i5)\}$
$\mathcal{J}$	$\{j1, j2, j3, j4\}$
$\mathcal{J}_i$	$\mathcal{J}_{i1} = \{j1, j2\}, \mathcal{J}_{i2} = \{j3\}, \mathcal{J}_{i3} = \{j4\}, \mathcal{J}_{i4} = \{j3\}, \mathcal{J}_{i5} = \{j4\}, \mathcal{J}_{i6} = \{j1, j2\}, \mathcal{J}_{i7} = \{j3\}, \mathcal{J}_{i8} = \{j3\}, \mathcal{J}_{i9} = \{j4\}$
$\mathcal{I}_j$	$\mathcal{I}_{j1} = \{i1, i6\}, \mathcal{I}_{j2} = \{i1, i6\}, \mathcal{I}_{j3} = \{i2, i4, i7, i8\}, \mathcal{I}_{j4} = \{i3, i5, i9\}$
$pt_{i,j}$	$pt_{i1,j1} = 4, pt_{i1,j2} = 4, pt_{i2,j3} = 2, pt_{i3,j4} = 2, pt_{i4,j3} = 6, pt_{i5,j4} = 5, pt_{i6,j1} = 8, pt_{i6,j2} = 9, pt_{i7,j3} = 3, pt_{i8,j3} = 2, pt_{i9,j4} = 4$

UW, for the production of A , B and C , respectively. The formal problem data of this illustrative example is summarized in Table 1.

3.2 Formal solution

A solution to the scheduling problem of this class can be given as a mapping from $\mathcal{I}$ to triples: $i \mapsto (J_i, T_i^s, T_i^r)$, where

$J_i \in \mathcal{J}_i$ is the unit assigned to task i
 T_i^s is the time when J_i receives all input intermediates (if any), and starts executing the task i
 T_i^r is the time when the last output material of i is removed from J_i , and it is released for executing other tasks

For simpler notation below, $T_i^f = T_i^s + pt_{i,J_i}$ is introduced as the finishing time of task i in J_i .

With these notations, the requirements of a feasible solution can be expressed as:

1. A unit can only be released from a task after the task is finished, i.e., for all $i \in \mathcal{I}$:

$$T_i^r \geq T_i^f$$

2. A unit can only work on one task at any given time, i.e., for all $i, i' \in \mathcal{I}, i \neq i'$ and $J_i = J_{i'}:]T_i^s, T_i^r[\cap]T_{i'}^s, T_{i'}^r[= \emptyset$
3. Production dependencies must be adhered to, i.e., for all $(i, i^+) \in \mathcal{S}: T_{i^+}^s \geq T_i^f$
4. Intermediate materials without dedicated units must be stored in the processing units, i.e., for all $(i, i^+) \in \mathcal{S}^{NIS}: T_i^s \geq T_{i^+}^s$
5. Intermediate materials with limited storage time must not be stored longer than allowed, i.e., for all $(i, i^+) \in \mathcal{S}^{LW}: T_{i^+}^s \leq T_i^f + st_{i,i^+}^{max}$

The goal of optimization is to provide a mapping $i \mapsto (J_i, T_i^s, T_i^r)$ in such a way that it satisfies the conditions above and $\max_{i \in \mathcal{I}} T_i^r$ is minimal.

Note, that these conditions only separate the feasible schedules from the infeasible ones. Depending on the objective function, however, some set of schedules can be excluded a priori if the remaining set of solutions is guaranteed to have at least one optimal.

As an example, by intuition, there is no point in keeping a unit occupied after all of the produced intermediates are transferred. Formally, T_i^r should be at most $\max_{(i,i^+) \in \mathcal{S}} T_{i^+}^s$, and for the final tasks $T_{i_p}^r = T_{i_p}^f$. If the objective function, makespan minimization in this case, does not benefit from such idle behavior of the units, then schedules with such idleness can be correctly disregarded from the search space.

Similarly, if there is a dedicated storage to store an intermediate, storing in the processing unit is unreasonable. Even if the unit is free in the optimal schedule, the search space can be reduced by excluding such solutions and finding the one, where the intermediate was immediately transferred to the storage and the unit released. Note, that for example, having a cost for storage units, or the introduction of transfer times would make this reduction incorrect. However, the features of the proposed problem class ensure that this reduction is safe.

Following these observations it is clear that after these reductions the solutions could be reformulated as a mapping $i \mapsto (J_i, T_i^s)$ and similarly to T_i^f, T_i^r becomes only a derived value:

$$T_i^r = \begin{cases} \max_{(i,i^+) \in \mathcal{S}^{NIS}} T_{i^+}^s & \text{if there is at least one } i^+ \in \mathcal{I}_i^+ \text{ so that } (i, i^+) \in \mathcal{S}^{NIS} \\ T_i^f & \text{otherwise} \end{cases}$$

4 Brief introduction to the S-graph framework

The S-graph framework was introduced by Sanmartí et al. (2002) to solve makespan minimization problems for batch processes with fixed batch sizes, precedential recipe and UIS or NIS storage policy. The proposed approach relies on a directed graph model, the S-graph, and the algorithms that perform various operations on such graphs or a set of such graphs.

In the S-graph model of a problem, called recipe-graph, a node is assigned to each task and to each product, later referred to as task-nodes and product-nodes, respectively. Directed arcs express a) the dependencies between tasks and b) the

Fig. 6 Example: recipe-graph

connection between the products and the tasks that produce them. Such a directed arc is called recipe-arc and has a weight that is the minimal possible processing time of the task it sources from. An example is shown in Fig. 6.

In this example, 3 products, A , B and C are produced. The production of A requires execution of three consecutive tasks, task i_1 , i_2 and i_3 , respectively. Task i_1 can be performed by either unit j_1 or j_2 , i_2 can be performed only by unit j_3 and there are two units (j_2 , j_4) available for task i_3 . The production of B contains two consecutive tasks and product C has parallel tasks in the recipe.

Note, that the recipe-graph itself does not encode all the information of the scheduling problem, e.g., the processing times for different assignments, but the set of suitable units is usually indicated in the recipe-graph below the name of the task. For example the processing time of task i_1 of the example in Fig. 6 is 3 but it is not visible whether it belongs to unit j_1 , j_2 or both of them.

The formal definition of a recipe-graph is a triple, $(N, A_1, \emptyset)$, where

$N = N^T \cup N^P$ is the set of nodes, where

$$N^T = \{n_i^T \mid i \in \mathcal{I}\}$$

$$N^P = \{n_p^P \mid p \in \mathcal{P}\}$$

$$A_1 = \{(n_i^T, n_{i^+}^T, \min_{j \in \mathcal{J}_i} pt_{i,j}) \mid (i, i^+) \in \mathcal{S}\} \cup \{(n_{i_p^*}^T, n_p^P, \min_{j \in \mathcal{J}_p^*} pt_{i_p^*,j}) \mid p \in \mathcal{P}\}$$

is the set of recipe-arcs

Fig. 7 A schedule-graph of recipe-graph in Fig. 6

In the graph model, each node inherently relates to the timing of an event. For task nodes, it is the starting time of the task, and for the product nodes it is the time the product is ready and shipped. In other words, n_i^T relates to T_i^s , and n_p^P to $T_{i_p}^r$. Moreover, the arcs in A_1 express some of the conditions on the timing of these events that are necessary for a feasible solution as discussed in the Sect. 3.

The published algorithms of the S-graph framework use the problem data to construct this recipe-graph and perform various operations to find the optimal schedule. The decisions made during this process are modeled in the graph by adding additional directed arcs, called schedule-arcs, and changing the weights of recipe-arcs if a unit assignment is made. The S-graph representing one schedule of the problem in Fig. 6 is shown in Fig. 7 where unit j_2 has changeover time (2 units) between tasks i_3 and i_8 . This graph is called a schedule-graph and its mathematical model is the (N, A_1, A_2) triplet, where A_2 represents the schedule-arcs which are denoted by blue arcs.

The details about how the algorithms explore the possible schedules are presented in Sanmartí et al. (2002). Just like the recipe-arcs, the schedule-arcs also express timing conditions on the events associated with the task- and product-nodes. The rules for inserting a schedule-arc is discussed in detail in Sect. 5.

In a solution, the longest path leading to a node determines the timing of its associated event. In Fig. 8 this is indicated by bold, italic numbers for each node, and the longest path is shown with bold arcs for product-node B. Based on these, the Gantt chart of the solution can be plotted easily as shown in Fig. 9.

The S-graph model detects infeasible schedules by finding a cycle in the graph.

Fig. 8 The longest path of schedule-graph in Fig. 7

Fig. 9 The Gantt diagram of solution in Fig. 8

5 Modeling approaches for limited storage time

As mentioned in Sect. 4, the arcs express timing differences and the longest path to a vertex is the timing of the associated event (starting of a task for a task-node, or shipping of a (by)product for a product-node). There are a few branching techniques for different problem classes published in the S-graph framework (Sanmartí et al. 2002; Adonyi et al. 2007; Ősz and Hegyháti 2018), but all of them rely on expressing scheduling decisions via the schedule-arcs. Regardless of the branching technique, the insertion of schedule-arcs is triggered by making the decision, that task i will be carried out before task i' in the same unit $J_i = J_{i'}$.

This decision immediately implies that $T_{i'}^s$ must be at least $T_i^f = T_i^s + pt_{i,J_i}$ to satisfy rule 2 of a feasible solution. This can easily be expressed by an arc leading from n_i^T to $n_{i'}^T$ with the weight of pt_{i,J_i} . This type of arc is often referred to as an UIS arc as it is enough to express necessary conditions (rule 1 and 2) if i is the

Fig. 10 Redundant UIS arc

final task or all the outputs of task i are materials with dedicated storage since that implies $T_i^r = T_i^f$.

Task i , however, may have several outputs without storage, i.e., $\{i^+ \in \mathcal{I}_i^+ \mid (i, i^+) \in \mathcal{S}^{NIS}\} \neq \emptyset$. For each of these tasks, i^+ , their execution must start before $J_i = J_{i'}$ can be released from task i and starts working on i' . In other words, $T_{i'}^s \geq T_i^r \geq T_{i^+}^s$ for all $i^+ \in \mathcal{I}_i^+$, $(i, i^+) \in \mathcal{S}^{NIS}$. This can be expressed by several 0-weighted arcs leading from $n_{i^+}^T$ to $n_{i'}^T$, which are often called NIS arcs.

Note, that for most of the branching algorithms the unit assignment decision on a task is made together with a sequencing decision. The decision to assign unit J_i to i is modeled by updating all recipe-arcs starting from i to have the weight of pt_{i,J_i} instead of $\min_{j \in \mathcal{J}_i} pt_{i,j}$.

Since there is a recipe-arc leading from n_i^T to $n_{i^+}^T$ with the weight of pt_{i,J_i} , the UIS arc is rendered redundant if at least one NIS arc is inserted. Regardless, an illustration is given in Fig. 10, where 2 of the 3 intermediates produced by i_5 do not have a storage (inputs of tasks i_7 and i_8), and the unit j_2 will perform task i_{12} later than i_5 . Since $pt_{i_5,j_2} = 5$, if there is a path (not including i_5) to i_7 or i_8 , that is at least 5 units longer than the one leading to i_5 (in the example it is 9 units longer to i_7), the 0 weighted NIS arc from $n_{i_7}^T$ or $n_{i_8}^T$ will express a stronger lower limit on $T_{i_{12}}^s$.

These arcs can sufficiently express requirements 1-4 of a feasible solution from Sect. 3. Rule 5 requires, that $T_{i^+}^s \leq T_i^f + st_{i,i^+}^{max}$ for all $(i, i^+) \in \mathcal{S}^{LW}$. In each branching step it has to be checked, whether such a constraint is violated or not. For this purpose, several alternative approaches are available that will be discussed in the following sections.

5.1 Negatively weighted LW arcs

$T_{i^+}^s \leq T_i^f + st_{i,i^+}^{max}$ can be reformulated as $T_i^f \geq T_{i^+}^s - st_{i,i^+}^{max}$ and by using $T_i^f = T_i^s + pt_{i,J_i}$:

$$T_i^s \geq T_{i^+}^s - st_{i,i^+}^{max} - pt_{i,J_i}$$

This constraint can be expressed by adding an arc from $n_{i^+}^T$ to n_i^T with the negative weight of $-st_{i,i^+}^{max} - pt_{i,J_i}$. This type of arcs will be referred to as LW arcs. Since J_i is not known in the beginning, instead of pt_{i,J_i} , $\max_{j \in J_i} pt_{ij}$ is used until the selection is made. The branching functions do not need any further alterations. The bounding function, however, needs adjustments. In the S-graph framework, the bounding function serves two purposes: giving a lower bound on the makespan of the solutions included in that subproblem and detecting infeasibilities. The default and most often used bounding function is the longest path algorithm, which in the original case also reports infeasibility by finding directed cycles. By introducing the aforementioned LW arcs, even the recipe-graph, the root of the B&B tree also has cycles. The bounding function has to be altered not to report these cycles and still provide the longest path in the graph.

Similarly to the original case, a cycle with positive length still means infeasibility and has to be reported. A cycle with negative total weight is acceptable, does not influence the longest path and need not to be reported. Cycles with 0 total length can be twofold: if it consists of NIS arcs, it indicates an infeasibility called cross-transfer (Hegyháti et al. 2009; Ferrer-Nadal et al. 2008), otherwise if it has LW arcs in it, it only indicates zero waiting for intermediates.

Although the longest path in this graph can still be found efficiently, in the implementation, the longest paths between vertices of the S-graphs are cached in a difference bound matrix (Dill 1990). The DBM is a $|N| \times |N|$ matrix, where the k th cell in the l th row contains the longest path from the l th vertex to the k th. This matrix provides a constant time access to any longest path and can easily be updated if a new assignment or sequencing decision is made.

This approach will be referred to as the NWA approach later.

5.2 Simple LP model

Another approach is to completely replace the bounding procedure with an LP model, which expresses the constraints of the recipe and the scheduling decisions made so far. In this LP model, there would be $|Z| + |\mathcal{P}| + 1$ non-negative continuous variables, one for the starting time of each task, x_i , one for the shipping time of each product, x_p , and one for the makespan, that is to be minimized, MS . In other words, a continuous x variable will be assigned to each node of the S-graph. To make descriptions simpler, notations x_i and $x_{n_i^T}$ will be used interchangeably, and similarly, x_p and $x_{n_p^P}$ will refer to the same variable.

The model would consist of 3 types of constraints. For each arc $(n, n', w_{n,n'})$ in the S-graph, a constraint in the form of $x_{n'} \geq x_n + w_{n,n'}$ is added to the LP model, regardless of the type of the arc (schedule or recipe), or the connected nodes (task or product). For each intermediate, $(i, i^+) \in \mathcal{S}^{LW}$ with LW policy, a constraint in the form of $x_{i^+} \leq x_i + \max_{j \in \mathcal{J}_i} pt_{i,j} + st_{i,i^+}^{max}$ is added, if the unit assignment for i has not yet been made. Otherwise, the constraint would take the form $x_{i^+} \leq x_i + pt_{i,j_i} + st_{i,i^+}^{max}$. For each product, $p \in P$, $MS \geq x_p$ is added.

The solution of this LP model provides a proper bound for the makespan and the subproblem is infeasible if there is no solution to it.

The model can also be extended with a $MS \leq MS^{cb}$ constraint, where MS^{cb} is the best found makespan, i.e., the upper bound in the overall B&B procedure. In this case, an infeasible LP model does not necessarily indicate an infeasible branch, only a suboptimal one, which needs to be pruned regardless. Adding this constraint keeps the LP solver from optimizing a bound already known to be worse than a solution found previously.

In this approach, the second type of constraints is equivalent to the LW arcs of the previous approach. In this sense, if the previous approach is cached with a DBM, there is no algorithmic benefit of using this LP model as it will not provide better bounds but definitely uses more computation to end up with the same result. Thus, this approach is more like a baseline for the more complicated LP bounding functions described in the next section and will be referred to as the *SLP* approach.

5.3 Relaxed MILP model

To compensate the additional computational burden of constructing and solving an LP model for bounding, the solution should provide tighter bounds. This could be achieved by a more complex LP model that is based on the relaxation of a general precedence based MILP formulation. The general idea is to keep an MILP model in sync with the decisions made in the S-graph B&B algorithm, i.e., if some decisions are made in the B&B tree, those decisions are implemented in the MILP model by fixing the value of key binary variables. Then, the bounding procedure would be the solution of the relaxed version of this MILP model. Since this model has assignment and sequencing decisions yet to be made in a relaxed form, there is a hope for providing better bounds than the previous simple LP model.

These models generally employ continuous variables ST_i, CT_i for the start and completion of task $i \in \mathcal{I}$, binary variables $Y_{i,j}$ for the assignment of unit $j \in \mathcal{J}_i$ to task $i \in \mathcal{I}$. Binary sequencing variables can be of the form $X_{i,i'}$ for tasks $i, i' \in \mathcal{I}$, $i \neq i'$ that may be assigned to the same unit, i.e., $\mathcal{J}_i \cap \mathcal{J}_{i'} \neq \emptyset$, or directly specified for each unit $j \in \mathcal{J}_i \cap \mathcal{J}_{i'}$ as $X_{i,j,i'}$. Moreover, an additional continuous variable, MS is used for the objective function.

There are several ways how these models can be formulated as it will be discussed at the end of the section. The general procedure for these approaches starts with generating the MILP model for the root of the S-graph B&B tree. Then, when a child subproblem is generated, the MILP model is updated with the decision made.

Precedence based models and S-graph have nearly identical search space as both methodologies address the scheduling problem as assignment and sequencing decisions and derive timing from those, as opposed to time slot and time point based models, which consider assignment and timing as the major decisions and derive sequencing based on those. As a result, a decision in the S-graph B&B tree can easily be expressed with setting some binary variables of a precedence based model. For example, when the Equipment-based B&B algorithm makes a decision to set i as the next task for the unit j , this decision could be reflected as setting:

- $Y_{i,j}$ to 1
- $Y_{i,j'}$ to 0 for all $j' \in \mathcal{J}_i \setminus \{j\}$
- $X_{i',i}$ to 1 and $X_{i,i'}$ to 0 for all $i' \in \mathcal{I}_j$ that has been assigned to j previously
- $X_{i,i'}$ and $X_{i',i}$ to 0, if i' is already assigned to a different unit or $j \notin \mathcal{J}_{i'}$

There are two ways, how this update can be implemented: either a copy of the MILP model is made and the alterations are done there, or only a single MILP model is kept in memory and only the individual lower and upper bounds for the binary variables are copied and updated. The latter approach requires less memory usage and copy operations.

Moreover, the solution of the LP problem of the child subproblem may be accelerated by using the solution of the parent node as the initial basis for the dual simplex method.

5.3.1 Shared parts of considered MILP models

The considered three MILP models share most of the constraints, only the ones related to sequencing may differ.

The first three constraints express the timing considerations of the production recipe:

$$CT_i = ST_i + \sum_{j \in \mathcal{J}_i} pt_{i,j} \cdot Y_{i,j} \quad \forall i \in \mathcal{I} \quad (1)$$

$$ST_{i^+} \geq CT_i \quad \forall (i, i^+) \in \mathcal{S} \quad (2)$$

$$ST_{i^+} \leq CT_i + st_{i,i^+}^{max} \quad \forall (i, i^+) \in \mathcal{S} \quad (3)$$

The next constraint ensures that exactly one unit is selected for each task:

$$\sum_{j \in \mathcal{J}_i} Y_{i,j} = 1 \quad \forall i \in \mathcal{I} \quad (4)$$

The constraints for the objective variable, MS , are as follows:

$$MS \geq CT_i \quad \forall i \in \mathcal{I} \quad (5)$$

$$MS \leq MS^{cb} \quad (6)$$

Each of the three MILP models have these constraints as a basis, and they are extended with the ones detailed below.

5.3.2 RLP1: three index precedence model

The first selected MILP model employs $X_{i,j,i'}$ variables defined for all $i, i' \in \mathcal{I}, i \neq i', j \in \mathcal{J}_i \cap \mathcal{J}_{i'}$. $X_{i,j,i'} = 1$ if and only if both i and i' are assigned to unit j , which performs i before i' . In all other cases, i.e., if the order is the other way around, or at least one of them is not assigned to unit j , the variable should take the value of 0.

The effects of sequencing decisions are expressed by the following two constraints:

$$\begin{aligned} ST_{i'} &\geq CT_i - M \cdot (1 - X_{i,j,i'}) \\ \forall i, i' \in \mathcal{I}, i \neq i', j \in \mathcal{J}_i \cap \mathcal{J}_{i'}, \nexists (i, i^+) \in \mathcal{S}^{NIS} \end{aligned} \quad (7)$$

$$\begin{aligned} ST_{i'} &\geq ST_{i^+} - M \cdot (1 - X_{i,j,i'}) \\ \forall (i, i^+) \in \mathcal{S}^{NIS}, i' \in \mathcal{I} \setminus \{i, i^+\}, j \in \mathcal{J}_i \cap \mathcal{J}_{i'} \end{aligned} \quad (8)$$

Technical constraints that ensure the proper relation between sequencing and assignment variables:

$$X_{i,j,i'} + X_{i',j,i} \geq Y_{i,j} + Y_{i',j} - 1 \quad \forall i, i' \in \mathcal{I}, i \neq i', j \in \mathcal{J}_i \cap \mathcal{J}_{i'} \quad (9)$$

$$X_{i,j,i'} + X_{i',j,i} \leq Y_{i,j} \quad \forall i, i' \in \mathcal{I}, i \neq i', j \in \mathcal{J}_i \cap \mathcal{J}_{i'} \quad (10)$$

5.3.3 RLP2: two index precedence model

The second MILP model employs sequencing variables with two indices, i.e., $X_{i,i'}$ defined for all $i, i' \in \mathcal{I}, i \neq i', \mathcal{J}_i \cap \mathcal{J}_{i'} \neq \emptyset$. The general difference is that only one sequencing variable is made for each possibly overlapping task pair instead of a separate one for each unit where it can happen. $X_{i,i'} = 1$ if both i and i' are assigned to the same unit, whichever it may be, and i is performed before i' . In all other cases, the variable is allowed to take any value.

The following two constraints express sequencing:

$$\begin{aligned} ST_{i'} &\geq CT_i - M \cdot (1 - X_{i,i'}) \\ \forall i, i' \in \mathcal{I}, i \neq i', \mathcal{J}_i \cap \mathcal{J}_{i'} \neq \emptyset, \nexists (i, i^+) \in \mathcal{S}^{NIS} \end{aligned} \quad (11)$$

$$\begin{aligned} ST_{i'} &\geq ST_{i^+} - M \cdot (1 - X_{i,i'}) \\ \forall (i, i^+) \in \mathcal{S}^{NIS}, i' \in \mathcal{I} \setminus \{i, i^+\}, \mathcal{J}_i \cap \mathcal{J}_{i'} \neq \emptyset \end{aligned} \quad (12)$$

Note that these constraints are not generated for all of the possible conflicting units.

There is only one technical constraint:

$$X_{i,i'} + X_{i',i} \geq Y_{i,j} + Y_{i',j} - 1 \quad \forall i, i' \in \mathcal{I}, i \neq i', j \in \mathcal{J}_i \cap \mathcal{J}_{i'} \quad (13)$$

5.3.4 RLP3: two index precedence model variant

The third model that was implemented is a slightly modified version of the second one. It also employs the $X_{i,i'}$ variables but their meaning is altered. $X_{i,i'} = 1$ if and only if i is started earlier than i' regardless of their assigned units.²

A technical constraint to ensure this is:

$$X_{i,i'} + X_{i',i} = 1 \quad \forall i, i' \in \mathcal{I}, i \neq i', \mathcal{J}_i \cap \mathcal{J}_{i'} \neq \emptyset \quad (14)$$

Sequencing is then expressed via the following constraints generated for all of the possible conflicting units, just in the case for the first model:

$$\begin{aligned} ST_{i'} &\geq CT_i - M \cdot (3 - Y_{i,j} - Y_{i',j} - X_{i,i'}) \\ \forall i, i' \in \mathcal{I}, i \neq i', j \in \mathcal{J}_i \cap \mathcal{J}_{i'}, \nexists (i, i^+) \in \mathcal{S}^{NIS} \end{aligned} \quad (15)$$

$$\begin{aligned} ST_{i'} &\geq ST_{i^+} - M \cdot (3 - Y_{i,j} - Y_{i',j} - X_{i,i'}) \\ \forall (i, i^+) \in \mathcal{S}^{NIS}, i' \in \mathcal{I} \setminus \{i, i^+\}, j \in \mathcal{J}_i \cap \mathcal{J}_{i'} \end{aligned} \quad (16)$$

The key advantage of this model is, that constraint (14) allows to reduce the number of sequencing variables by half. Since $X_{i,i'} + X_{i',i} = 1$ for all i and i' that can share a unit, in essence, only one of those variables are “free” and the other is “fixed” or “derived” from the former one. This means, that one of those variables can be replaced with a simple linear expression of the other, e.g.: $X_{i,i'} = (1 - X_{i',i})$, in all of the constraints where it appears.

5.4 Other techniques

During our investigation, several other ideas came up and were implemented. Two of them are briefly introduced below. These ideas were later disregarded, as they turned out to be algorithmically inferior compared to the negatively weighted LW arc approach (Hegyháti 2015).

5.4.1 Recursive search

Instead of inserting the negatively weighted arcs into the graph and altering the bounding procedure, S-graph could still reflect the UW case and use the original bounding function. In order to find LW/ZW infeasibilities, a recursive search

² If i and i' start at exactly the same time, one of the variables is set to 0 and the other to 1, which does not influence the soundness of the model.

Table 2 Processing times [τ_u] of tasks for examples E1 (products 1..11) and E7 (products 1..9)

Task	Unit	Product										
		1	2	3	4	5	6	7	8	9	10	11
1	1	5	8	22	16	17	7	8	12	14	17	22
	2	4	11	5	20	8	8	21	4	17	14	6
2	3	17	29	22	18	20	17	27	25	21	23	21
	4	24	26	18	20	23	17	21	24	26	20	19

Table 3 Processing times [τ_u] of tasks for examples E2 (products 1..9) and E8 (products 1..10)

Task	Unit	Product									
		1	2	3	4	5	6	7	8	9	10
1	1	12	16	7	15	12	4	11	16	6	7
	2	14	12	5	5	8	19	2	12	2	19
2	3	51	17	24	62	53	21	34	37	20	55
	4	63	50	43	44	34	37	36	64	42	17
	5	17	41	39	33	21	23	35	24	70	25

algorithm could be used, which tests, whether there is a forced upper bound on the starting time of a task, that is lower than another forced lower bound.

This approach provides less tight bounds and requires additional attention at leaf nodes to evaluate the makespan.

5.4.2 Conversion to ZW and external graph

In the case of $st_{i,i^+}^{max} = 0$, the intermediate between i and i^+ cannot be stored at all and must be processed immediately. This renders the difference between the starting times of i and i^+ fixed. This also means, if i^+ is forced to start later due to unit scheduling, so does i .

To tackle ZW cases, one can build an auxiliary graph, whose nodes represent these strongly connected vertex sets. New decisions not only insert arcs in the S-graph but to this auxiliary graph as well. Non-negative directed cycles will indicate infeasible solutions.

Problems with more general, LW constraints can be converted to equivalent problems with ZW and UW vertices only (Hegyháti et al. 2011).

6 Empirical results

The performance of the proposed approach was tested on several examples, which have been introduced by Liu and Karimi (2007). They examined multistage multiproduct batch plants with nonidentical parallel units and generated 13 problems. In this type of plants the products are processed sequentially with the same order of stages and more than one unit is available for each stage (task). The processing

Table 4 Processing times [τ_u] of tasks for examples E3 (products 1..8) and E9 (products 1..9)

Task	Unit	Product								
		1	2	3	4	5	6	7	8	9
1	1	23	5	18	15	10	2	11	16	8
	2	11	12	12	10	20	8	22	4	5
	3	15	11	22	22	11	11	17	22	10
2	4	9	9	10	7	8	6	12	11	20
	5	7	19	10	15	8	17	8	11	25
	6	16	15	19	16	7	6	8	17	10
	7	8	8	11	14	11	13	7	15	15

Table 5 Processing times [τ_u] of tasks for examples E4 (products 1..7) and E10 (products 1..9)

Task	Unit	Product								
		1	2	3	4	5	6	7	8	9
1	1	15	6	12	16	16	12	4	17	11
	2	5	2	14	12	12	8	19	11	2
2	3	62	20	51	17	37	53	21	25	17
	4	44	42	63	50	64	34	37	30	40
3	5	16	15	23	19	6	16	32	11	25
	6	26	32	19	28	19	7	29	16	9
	7	21	17	15	33	25	25	8	29	15

Table 6 Processing times [τ_u] of tasks for examples E5 (products 1..8), E11 (products 1..9) and E13 (products 1..10)

Task	Unit	Product									
		1	2	3	4	5	6	7	8	9	10
1	1	15	6	12	16	16	4	7	12	8	7
	2	5	2	14	12	12	19	5	8	17	15
	3	3	3	2	13	14	11	6	18	8	15
2	4	62	20	51	17	37	21	24	53	30	19
	5	44	42	63	50	64	37	43	34	21	18
3	6	16	15	23	19	6	32	29	16	30	35
	7	26	32	19	28	19	29	22	7	30	11
	8	21	17	15	33	25	8	28	25	30	14
	9	29	18	13	16	24	28	10	5	15	27
	10	21	28	23	16	7	33	11	20	17	11

times of the tasks in the examples are given in Tables 2, 3, 4, 5, 6 and 7. E.g. Table 3 shows that in example E2, 9 products have to be produced in 2 steps, where the first task can be performed by unit 1 or 2 and the second task by unit 3, 4 or 5.

Liu and Karimi (2007) assumed unlimited intermediate storage and unlimited wait (UIS/UW) policy and later they also examined the NIS/ZW case solving the

Table 7 Processing times [tu] of tasks for examples E6 (products 1..5) and E12 (products 1..6)

Task	Unit	Product					
		1	2	3	4	5	6
1	1	16	17	15	5	17	11
	2	22	7	19	13	6	7
	3	15	11	5	6	7	11
2	4	10	13	10	13	8	15
	5	13	14	7	11	10	15
3	6	13	9	11	12	20	25
	7	18	10	17	10	14	17
	8	18	19	11	17	18	16
4	9	15	21	28	10	26	15
	10	23	21	24	22	14	15

same examples (Liu and Karimi 2008). In this paper all examples were solved with UIS/UW, UIS/ZW, NIS/UW and NIS/ZW policies and got the same optimum for all cases.

The tests were run on an Intel Core i5-660 3.33 GHz CPU with 4 GB RAM with 3600 s time limit. In the tables the following notations are used for the methods:

- UW: UIS/NIS UW with the original S-graph framework for reference (Sanmartí et al. 2002)
- NEG: negative arcs approach (Sect. 5.1)
- SLP: simple LP bound (Sect. 5.2)
- RLP{1,2,3}{c,s}: bounding with relaxed MILP models (Sect. 5.3)
 - c: copy the model when branching and start from the parent solution (SLP also works like this)
 - s: use a shared singleton LP model

The CPU time of the program was limited to 3600 s, after this time the algorithm was stopped and the current best solution is presented. Tables 8, 9, 10, 11 show the objective values (tu) and the CPU times (s) of the algorithms to solve the 13 examples. In UW case, UIS algorithm cannot find the optimum of examples E10 and E11 but for other examples the CPU times of UIS and NIS are close to each other. In ZW case, the negative arc algorithm was always the fastest with the same speed as UW/ NIS algorithm.

The previous examples were multiproduct problems, where products visit the same stages in the same order. However, the presented methods can solve the more general multipurpose case, where the stage order can be arbitrary for each product. To investigate the effect of the recipe type on the performance, multipurpose examples have been generated from the previous problems by reversing the stage order of the first 4 products in each instance. UIS policy was assumed at each stage. It can be

Table 8 Objective values [τ_u] for NIS policy (bold value means non optimal solution)

Ex.	UW	ZW							
		NEG	SLP	RLP1c	RLP2c	RLP3c	RLP1s	RLP2s	RLP3s
E1	118	118	120	120	120	120	120	120	120
E2	84	84	84	84	84	84	84	84	84
E3	33	33	33	33	33	33	33	33	33
E4	133	133	133	133	133	133	133	133	133
E5	148	148	148	148	154	148	148	148	148
E6	72	72	72	72	72	72	72	72	72
E7	97	97	97	97	97	97	97	97	97
E8	89	89	89	109	89	89	101	89	89
E9	33	33	38	38	38	38	38	38	38
E10	153	153	153	155	157	158	155	158	153
E11	154	154	154	155	155	154	155	154	154
E12	76	76	76	76	76	76	76	76	76
E13	162	162	162	166	166	162	166	166	166

Table 9 CPU times [s] for NIS policy (bold value means the time limit was reached)

Ex.	UW	ZW							
		NEG	SLP	RLP1c	RLP2c	RLP3c	RLP1s	RLP2s	RLP3s
E1	1269	1252	3600	3600	3600	3600	3600	3600	3600
E2	2.38	2.35	26.03	431.1	337.3	180.6	491.2	234.6	221.7
E3	8.74	8.93	93.14	1474	1111	399.2	1664	1037	466.4
E4	0.23	0.22	1.59	23.52	78.52	13.02	25.40	17.41	10.59
E5	1.36	1.32	9.98	235.9	3600	84.44	253.8	210.4	97.34
E6	0.19	0.18	1.11	7.39	5.38	48.86	8.43	5.73	6.63
E7	9.58	9.42	109.7	1616	995.3	667.9	1707	885.7	721.2
E8	27.74	27.9	331.4	3600	3600	2364	3600	2908	2576
E9	1578	1559	3600	3600	3600	3600	3600	3600	3600
E10	135.2	138.1	1738	3600	3600	3600	3600	3600	604.8
E11	500	499.2	3329	3600	3600	612.2	3600	1725	781.8
E12	2.31	2.30	19.63	151.9	160.6	430.7	220.7	167.1	264.5
E13	76.97	76.85	567.6	3600	3600	3600	3600	3600	3600

seen from the test results that the methods perform similarly for multipurpose examples as it was seen in the multiproduct case (Tables 12, 13).

Liu and Karimi (2008) presented a hybrid plant example, where both NIS, UIS, UW, and ZW policies occur throughout the recipe. In the example there are 4 stages, and there is NIS/UW policy between tasks 1 and 2, NIS/ZW after task 2, and UIS/UW between tasks 3 and 4. The processing times of the tasks are given in Table 14. The

Table 10 Objective values [τ_u] for UIS policy (bold value means non optimal solution)

Ex.	UW	ZW							
		NEG	SLP	RLP1c	RLP2c	RLP3c	RLP1s	RLP2s	RLP3s
E1	118	118	120	120	120	120	120	122	120
E2	84	84	84	84	84	84	84	84	84
E3	33	33	33	33	33	33	33	33	33
E4	133	133	133	133	145	133	133	133	133
E5	148	148	148	148	148	148	148	148	148
E6	72	72	72	72	72	72	72	72	72
E7	97	97	97	97	97	97	97	97	97
E8	89	89	89	89	89	89	89	89	89
E9	33	33	38	38	38	38	38	38	38
E10	158	158	158	158	158	158	158	155	155
E11	155	155	155	155	155	155	155	154	156
E12	76	76	76	76	76	76	76	76	79
E13	162	162	162	166	166	163	166	166	168

Table 11 CPU times [s] for UIS policy (bold value means the time limit was reached)

Ex.	UW	ZW							
		NEG	SLP	RLP1c	RLP2c	RLP3c	RLP1s	RLP2s	RLP3s
E1	1163	1193	3600	3600	3600	3600	3600	3600	3600
E2	2.25	2.22	25.3	379.56	288.35	156.97	414.69	223.57	202.66
E3	10	9.65	108.84	1018	790.4	434	1631	560.9	535.6
E4	0.66	0.63	5.73	64.28	3600	28	59.09	41.65	34.85
E5	1.28	1.29	9.95	234.6	206.5	82.32	194.9	150.8	93.27
E6	0.22	0.22	1.47	57.95	8.4	6.13	11.48	7.89	8.30
E7	9.20	9.24	112	1649	1023	647.8	1896	850.8	849.4
E8	13.55	13.35	160.9	2809	2099	1158	3600	1644	1637
E9	2222	2110	3600	3600	3600	3600	3600	3600	3600
E10	3600	3600	3600	3600	3600	3600	3600	3600	3600
E11	3600	3600	3600	3600	3600	3600	3600	1056	3600
E12	10.89	10.82	104.3	907.3	686.9	429.5	1028	718.4	3600
E13	73.79	73.23	566.9	3600	3600	3600	3600	3600	3600

Table 12 Objective values [τ_u] for multipurpose examples (bold value means non optimal solution)

Ex.	UW	ZW							
		NEG	SLP	RLP1c	RLP2c	RLP3c	RLP1s	RLP2s	RLP3s
E1	114	114	116	120	120	119	120	120	119
E2	82	82	82	82	82	82	82	82	82
E3	29	29	29	29	29	29	29	29	29
E4	136	136	136	136	136	136	136	136	136
E5	151	151	151	151	151	151	151	151	151
E6	68	68	68	68	68	68	68	68	68
E7	93	93	93	93	93	93	93	93	93
E8	87	87	87	87	87	87	87	87	87
E9	29	29	29	31	31	31	31	31	31
E10	162	162	162	162	162	162	162	156	156
E11	152	152	152	152	152	152	152	152	152
E12	68	68	68	68	68	68	68	68	68
E13	161	161	161	164	164	166	164	164	164

Table 13 CPU times [s] for multipurpose examples (bold value means the time limit was reached)

Ex.	UW	ZW							
		NEG	SLP	RLP1c	RLP2c	RLP3c	RLP1s	RLP2s	RLP3s
E1	933.72	908.42	3600	3600	3600	3600	3600	3600	3600
E2	1.86	1.81	20.75	420.97	307.4	139.08	433.58	191.2	166.73
E3	1.47	1.49	13.57	198.24	160.09	84.74	229.49	125.62	116.63
E4	0.16	0.17	1.13	16.35	13.32	6.4	15.47	9.62	7.95
E5	1.28	1.26	9.66	222.37	178.04	88.95	198.13	131.63	90.09
E6	0.04	0.05	0.08	0.46	0.32	0.2	0.41	0.3	0.23
E7	6.9	6.77	84.36	1235.62	827.46	474.32	1176.03	656.99	520.4
E8	10.32	10.21	121.4	2650.25	1990.57	910.35	2778.57	1209.22	1119.26
E9	237.79	237.33	2623.82	3600	3600	3600	3600	3600	3600
E10	3600	3600	3600	3600	3600	3600	3600	3600	3600
E11	6.7	6.57	51.47	1605.45	1174.87	476.02	1508.45	900.61	563.18
E12	0.06	0.06	0.17	59.09	1.79	0.82	2.23	1.14	0.86
E13	70.36	68.78	528.99	3600	3600	3600	3600	3600	3600

Table 14 Processing times [tu] of tasks for the hybrid example

Task	Unit	Product				
		1	2	3	4	5
1	1	21	50	16	49	81
	2	21	50	16	49	81
2	3	59	113	69	71	121
	4	46	65	120	138	125
	5	103	86	66	97	138
3	6	98	116	80	63	100
	7	98	116	80	63	100
	8	98	116	80	63	100
4	9	52	49	62	57	75
	10	58	53	65	69	53

Table 15 CPU times [s] for the hybrid example

NEG	SLP	RLP1c	RLP2c	RLP3c	RLP1s	RLP2s	RLP3s
12.59	121.95	729	638	485.4	937.5	682.3	536.6

minimal makespan is 361 tu, which was found by all methods. The CPU times of the algorithms are given in Table 15.

7 Concluding remarks

There are plenty of advantages and disadvantages between approaches based on general purpose solvers (e.g., MILP optimizers) and those relying on customized models and algorithms. One clear benefit of the latter can be the ability to change and tune the inner workings of the optimization algorithms more easily. This paper follows that principle, and examined several different options for modeling limited-wait storage policy in the S-graph framework. These approaches were all implemented and tested extensively. The results clearly showed the superiority of the negative-weighted arc approach in most of the cases in terms of efficiency.

This unambiguous result is favored for future work as developments tackling more complex problems can build on top of this technique and have the certainty that this subtask is addressed in the best possible way.

Acknowledgements Open access funding provided by University of Pannonia. We acknowledge the financial support of Széchenyi 2020 under the EFOP-3.6.1-16-2016-00015. This research was also

supported from the Thematic Excellence Program 2019 the grant of the Hungarian Ministry for Innovation and Technology (Grant Number: NKFIH-843-10/2019).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Adonyi R, Biros G, Holczinger T, Friedler F (2008) Effective scheduling of a large-scale paint production system. *J Clean Prod* 16(2):225–232. <https://doi.org/10.1016/j.jclepro.2006.08.021>
- Adonyi R, Holczinger T, Friedler F (2007) Novel branching procedure for S-graphs based scheduling of batch processes. In: Jezowski J (ed) Proceedings of 19th Polish conference of chemical and process engineering. Oficyna Wydawnicza Politechniki Wroclawskiej, p 9
- Allahverdi A (2016) A survey of scheduling problems with no-wait in process. *Eur J Oper Res* 255(3):665–686. <https://doi.org/10.1016/j.ejor.2016.05.036>
- Allahverdi A, Pesch E, Pinedo M, Werner F (2018) Scheduling in manufacturing systems: new trends and perspectives. *Int J Prod Res* 56(19):6333–6335. <https://doi.org/10.1080/00207543.2018.1504252>
- An YJ, Kim YD, Choi SW (2016) Minimizing makespan in a two-machine flowshop with a limited waiting time constraint and sequence-dependent setup times. *Comput Oper Res* 71:127–136. <https://doi.org/10.1016/j.cor.2016.01.017>
- Balas E (1969) Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Oper Res* 17(6):941–957
- Callahan JR (1971) The nothing hot delay problem in the production of steel. Ph.D. thesis, Department of Industrial Engineering, University of Toronto, Canada
- Condotta A, Shakhlevich N (2012) Scheduling coupled-operation jobs with exact time-lags. *Discrete Appl Math* 160(16):2370–2388. <https://doi.org/10.1016/j.dam.2012.05.026>
- D'Ariano A, Pacciarelli D, Pranzo M (2007) A branch and bound algorithm for scheduling trains in a railway network. *Eur J Oper Res* 183(2):643–657. <https://doi.org/10.1016/j.ejor.2006.10.034>
- Dill DL (1990) Timing assumptions and verification of finite-state concurrent systems. In: Sifakis J (ed) Automatic verification methods for finite state systems. Springer, Berlin, pp 197–212. https://doi.org/10.1007/3-540-52148-8_17
- Ferrer-Nadal S, Capón-García E, Méndez CA, Puigjaner L (2008) Material transfer operations in batch scheduling. A critical modeling issue. *Ind Eng Chem Res* 47:7721–7732. <https://doi.org/10.1021/ie800075u>
- Ghaeli M, Bahri PA, Lee P, Gu T (2005) Petri-net based formulation and algorithm for short-term scheduling of batch plants. *Comput Chem Eng* 29(2):249–259. <https://doi.org/10.1016/j.compchemeng.2004.08.025>
- Gicquel C, Hege L, Minoux M, van Canneyt W (2012) A discrete time exact solution approach for a complex hybrid flow-shop scheduling problem with limited-wait constraints. *Comput Oper Res* 39(3):629–636. <https://doi.org/10.1016/j.cor.2011.02.017>
- Grenouilleau F, Legrain A, Lahrichi N, Rousseau LM (2019) A set partitioning heuristic for the home health care routing and scheduling problem. *Eur J Oper Res* 275(1):295–303. <https://doi.org/10.1016/j.ejor.2018.11.025>
- Hall NG, Sriskandarajah C (1996) A survey of machine scheduling problems with blocking and no-wait in process. *Oper Res* 44(3):510–525. <https://doi.org/10.1287/opre.44.3.510>
- Hegyháti M (2015) Batch process scheduling: extensions of the S-graph framework. Ph.D. thesis, University of Pannonia. <https://doi.org/10.18136/PE.2015.585>

- Hegyháti M, Friedler F (2010) Overview of industrial batch process scheduling. *Chem Eng Trans* 21:895–900. <https://doi.org/10.3303/CET1021150>
- Hegyháti M, Majozsi T, Holczinger T, Friedler F (2009) Practical infeasibility of cross-transfer in batch plants with complex recipes: S-graph vs MILP methods. *Chem Eng Sci* 64(3):605–610. <https://doi.org/10.1016/j.ces.2008.10.018>
- Hegyháti M, Holczinger T, Szoldatics A, Friedler F (2011) Combinatorial approach to address batch scheduling problems with limited storage time. *Chem Eng Trans* 25:495–500. <https://doi.org/10.3303/CET1125083>
- Holczinger T, Romero J, Puigjaner L, Friedler F (2002) Scheduling of multipurpose batch processes with multiple batches of the products. *Hung J Ind Chem* 30:263–270. <https://doi.org/10.1002/aic.10036>
- Holczinger T, Ósz O, Hegyháti M (2019) Scheduling approach for on-site jobs of service providers. *Flex Serv Manuf J*. <https://doi.org/10.1007/s10696-019-09359-2>
- Holczinger T, Hegyháti M, Friedler F (2012) Simultaneous heat integration and batch process scheduling. In: CHISA 2012—20th international congress of chemical and process engineering and PRES 2012—15th conference PRES. <https://doi.org/10.3303/CET1229057>
- Holczinger T, Majozsi T, Hegyháti M, Friedler F (2007) An automated algorithm for throughput maximization under fixed time horizon in multipurpose batch plants: S-Graph approach. In: Plesu V, Agachi PS (eds) 17th European symposium on computer aided process engineering, computer aided chemical engineering, vol 24. Elsevier, pp 649–654. [https://doi.org/10.1016/S1570-7946\(07\)80131-3](https://doi.org/10.1016/S1570-7946(07)80131-3)
- Jung JH, Lee HK, Yang DR, Lee IB (1994) Completion times and optimal scheduling for serial multi-product processes with transfer and set-up times in zero-wait policy. *Comput Chem Eng* 18(6):537–543. [https://doi.org/10.1016/0098-1354\(93\)E0009-X](https://doi.org/10.1016/0098-1354(93)E0009-X)
- Liu Y, Karimi I (2007) Scheduling multistage, multiproduct batch plants with nonidentical parallel units and unlimited intermediate storage. *Chem Eng Sci* 62(6):1549–1566. <https://doi.org/10.1016/j.ces.2006.11.053>
- Liu Y, Karimi I (2008) Scheduling multistage batch plants with parallel units and no interstage storage. *Comput Chem Eng* 32(4):671–693. <https://doi.org/10.1016/j.compchemeng.2007.02.002>
- Majozsi T, Friedler F (2006) Maximization of throughput in a multipurpose batch plant under fixed time horizon: S-graph approach. *Ind Eng Chem Res* 45:6713–6720. <https://doi.org/10.1021/ie0604472>
- Méndez CA, Cerdá J, Grossmann IE, Harjunkoski I, Fahl M (2006) State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Comput Chem Eng* 30(6–7):913–946. <https://doi.org/10.1016/j.compchemeng.2006.02.008>
- Moon S, Park S, Lee WK (1996) New milp models for scheduling of multiproduct batch plants under zero-wait policy. *Ind Eng Chem Res* 35(10):3458–3469. <https://doi.org/10.1021/ie9601458>
- Nguyen S, Zhang M, Johnston M, Tan KC (2019) Genetic programming for job shop scheduling. Springer, Cham, pp 143–167. https://doi.org/10.1007/978-3-319-91341-4_8
- Ósz O, Hegyháti M (2018) An S-graph based approach for multi-mode resource-constrained project scheduling with time-varying resource capacities. *Chem Eng Trans* 70(2017):1165–1170. <https://doi.org/10.3303/CET1870195>
- Papp Á, Ósz O, Hegyháti M (2018) Review and comparison of milp approaches for cyclic scheduling of robotic cells. In: VOCAL 2018—8th VOCAL optimization conference: advanced algorithms, pp 79–84
- Park J, Mei Y, Nguyen S, Chen G, Zhang M (2018) An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. *Appl Soft Comput* 63:72–86. <https://doi.org/10.1016/j.asoc.2017.11.020>
- Salvador MS (1973) A solution to a special class of flow shop scheduling problems. In: Elmaghraby SE (ed) Symposium on the theory of scheduling and its applications. Springer, Heidelberg, pp 83–91
- Sanmartí E, Puigjaner L, Holczinger T, Friedler F (2002) Combinatorial framework for effective scheduling of multipurpose batch plants. *AIChE J* 48(11):2557–2570. <https://doi.org/10.1002/aic.690481115>
- Schoppmeyer C, Subbiah S, Engell S (2012) Modeling and solving batch scheduling problems with various storage policies and operational policies using timed automata. In: Karimi IA, Srinivasan R (eds) 11th international symposium on process systems engineering, vol 31. Elsevier, Amsterdam, pp 635–639. <https://doi.org/10.1016/B978-0-444-59507-2.50119-0>
- Shaik MA, Floudas CA (2009) Novel unified modeling approach for short-term scheduling. *Ind Eng Chem Res* 48(6):2947–2964. <https://doi.org/10.1021/ie8010726>

- Zhang S, Wong TN (2018) Integrated process planning and scheduling: an enhanced ant colony optimization heuristic with parameter tuning. *J Intell Manuf* 29(3):585–601. <https://doi.org/10.1007/s10845-014-1023-3>
- Zhou N, Wu M, Zhou J (2018) Research on power battery formation production scheduling problem with limited waiting time constraints. In: 2018 10th international conference on communication software and networks (ICCSN), pp 497–501. <https://doi.org/10.1109/ICCSN.2018.8488247>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.